EDBConnector

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

Class Description

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

Ein EDBConnector stellt die FunktionalitÙt der NeXT-eigenen Connectoren zur Verfögung. Dazu gehðren der Source der Verbindung, der Name des Outlets, das Ziel und eventuell eine Action. Daröberhinaus enthÙlt ein EDBConnector noch eine Sequencenummer, die bei MultiUse-Outlets von 1 ausgehend durchlaufend ist und es dem Sourceobjekt ermðglicht, die Reihenfolge der Verbindungen zu erfahren.

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

1) Wenn source auf set*Outlet*: 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 set*Outlet*Action: antwortet, so wird diese Methode aufgerufen mit sel_getUid(selName) als Argument.

6) Mittels object_setInstanceVariable(source,"*outlet*Action",sel_getUid(selName)) wird versucht, den Wert der Instanzvariablen direkt zu setzen.
7) Ende.

Instance Variables

id	**source**
id	**destination**
NXAtom	**outletName**
const char	***selName**
u_short	**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 för alle Outlets mit gleichem Namen und gleicher Source in der Reihenfolge der Entstehung der Verbindungen.

Adopted Protocols

IBConnectors	- source - destination - establishConnection - nibInstantiate - 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 löckenlos 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**