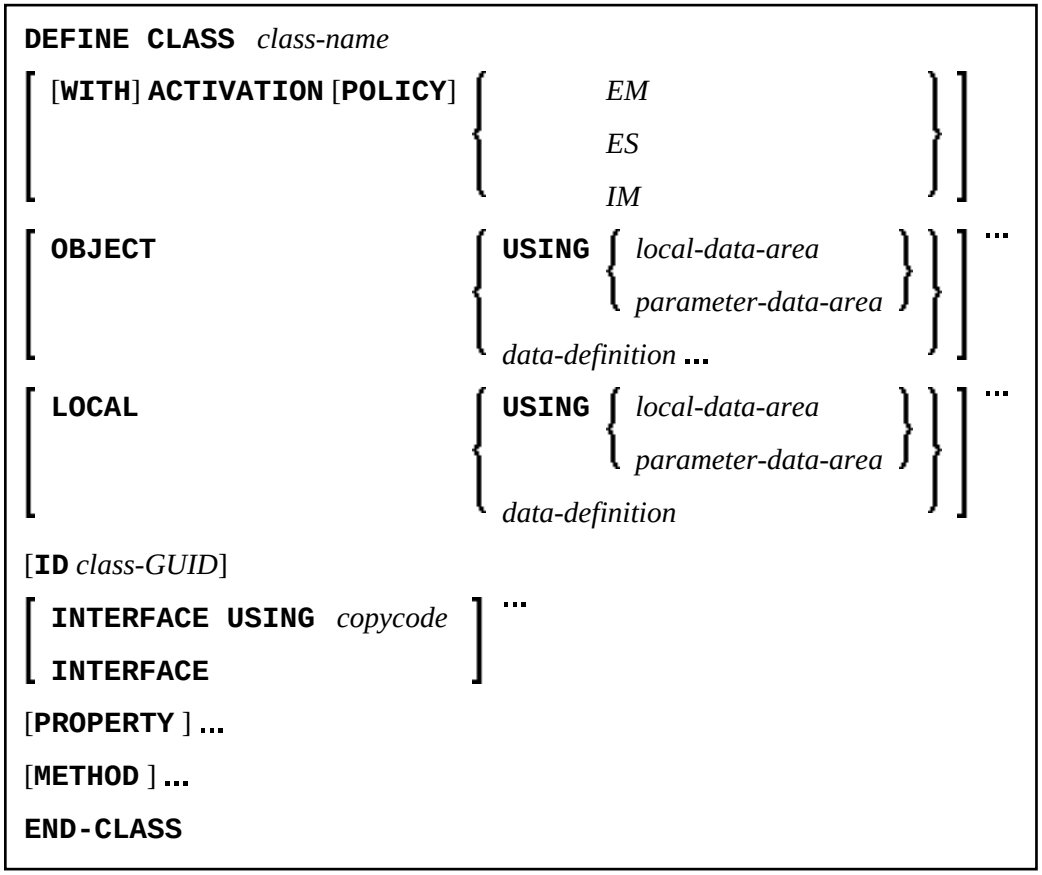
DEFINE CLASS



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` | `INTERFACE` | `METHOD` | `PROPERTY` | `SEND METHOD`

Gehört zur Funktionsgruppe: *Komponenten-basierte Anwendungen erstellen*

Funktion

Das Statement `DEFINE CLASS` dient dazu, eine Klasse innerhalb eines Natural Class-Moduls anzugeben.

Ein Natural Class-Modul besteht aus einem `DEFINE CLASS`-Statement gefolgt von einem `END`-Statement.

Syntax-Beschreibung

class-name	**Klassen-Name:** Dies ist der Name, der von Clients benutzt wird, um Objekte dieser Klasse zu erstellen. Er kann maximal bis zu 32 Zeichen lang sein und Punkte enthalten. Deshalb kann es Klassen-Namen geben wie: *company-name. application-name. class-name* Jeder Bestandteil zwischen den Punkten (...) muss den Natural-Namenskonventionen für Benutzervariablen entsprechen. Wenn die Klasse von in verschiedenen Programmiersprachen geschriebenen Clients verwendet werden soll, sollte der Klassen-Name so gewählt werden, dass er nicht gegen die in diesen Sprachen geltenden Namenskonventionen verstößt.
WITH ACTIVATION POLICY	**WITH ACTIVATION POLICY-Klausel:** Diese Klausel dient dazu, die Activation Policy zu definieren, die für die aktuelle Klasse registriert ist. Sie können folgende Parameter angeben:
	`EM` — Die Activation Policy ist `ExternalMultiple`.
	`ES` — Die Activation Policy ist `ExternalSingle`.
	`IM` — Die Activation Policy ist `InternalMultiple`.
	Wenn die Klasse mit `STOW` gespeichert und registiert wird, überschreibt die Einstellung in der `WITH ACTIVATION POLICY`-Klausel die mit dem Profilparameter `ACTPOLICY` vorgenommene Einstellung, aber sie wird ihrerseits durch die manuelle Registrierung mittels `REGISTER`-Kommando und expliziter Activation-Policy-Definition überschrieben. Weitere Informationen siehe *Activation Policies* in der *Operations*-Dokumentation.
OBJECT	**OBJECT-Klausel:** Die `OBJECT`-Klausel dient dazu, Objektdaten zu definieren. Die Syntax der `OBJECT`-Klausel entspricht der für die `LOCAL`-Klausel des `DEFINE DATA`-Statements. Weitere Informationen siehe Beschreibung der `LOCAL`-Klausel des `DEFINE DATA`-Statements.

LOCAL	**LOCAL-Klausel:** Die LOCAL-Klausel dient dazu, global eindeutige IDs (GUID = Globally Unique ID) in die Klassen-Definition aufzunehmen. GUIDs müssen nur definiert werden, wenn eine Klasse für DCOM registriert werden soll. GUIDs werden meistens in einer Local Data Area (LDA) definiert. Weitere Informationen siehe *Globally Unique Identifiers (GUIDs)* im *Leitfaden zur Programmierung*. Die Syntax der LOCAL-Klausel entspricht der für die LOCAL-Klausel des DEFINE DATA-Statements. Weitere Informationen siehe Beschreibung der LOCAL-Klausel des DEFINE DATA-Statements.
ID	**ID-Klausel:** Die ID-Klausel dient dazu, der Klasse eine GUID zuzuweisen. Die GUID der Klasse ist der Name einer in der Data Area definierten GUID, die mit der LOCAL-Klausel eingefügt wird. Die Klasse GUID ist eine (mit Namen versehene) alphanumerische Konstante. Einer Klasse muss eine GUID zugewiesen werden, wenn diese unter DCOM registriert werden soll.
INTERFACE USING	**INTERFACE USING-Klausel:** Die INTERFACE USING-Klausel wird verwendet, um einen Copycode aufzunehmen, der INTERFACE-Statements enthält.
copycode	**Copycode:** Der von der INTERFACE USING-Klausel verwendete Copycode kann eines oder mehrere INTERFACE-Statements enthalten.
PROPERTY	**PROPERTY-Statement:** Das PROPERTY-Statement wird benutzt, um einer Property einen Objektdaten-Operanden als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.
METHOD	**METHOD-Statement:** Das METHOD-Statement wird benutzt, um einer Methode ein Subprogramm als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.
END-CLASS	Das für Natural reservierte Wort END-CLASS muss zum Beenden des DEFINE CLASS-Statements benutzt werden.