

EDBConnector

Inherits From:	Object
Conforms To:	IBConnectors
Declared In:	EDBConnector.h

Class Description

In InterfaceBuilder können Verbindungen zwischen zwei Objekten hergestellt werden. Für jede dieser Verbindungen wird ein Objekt instantiiert, das dem Protokoll IBConnectors gehorcht. Zu dieser Klasse von Objekten gehört EDBConnector. Deren Aufgabe ist es, zur Laufzeit Verbindungen, die in InterfaceBuilder hergestellt wurden, in Aufrufe wie z.B. setDelegate: umzuwandeln.

Ein EDBConnector stellt die Funktionalität der NeXT-eigenen Connectoren zur Verfügung. Dazu gehören der Source der Verbindung, der Name des Outlets, das Ziel und eventuell eine Action. Darüberhinaus enthält ein EDBConnector noch eine Sequenznummer, die bei MultiUse-Outlets

von 1 ausgehend durchlaufend ist und es dem Sourceobjekt ermöglicht, die Reihenfolge der Verbindungen zu erfahren.

Um nun Verbindungen in Methodenaufrufe umzuwandeln, wurde ein Protokoll erstellt, das in Folge genau erklärt wird. Die Reihenfolge der Schritte ist von oben nach unten. Als Beispiel stehe *outlet* als beliebiger Name eines Outlets, auf Groß- und Kleinschreibung ist zu achten.

- 1) Wenn source auf *setOutlet:* antwortet, so wird diese Methode aufgerufen mit destination als Argument. Weiter bei 4).
- 2) Wenn source auf *establishEDBConnection:* antwortet, so wird diese Methode aufgerufen mit self als Argument, d.h. source kann alle Information vom Connector abfragen. Weiter bei 4).
- 3) Mittels *object_setInstanceVariable(source,"outlet",destination)* wird versucht, den Wert der Instanzvariable direkt zu setzen.
- 4) Ist keine Action vorhanden, so weiter bei 7).

- 5) Wenn source auf `setOutletAction:` antwortet, so wird diese Methode aufgerufen mit `sel_getUid(selName)` als Argument.
- 6) Mittels `object_setInstanceVariable(source,"outletAction",sel_getUid(selName))` wird versucht, den Wert der Instanzvariablen direkt zu setzen.
- 7) Ende.

Instance Variables

id
id
NXAtom

source
destination
outletName

const char
u_short

***selName**
sequence

source

Der Source der Verbindung

destination

Das Ziel der Verbindung

outletName

Der Name des Outlets (Achtung, NXUniqueString verwenden beim Setzen!)

selName

Der Name der Action

sequence

Ist ein Outlet ein MultiUse-Outlet, so ist diese Nummer eine eindeutige von 1 aufsteigende Nummer für alle Outlets mit

gleichem Namen und gleicher Source in der Reihenfolge der Entstehung der Verbindungen.

Adopted Protocols

IBConnectors

- source
- destination
- establishConnection
- niblInstantiate
- renewObject:to:
- read:
- write:

Method Types

Initializing and freeing

- init
- initWithSource:andDestination:forOutlet:andSel:
- free

Instance Variables

- outletName
- setOutletName:
- selName
- setSelName:

- sequence
- setSequence:

Instance Methods

free

- free

Freed selName und returniert [super free].

init

- **init**

Returniert initWithSource:andDestination:forOutlet:andSel: mit allen Parameter nil bzw. NULL.

See also: - **initWithSource:andDestination:forOutlet:andSel:**

initWithSource:andDestination:forOutlet:andSel:

- **initWithSource:***source* **andDestination:***destination* **forOutlet:**(const char *)*name* **andSel:**
(const char *)*sel*

Setzt die Instanzvariablen entsprechend und returniert self.

outletName

- (NXAtom)**outletName**

selName

- (const char *)**selName**

Der Name der Action.

sequence

- (u_short)**sequence**

Nur interessant, wenn von einem Outlet aus mehrere Verbindungen gezogen wurden (MultiUse-Outlet). Diese Nummer ist eindeutig und von 1 aufsteigend lückenlos und gibt die Reihenfolge wieder, in der die Verbindungen entstanden sind.

See also: - **setSequence:**

setOutletName:

- **setOutletName:**(NXAtom)*name*

Hier unbedingt wirklich ein NXAtom (d.h. Returnwert von NXUniqueString) übergeben. Returniert self.

setSelName:

- **setSelName:**(const char *)*name*

setSequence:

- **setSequence:**(u_short)*sequence*

Sh. -sequence. Returniert self.

See also: - **sequence**

