LotusScript – so einfach geht es los

EntwicklerCamp Gelsenkirchen, Hands-on 6, Dienstag, 26.03.2012, 16:00 – 17:30 Uhr

Referent:

System:

Lotus Notes 8.5.3 deutsch, Designer und Client

Inhalt

LotusScript – so einfach geht es los 1

… das finde ich nicht 2

… hinten herum 5

Aufbau eines Agenten 5

Dokumente erstellen 6

Dokumente lesen 10

Dokumente ändern 12

… das geht nicht 15

Debugging 15

Errorhandling 17

… vorne herum 19

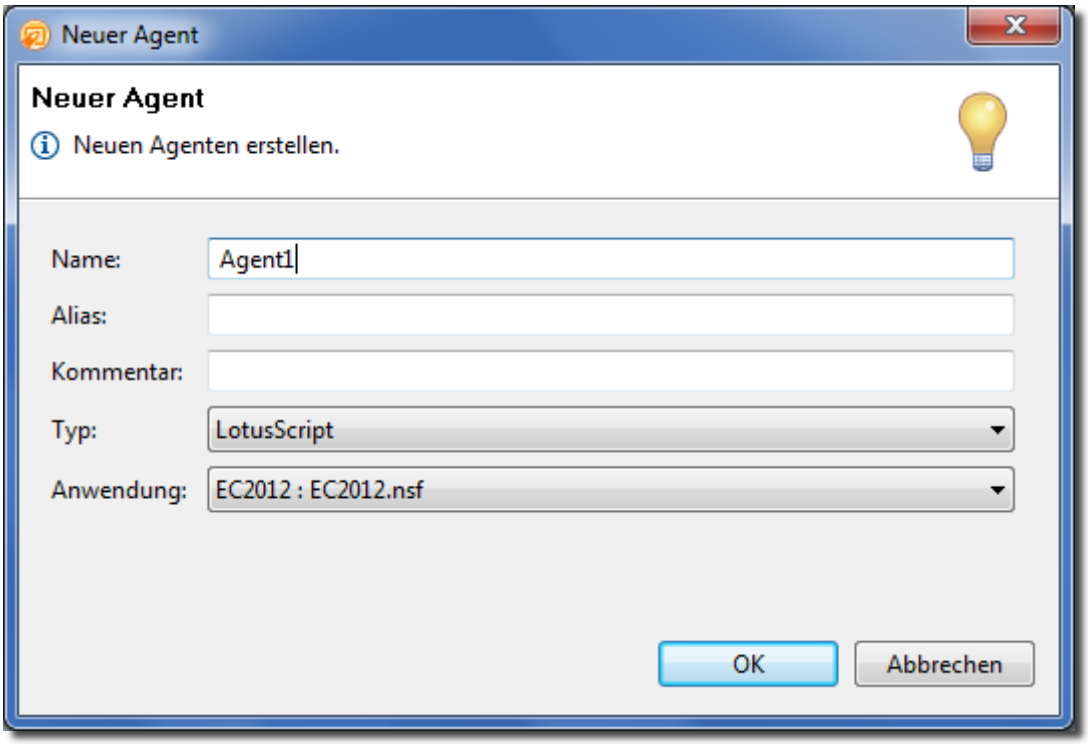
Eine einfache Schaltfläche 19

Validierung per Script 21

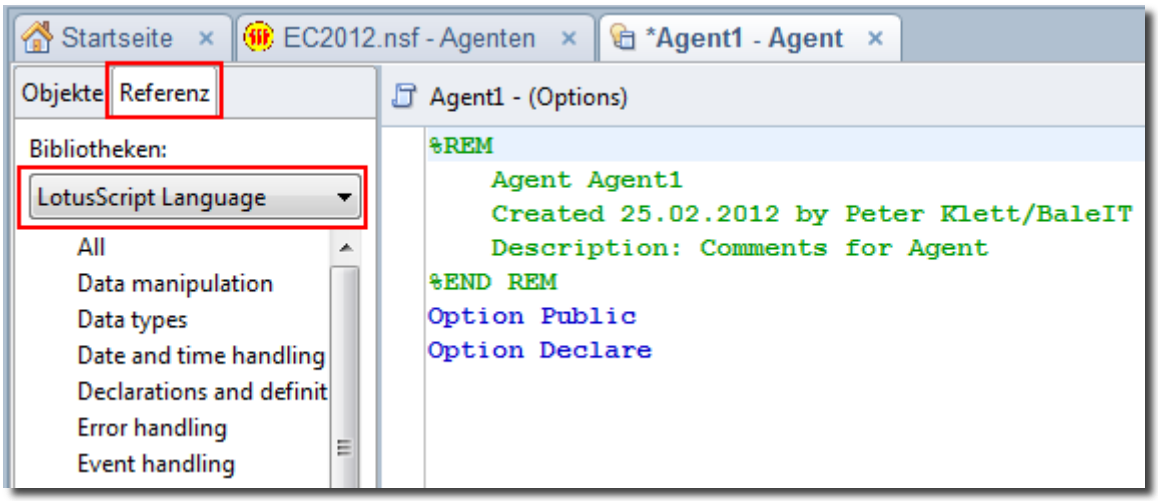
Dokumentenhistorie 23

… das finde ich nicht

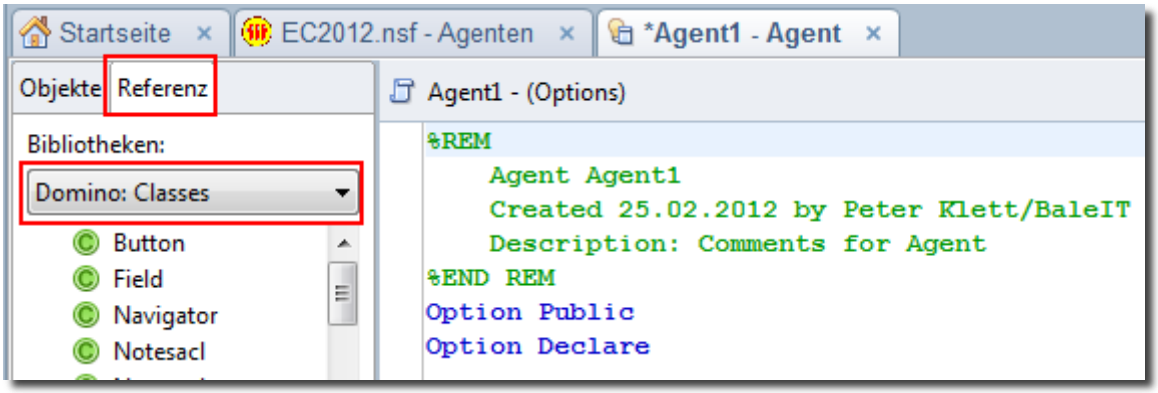
Wir erstellen eine neue Datenbank EC2012 und öffnen diese im Designer. Unter Code – Agenten erstellen wir einen neuen Agenten Agent1, Typ: LotusScript.



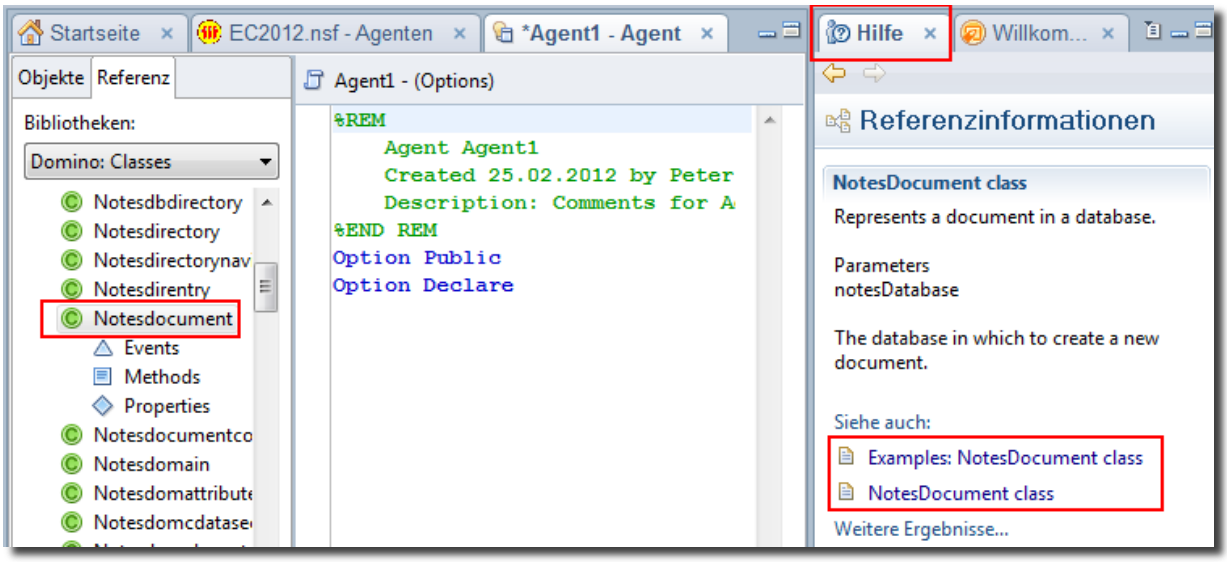
Unter Referenz finden wir alle notwendigen Informationen über die Sprache LotusScript



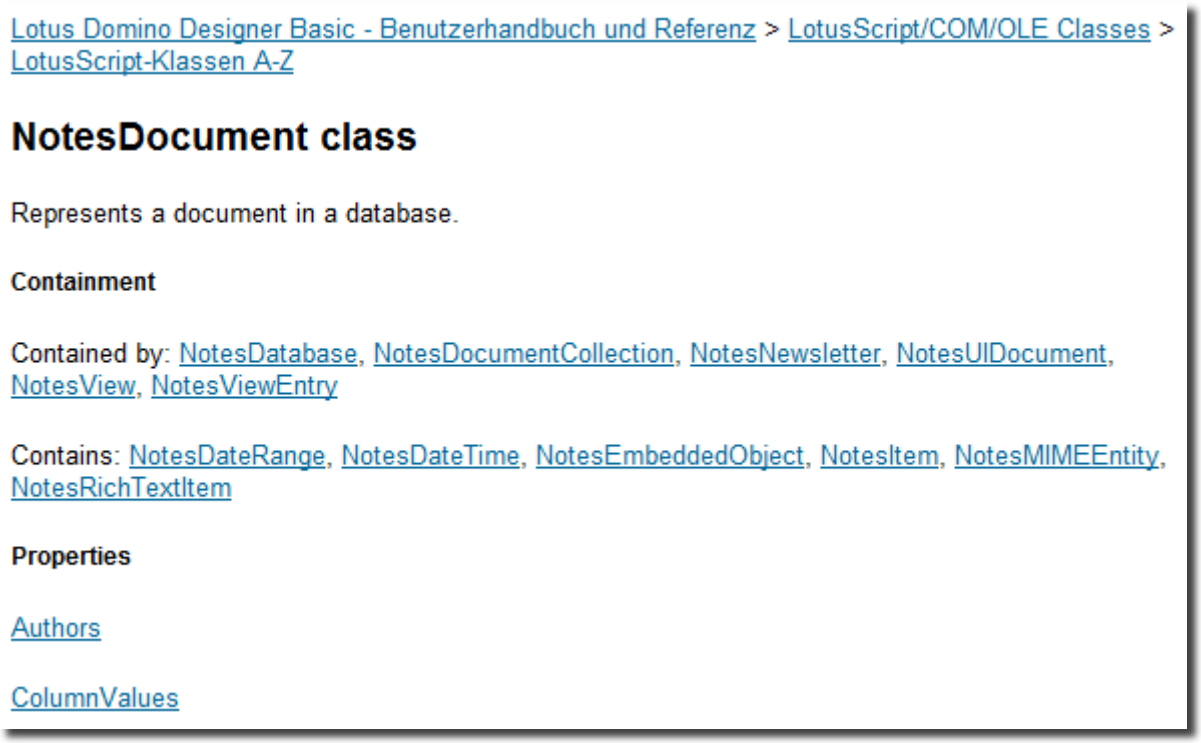
und die Domino-Klassen.



In den Domino-Klassen suchen wir z.B. die Klasse NotesDocument, klicken darauf und drücken F1. Ein Doppelklick auf den Reiter Hilfe öffnet die Hilfe im Vollbild-Modus.



Unter Siehe auch finden wir weitere Informationen zu der gewählten Klasse mit allen Eigenschaften und Methoden.



Lotus Domino Designer Basic - Benutzerhandbuch und Referenz > LotusScript/COM/OLE Classes > LotusScript-Klassen A-Z

NotesDocument class

Represents a document in a database.

Containment

Contained by: NotesDatabase, NotesDocumentCollection, NotesNewsletter, NotesUIDocument, NotesView, NotesViewEntry

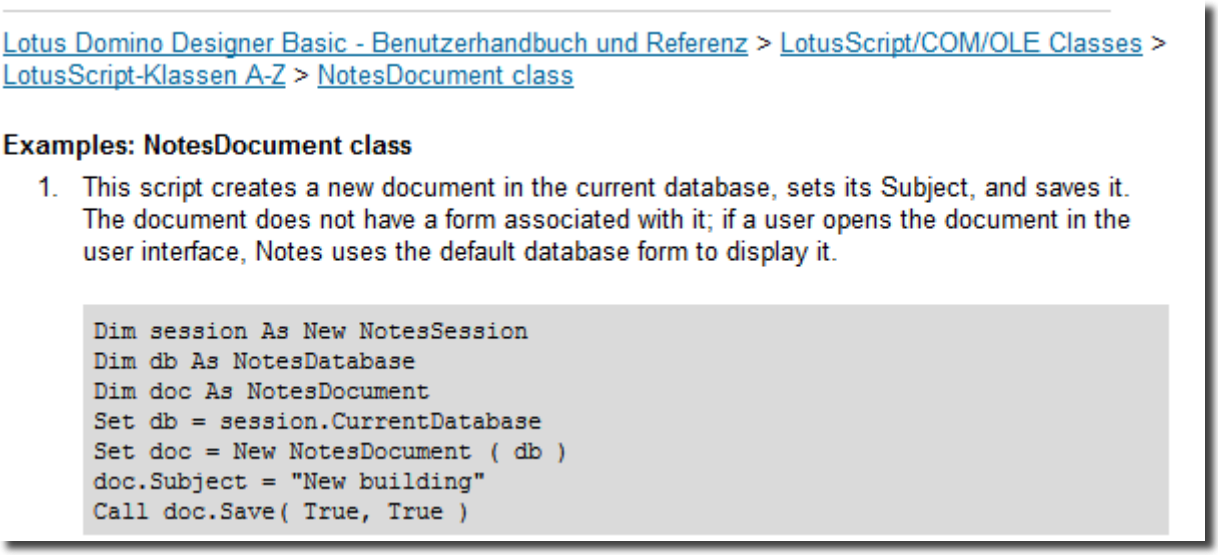
Contains: NotesDateRange, NotesDateTime, NotesEmbeddedObject, NotesItem, NotesMIMEEntity, NotesRichTextItem

Properties

Authors

ColumnValues

Sehr hilfreich sind die Beispiele unter Examples: NotesDocument class.



Lotus Domino Designer Basic - Benutzerhandbuch und Referenz > LotusScript/COM/OLE Classes > LotusScript-Klassen A-Z > NotesDocument class

Examples: NotesDocument class

1. This script creates a new document in the current database, sets its Subject, and saves it. The document does not have a form associated with it; if a user opens the document in the user interface, Notes uses the default database form to display it.

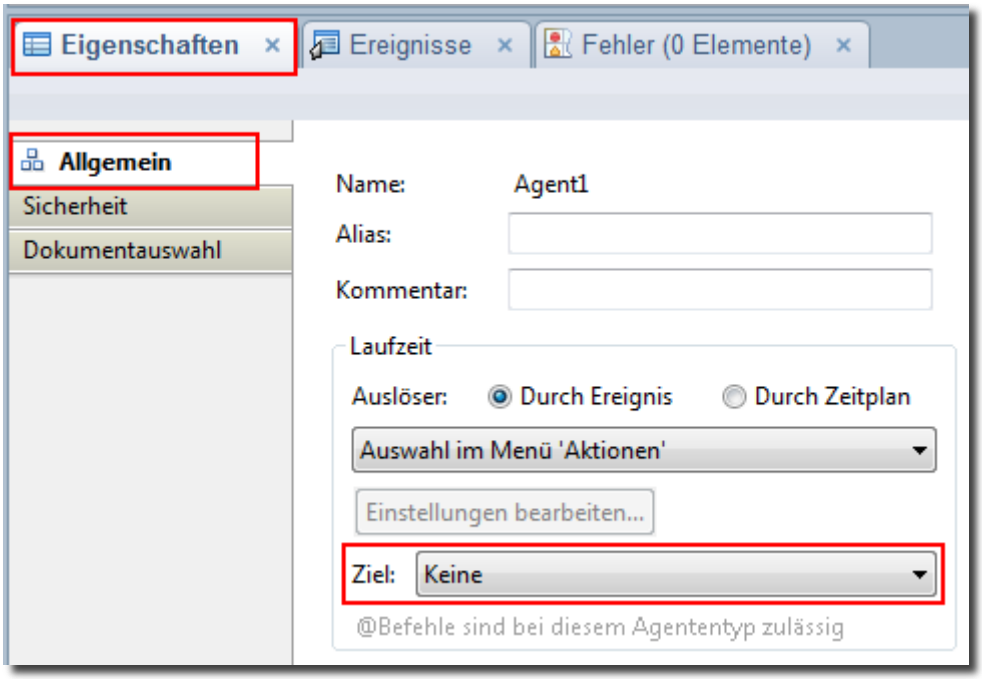
```
Dim session As New NotesSession
Dim db As NotesDatabase
Dim doc As NotesDocument
Set db = session.CurrentDatabase
Set doc = New NotesDocument ( db )
doc.Subject = "New building"
Call doc.Save( True, True )
```

Die Beispielscripte können wir über die Zwischenablage übernehmen und haben das erste lauffähige Script.

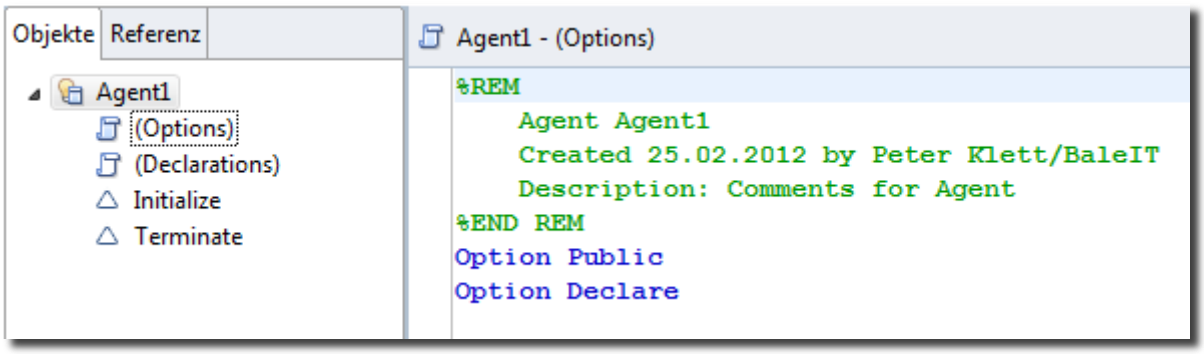
… hinten herum

Aufbau eines Agenten

Zuerst bewegen wir uns im Backend. Wir treten also an die Notesdokumente von hinten heran, im geschlossenen Zustand, nicht über das UI, das User-Interface. In den Eigenschaften des Agenten wählen wir als Ziel „Keine“, damit der Agent gestartet werden kann, ohne dass ein Dokument markiert sein muss.



Unser Agent1 hat verschiedene Objekte



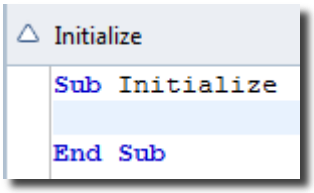
Options

Hier können z.B. ScriptBibliotheken eingebunden werden. Wir sorgen zuerst nur dafür, dass wir immer Option Declare gesetzt haben. Diese Option hilft uns, Tippfehler bei Deklarierungen zu erkennen. So manche Stunde unnötige Fehlersuche kann durch die Option verhindert werden.

Declarations

In diesem Objekt werden globale Variablen deklariert, benötigen wir zu Beginn auch noch nicht.

Initialize



Unser auszuführendes Script schreiben wir in dieses Objekt. Das Initialize ist eine Sub und beginnt mit Sub Initialize und endet mit End Sub, unser Script kommt dazwischen. Initialize bedeutet, dass das Script, das sich in der Sub befindet, beim Initialisieren des Agenten gestartet wird.

Terminate

Terminate ist wie Initialize eine Sub, die beim Beenden des Agenten ausgeführt wird. I.d.R. benötigen wir die nicht.

Dokumente erstellen

Unser Agent1 soll in unserer leeren Datenbank ein paar Dokumente erstellen. Dazu kopieren wir uns zuerst das Beispielscript aus der Hilfe:

```
Initialize
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim doc As NotesDocument
    Set db = session.CurrentDatabase
    Set doc = New NotesDocument ( db )
    doc.Subject = "New building"
    Call doc.Save( True, True )
End Sub
```

Dazu ein paar Erläuterungen:

In LotusScript müssen wir Variablen deklarieren, das erfolgt mit dem Befehl Dim *Variable* As.

Dim db As NotesDatabase legt die Variable db an, in der eine Notes-Datenbank gespeichert werden kann. db ist dabei ein frei von uns zu wählender Name für die Variable.

Wenn die Variable db mit einem Objekt gesetzt werden soll, benötigen wir den Befehl Set *Variable* =.

Set db = session.CurrentDatabase

Dadurch wird db zur aktuellen Datenbank, die wir aus der NotesSession ermitteln, die zuvor mit
Dim session As New NotesSession
deklariert und zugeordnet wurde. Dim session As New NotesSession ist i.d.R. die Grundlage für alle Backend-Scripte.

Es ist nicht immer zwingend notwendig, Deklaration und Setzen zu trennen bzw. in einem Schritt zu schreiben.

Dim session As New NotesSession

ließe sich auch als

```
Dim session As NotesSession
Set session = New NotesSession
```

schreiben. Zu beachten ist, dass Dim in Verbindung mit As, und Set in Verbindung mit = verwendet wird.

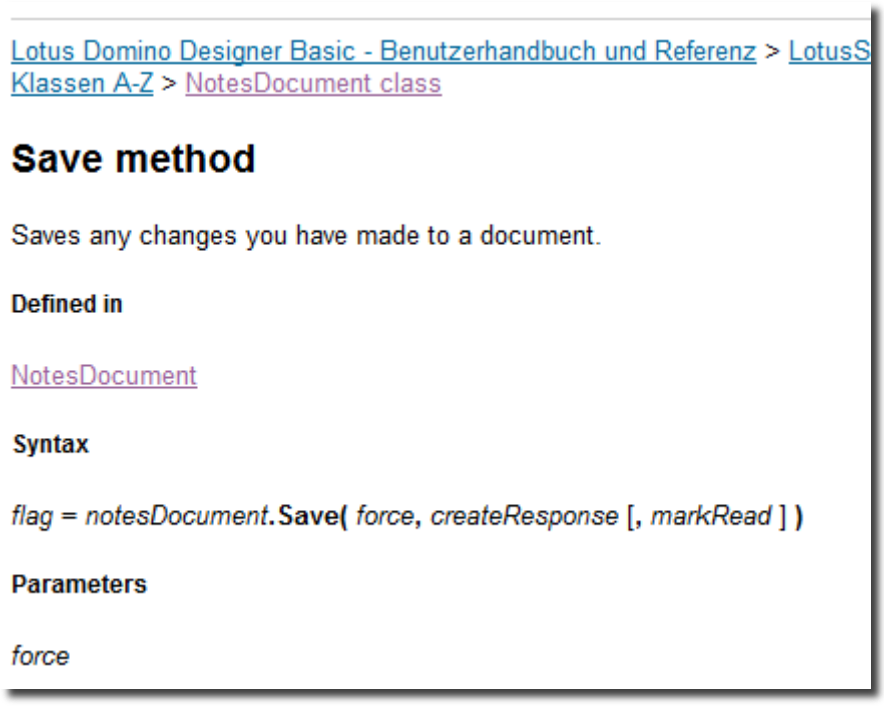
```
Dim doc As NotesDocument
Set doc = New NotesDocument (db)
```

ginge auch in einer Zeile

```
Dim doc As New NotesDocument (db)
```

Unser übernommenes Beispielscript führt nun folgendes aus:

- Zuerst wird eine NotesSession initialisiert
- Dann werden Variablen für eine Notes-Datenbank und ein Notes-Dokument deklariert
- Die Variable db wird auf die aktuelle Datenbank gesetzt
- In der Datenbank wird ein neues Notes-Dokument erstellt und der Variablen doc zugeordnet. WICHTIG: Das Notes-Dokument befindet sich nur im Arbeitsspeicher und ist nicht in der Datenbank gespeichert
- Das Item „Subject“ in dem Dokument wird mit einem String gefüllt, wenn es das Item noch nicht gibt, wird es dadurch erzeugt
- Das doc wird gespeichert. Hierzu wird eine Methode verwendet, die in der Klasse NotesDocument definiert ist. Die Beschreibung der Methode und ihrer Parameter finden wir in der Hilfe, optionale Parameter, die nicht angegeben werden müssen, werden in [eckigen Klammern] dargestellt


Lotus Domino Designer Basic - Benutzerhandbuch und Referenz > LotusS
Klassen A-Z > NotesDocument class

Save method

Saves any changes you have made to a document.

Defined in

NotesDocument

Syntax

flag = notesDocument.**Save(** *force*, *createResponse* [, *markRead*] **)**

Parameters

force

Wir erweitern und ändern das Beispielscript so, dass wir 10 Dokumente erstellen, das Item Form mit „dokument“ belegen, und ein weiteres Item ID aufnehmen, das die UniversalID des Dokuments beinhalten soll. Dabei lernen wir die einfachen Variablen kennen, die keine Domino-Klassen

darstellen. Die Deklaration dieser Variablen erfolgt wie bei den Klassen, nur werden diese Variablen nicht gesetzt, sondern einfach mit dem gewünschten Wert belegt.

Dim i As Integer

deklariert eine Variable i vom Typ Integer, eine ganze Zahl (2 Byte, Wertebereich -32768 bis +32767)

Um der Variablen i den Wert 5 zuzuweisen, verwenden wir nicht Set, sondern

i = 5

Die Variable i nutzen wir in unserem Script als Laufvariable, um 10 Dokumente zu erstellen. Dazu benötigen wir zusätzlich eine For-Schleife.

For i = 1 to 10

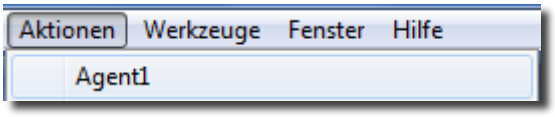
Next i

Nach der Änderung sieht unser Agent so aus:

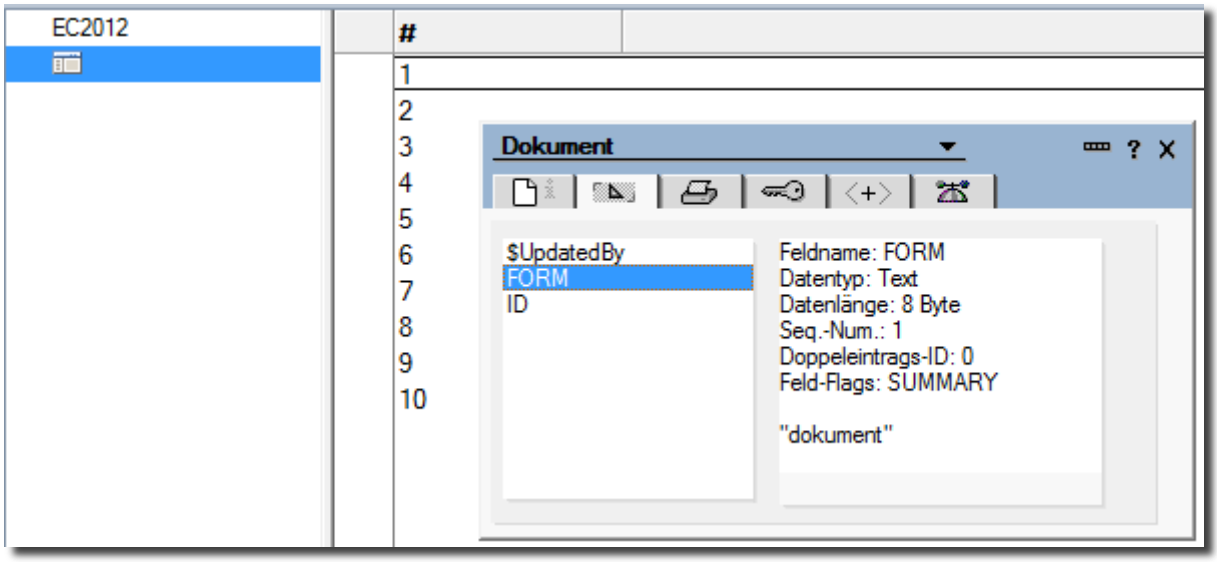
Initialize

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim doc As NotesDocument
    Dim i As Integer
    Set db = session.CurrentDatabase
    For i = 1 To 10
        Set doc = New NotesDocument ( db )
        doc.Form = "dokument"
        doc.ID = doc.UniversalID
        Call doc.Save( True, True )
    Next i
End Sub
```

Wir speichern den Agenten (STRG+S), wechseln in den Notes-Client, starten den Agenten

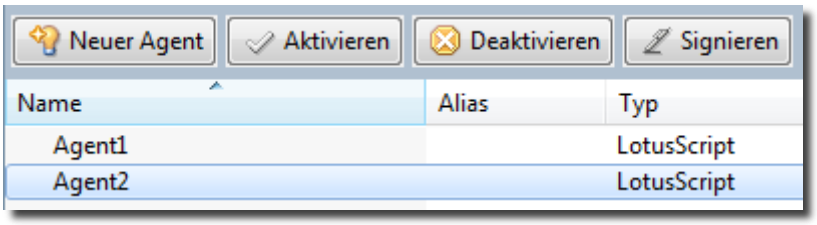


und erhalten als Ergebnis 10 neue Dokumente in unserer Datenbank.



Dokumente lesen

Mit unserem zweiten Agenten Agent2 wollen wir die soeben erstellten Dokumente (alle Dokumente der Datenbank) per Script lesen und uns die IDs aus dem Item ID ausgeben lassen.
Dazu verlassen wir den Agenten (ESC) und kopieren ihn in der Ansicht der Agenten im Designer. Anschließend wird er umbenannt in Agent2.



Wir verwenden dazu die Klasse NotesDocumentCollection. In NotesDatabase ist eine NotesDocumentCollection AllDocuments definiert, die uns eine Collection mit allen Dokumenten der Datenbank zurückgibt. Die Ausgabe der gewünschten Information erfolgt mittels Print in der Statuszeile.

Initialize

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim col As NotesDocumentCollection
    Dim doc As NotesDocument
    Set db = session.CurrentDatabase
    Set col = db.AllDocuments
    Set doc = col.GetFirstDocument
    Do While Not doc Is Nothing
        Print doc.ID (0)
        Set doc = col.GetNextDocument (doc)
    Loop
End Sub
```

Anmerkungen zu dem Script:

Wir gehen wieder über die Session und holen uns daraus die aktuelle Datenbank. Aus der Datenbank füllen wir unsere NotesDocumentColletion mit AllDocuments der aktuellen Datenbank db. Schließlich setzen wir doc auf das erste Dokument der Collection. Mit der Kopfschleife

Do While Not doc Is Nothing

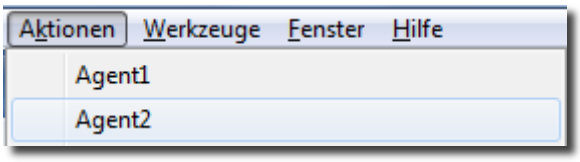
Loop

durchlaufen wir alle Dokumente der Collection. Set doc = col.GetNextDocument (doc) gibt uns jeweils das nächste Dokument in der Collection in Abhängigkeit von aktuellen Dokument doc zurück. Wenn doc das letzte Dokument der Collection ist, ist der Rückgabewert Nothing, auf den wir in der Schleife prüfen.
Die Ausgabe des Inhaltes des Items ID erfolgt mittels Print. Print doc.ID (0).

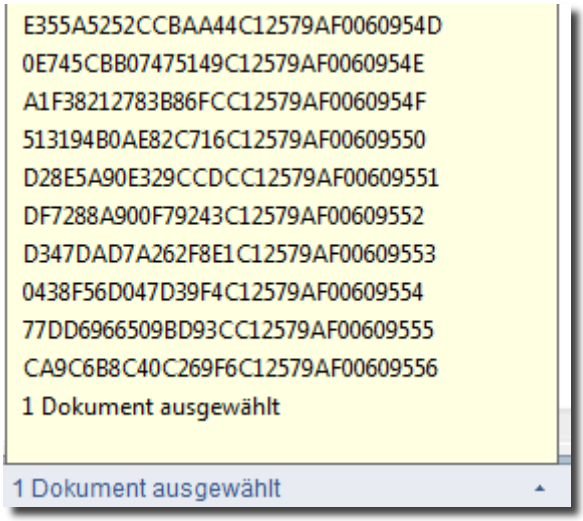
Was bedeutet die (0) in doc.ID (0)?

Alle Items in Notes-Dokumenten sind Arrays (wir ignorieren bewusst RichtextItems), unabhängig davon, ob es sich tatsächlich um Mehrfachwerte handelt, oder nicht. Ein Einfachwert ist dann eben ein Array mit einem Element. Der Befehl Print erwartet einen String, und kein Array. Deshalb erfolgt die Ausgabe des ersten Elementes des Items, das ein String ist. Das erste Element wird mit der (0) angesprochen, das zweite mit (1) usw..

Wir wechseln wieder in den Notes-Client, nachdem wir den Agent2 gespeichert haben, starten ihn



und erhalten als Ergebnis die Ausgabe der IDs in der Statuszeile.



Dokumente ändern

Mit dem dritten und letzten Backend-Agenten wollen wir unsere Dokumente ändern. Dieses Mal ermitteln wir die Dokumente aus einer Ansicht. Im konkreten Anwendungsfall ist das unbedeutend, ob wir eine Ansicht oder eine NotesDocumentCollection verwenden, sondern soll nur eine weitere Methode darstellen.

In unserer Datenbank benennen wir die Standardansicht um in „dokumente“, die erste Spalte wird nicht sortiert und zeigt die ID. In der zweiten Spalte zeigen wir das letzte Änderungsdatum (@Modified).

Der Agent soll jetzt die ID so errechnen, dass im Item ID steht: „ID: *UniversalID*“. Der feste Text „ID: “ und dazu die UniversalID des Dokuments.

Wir kopieren den Agent2 in den Agent3 und nehmen folgende Änderungen vor:

Initialize

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim view As NotesView
    Dim doc As NotesDocument
    Set db = session.CurrentDatabase
    Set view = db.GetView ("dokumente")
    Set doc = view.GetFirstDocument
    Do While Not doc Is Nothing
        doc.ID = "ID: " & doc.UniversalID
        Call doc.Save (True, True)
        Set doc = view.GetNextDocument (doc)
    Loop
End Sub
```

Erläuterungen:

Eine Ansicht (NotesView) funktioniert im Zugriff wie eine NotesDocumentCollection. Wir können mit NotesView.GetFirstDocument das erste Dokument der Ansicht suchen und mit NotesView.GetNextDocument (doc) das nächste Dokument in der Ansicht in Abhängigkeit von dem aktuellen Dokument doc. Die Dokumente in einer Ansicht sind nach den Einstellungen der Ansicht sortiert, wogegen die Dokumente in einer NotesDocumentCollection grundsätzlich nach Erstelldatum sortiert sind.

Die neue ID errechnen wir, indem der String „ID: “ mit der UniversalID des Dokuments kombiniert wird. Um zwei Strings (also Texte) aneinander zu hängen, verwenden wir das Zeichen „&“, niemals „+“, obwohl das grundsätzlich auch funktioniert. Es kann aber bei unsauberem Typenhandling zu Fehlern kommen, Beispiel:

„4“ & „3“ = „43“
„4“ + „3“ = „43“
4 & 3 = „43“
aber
4 + 3 = 7

Zu beachten ist, dass bei doc.UniversalID keine (0) verwendet wird, da doc.UniversalID kein Array (kein Item), sondern eine Eigenschaft des Dokuments ist, und diese Eigenschaft hat den Typ „String“.

Wir speichern den Agenten, wechseln in den Notes-Client, und starten ihn.

vorher:

IDs	Modified
E355A5252CCBAA44C12579AF0060954D	25.02.2012 18:34:56
0E745CBB07475149C12579AF0060954E	25.02.2012 18:34:56
A1F38212783B86FCC12579AF0060954F	25.02.2012 18:34:56
513194B0AE82C716C12579AF00609550	25.02.2012 18:34:56
D28E5A90E329CCDCC12579AF00609551	25.02.2012 18:34:56
DF7288A900F79243C12579AF00609552	25.02.2012 18:34:56
D347DAD7A262F8E1C12579AF00609553	25.02.2012 18:34:56
0438F56D047D39F4C12579AF00609554	25.02.2012 18:34:56
77DD6966509BD93CC12579AF00609555	25.02.2012 18:34:56
CA9C6B8C40C269F6C12579AF00609556	25.02.2012 18:34:56

nachher:

IDs	Modified
ID: E355A5252CCBAA44C12579AF0060954D	25.02.2012 18:38:20
ID: 0E745CBB07475149C12579AF0060954E	25.02.2012 18:38:20
ID: A1F38212783B86FCC12579AF0060954F	25.02.2012 18:38:20
ID: 513194B0AE82C716C12579AF00609550	25.02.2012 18:38:20
ID: D28E5A90E329CCDCC12579AF00609551	25.02.2012 18:38:20
ID: DF7288A900F79243C12579AF00609552	25.02.2012 18:38:20
ID: D347DAD7A262F8E1C12579AF00609553	25.02.2012 18:38:20
ID: 0438F56D047D39F4C12579AF00609554	25.02.2012 18:38:20
ID: 77DD6966509BD93CC12579AF00609555	25.02.2012 18:38:20
ID: CA9C6B8C40C269F6C12579AF00609556	25.02.2012 18:38:20

Die IDs sind verändert worden, das Änderungsdatum der Dokumente auch.
Wir starten den Agenten ein zweites Mal:

IDs	Modified
ID: E355A5252CCBAA44C12579AF0060954D	25.02.2012 18:39:51
ID: 0E745CBB07475149C12579AF0060954E	25.02.2012 18:39:51
ID: A1F38212783B86FCC12579AF0060954F	25.02.2012 18:39:51
ID: 513194B0AE82C716C12579AF00609550	25.02.2012 18:39:51
ID: D28E5A90E329CCDCC12579AF00609551	25.02.2012 18:39:51
ID: DF7288A900F79243C12579AF00609552	25.02.2012 18:39:51
ID: D347DAD7A262F8E1C12579AF00609553	25.02.2012 18:39:51
ID: 0438F56D047D39F4C12579AF00609554	25.02.2012 18:39:51
ID: 77DD6966509BD93CC12579AF00609555	25.02.2012 18:39:51
ID: CA9C6B8C40C269F6C12579AF00609556	25.02.2012 18:39:51

Die IDs sind unverändert, aber das Änderungsdatum hat sich aktualisiert.

Solch ein Agent stellt im Notesumfeld mit die größte Sünde dar, die man begehen kann. Im Unterschied zur Formelsprache ist LotusScript mächtiger, verlangt vom Entwickler aber auch mehr

mitdenken und steuern. Der Formelagent

FIELD ID := „ID:“ + @Text (@DocumentUniqueID)

würde genau das gleiche Ergebnis produzieren, die Dokumente aber nur speichern, wenn tatsächlich eine Änderung eingetreten ist. Unser Scriptagent würde ohne Notwendigkeit die Dokumente für alle Benutzer auf ungelesen setzen, die Gefahr von Speicher- und Replizierkonflikten erhöhen und in einem replizierten Umfeld ein immens großes Replizieraufkommen verursachen.
Deshalb benötigen wir eine Steuerung, das Dokument darf nur bei Änderungen gespeichert werden.
Wir modifizieren den Agenten wie folgt (Agent3mod1)

Initialize

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim view As NotesView
    Dim doc As NotesDocument
    Dim neueid As String
    Set db = session.CurrentDatabase
    Set view = db.GetView ("dokumente")
    Set doc = view.GetFirstDocument
    Do While Not doc Is Nothing
        neueid = "ID: " & doc.UniversalID
        If doc.ID (0) <> neueid Then
            doc.ID = neueid
            Call doc.Save (True, True)
        End If
        Set doc = view.GetNextDocument (doc)
    Loop
End Sub
```

Erläuterungen:

Zuerst deklarieren wir uns eine neue Variable neueid vom Datentyp String, in die wir die neue ID des Dokumentes berechnen. Anschließend vergleichen wir die neue ID mit der in dem Dokument gespeicherten. Nur wenn die neue ID von der bereits gespeicherten abweicht, setzen wir die neue ID und speichern das Dokument doc.

Diesen Agenten können wir beliebig oft starten, er wird die Dokumente, die nicht zu ändern sind, in Ruhe lassen.

Variationsmöglichkeiten:

Mit NotesDatabase.Search könnten wir uns eine NotesDocumentCollection erstellen, die nur die zu ändernden Dokumente enthält

Set col = db.Search ({ID != “ID: “ + @Text (@DocumentUniqueID)}, Nothing, 0)

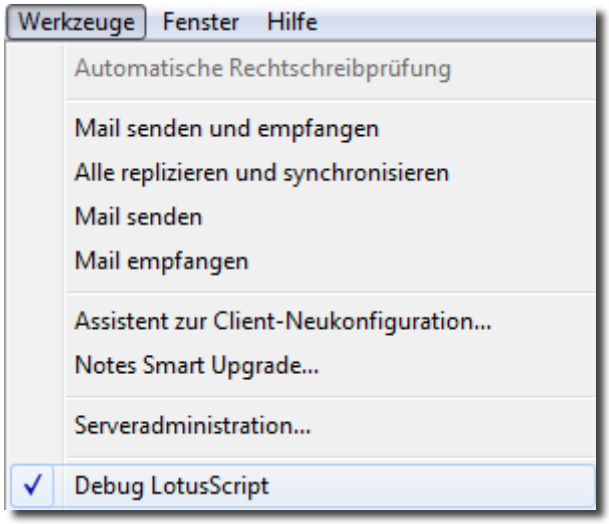
und alle Dokumente der Collection ändern.

Eine weitere Möglichkeit stellt NotesDatabase.UnprocessedDocuments dar. Diese Collection enthält alle markierten Dokumente. Dazu muss im Agenten das Ziel von „Keine“ auf „Alle ausgewählten Dokumente“ geändert werden.

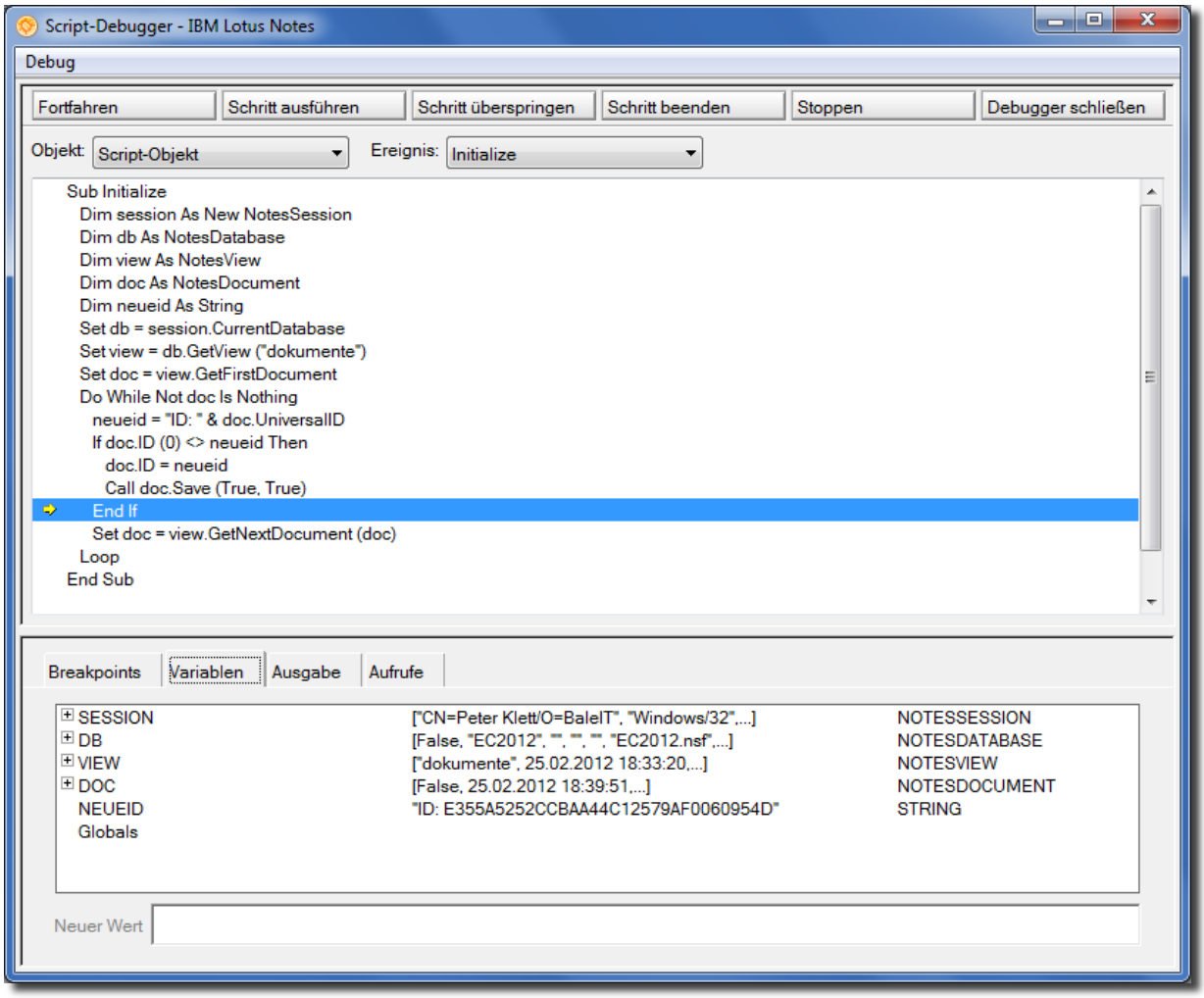
… das geht nicht

Debugging

Was tun, wenn unser Script nicht das gewünschte Ergebnis liefert? Ein sehr wertvolles Werkzeug ist der Debugger. Gestartet wird er über Werkzeuge – Debug LotusScript



Nach Aktivieren des Debuggers starten wir unseren Agenten.



Der Debugger erscheint. Mit „Schritt ausführen“ können wir das Script Schritt für Schritt ausführen lassen. Im unteren Fenster werden im Reiter „Variablen“ die Variablen mit ihren aktuellen Werten angezeigt. Der aktuell ausgeführte Befehl wird im Script markiert.

Errorhandling

Wichtig bei der Programmierung in LotusScript ist ein Errorhandling, was macht das Script beim Auftreten eines Fehlers? Steuern wir das nicht selbst, bricht Notes im Fehlerfall das Script ab und bringt eine Fehlermeldung, die uns nicht immer weiter hilft, z.B. ein „type mismatch“ oder ein „object variable not set“. Die Fehlermeldung ist grundsätzlich richtig, aber wobei ist der Fehler aufgetreten?

Spezielles Errorhandling

Grundsätzlich sollten wir überall dort, wo ein Fehler auftreten kann, den Fehlerfall abfangen. Wir nehmen als Grundlage unseren Agent3mod1 und kopieren ihn zum Agent3mod2. In diesem Agent ist eine potentielle Fehlerquelle enthalten. Wir greifen auf die Ansicht „dokumente“ zu, die in der Datenbank fehlen könnte. Deshalb prüfen wir im speziellen Errorhandling nach dem Setzen der Ansicht ab, ob die Ansicht geöffnet werden konnte. Falls nicht, soll eine Fehlermeldung ausgegeben und das Script beendet werden.

Initialize

```
Sub Initialize
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim view As NotesView
    Dim doc As NotesDocument
    Dim neueid As String
    Set db = session.CurrentDatabase
    Set view = db.GetView ("dokumente")
    If view Is Nothing Then
        MsgBox "Ansicht <dokumente> konnte nicht geöffnet werden", 16, "Fehler"
        Exit Sub
    End If
    Set doc = view.GetFirstDocument
    Do While Not doc Is Nothing
        neueid = "ID: " & doc.UniversalID
        If doc.ID (0) <> neueid Then
            doc.ID = neueid
            Call doc.Save (True, True)
        End If
        Set doc = view.GetNextDocument (doc)
    Loop
End Sub
```

Allgemeines Errorhandling

Für nicht erwartete Fehler bietet sich ein allgemeines Errorhandling an. Beim allgemeinen Errorhandling wird zu Beginn des Scriptes definiert, was im Falle eines Fehlers passieren soll. Dabei wird eine Sprungmarkierung angegeben, in die das Script springen soll. Die Sprungmarkierung wird am Ende des Scriptes definiert. Wir müssen beachten, dass wir unser Script vor der Sprungmarkierung beenden, damit es nicht beim fehlerfreien Lauf in die Fehlerroutine gerät.
Als Basis nehmen wir wieder den Agent3mod1 und kopieren ihn zum Agent3mod3.

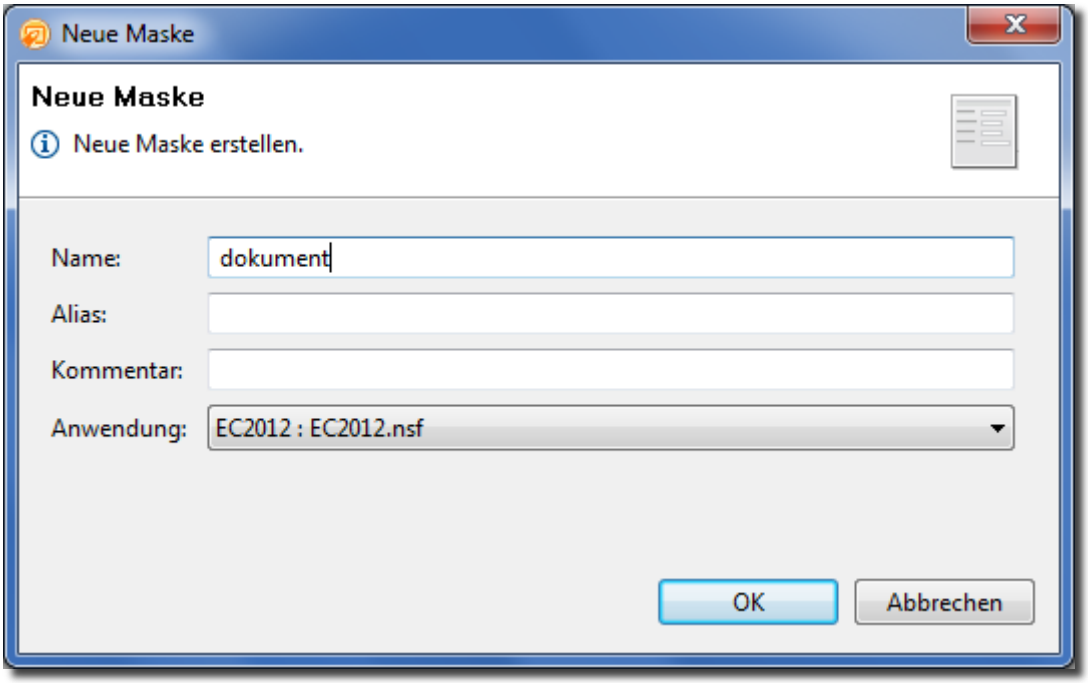
△ Initialize

```
Sub Initialize
    On Error GoTo Fehler
    Dim session As New NotesSession
    Dim db As NotesDatabase
    Dim view As NotesView
    Dim doc As NotesDocument
    Dim neueid As String
    Set db = session.CurrentDatabase
    Set view = db.GetView ("dokumente")
    Set doc = view.GetFirstDocument
    Do While Not doc Is Nothing
        neueid = "ID: " & doc.UniversalID
        If doc.ID (0) <> neueid Then
            doc.ID = neueid
            Call doc.Save (True, True)
        End If
        Set doc = view.GetNextDocument (doc)
    Loop
    Exit Sub
Fehler:
    MsgBox Error & "in Zeile " & Erl, 16, "Fehler"
    Exit Sub
End Sub
```

Error enthält die Fehlermeldung, Erl, die Zeile des Scriptes, in dem der Fehler aufgetreten ist. Als weitere Fehlervariable gibt es noch Err, die die Fehlernummer enthält.

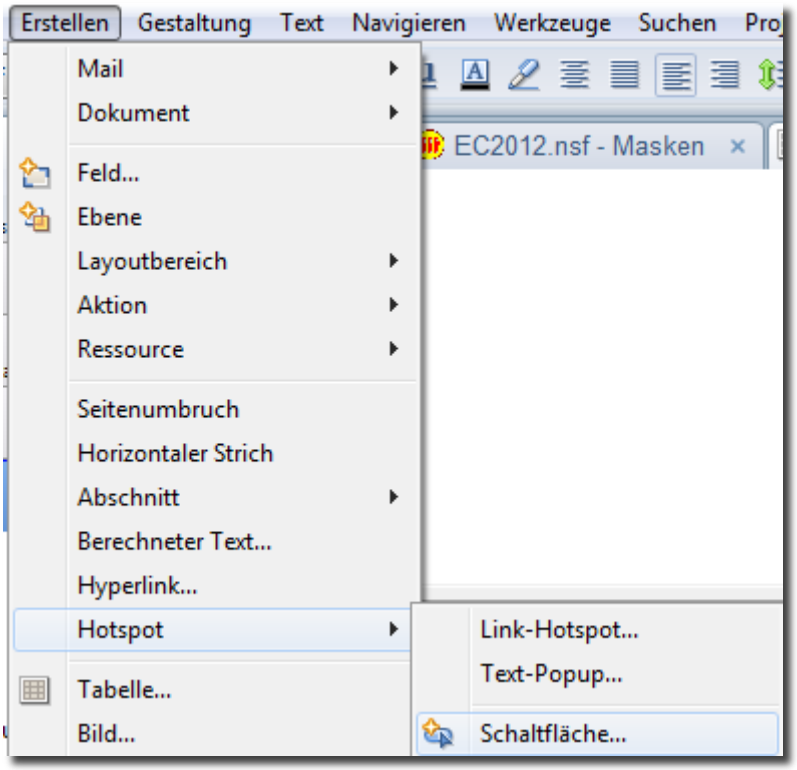
… vorne herum

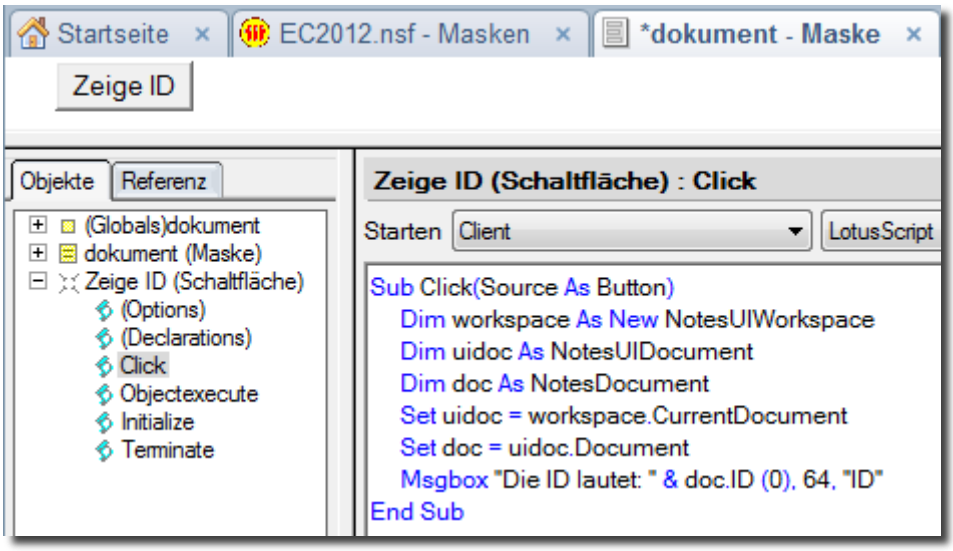
Im nächsten Schritt gehen wir vorne herum auf die Dokumente zu, über das UI, das User-Interface. Dazu erstellen wir uns in unserer Datenbank die Maske „dokument“.



Eine einfache Schaltfläche

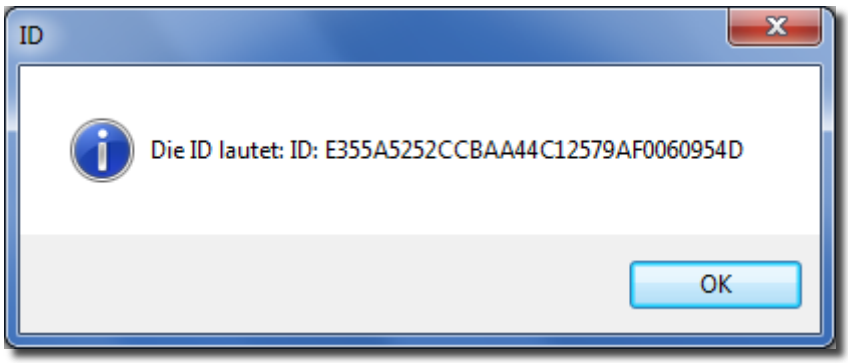
Die Maske bekommt noch kein Feld, sondern nur eine Schaltfläche mit der Bezeichnung „Zeige ID“.





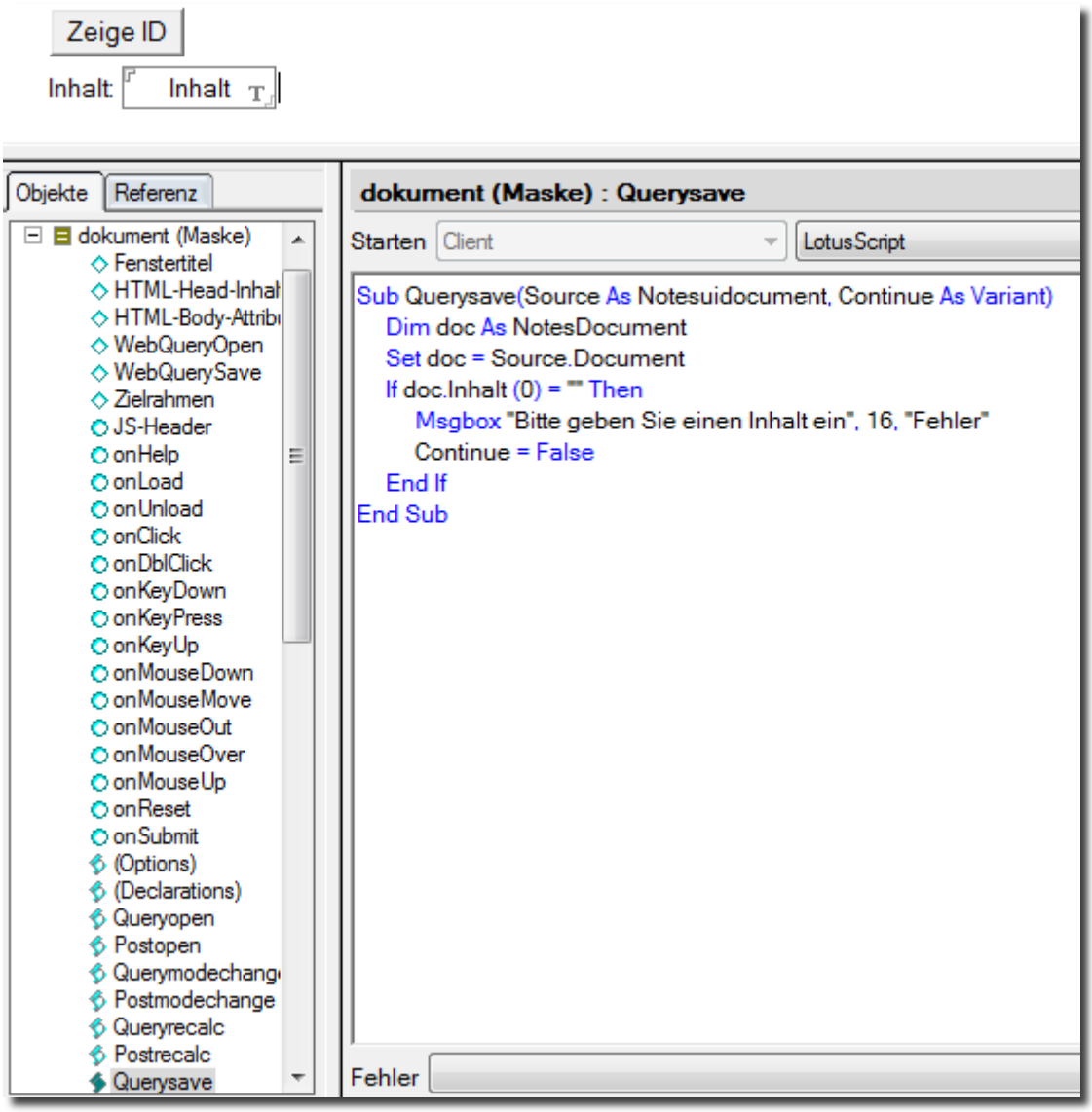
Erläuterungen:

Bei einer Schaltfläche schreiben wir das auszuführende Script in die Sub Click, nachdem wir die Sprache von Formel auf LotusScript geändert haben. Beim Klicken des Buttons soll das Script ausgeführt werden.
Der Einstieg über das Frontend erfolgt i.d.R. über die Klasse NotesUIWorkspace. In NotesUIWorkspace finden wir unter CurrentDocument das aktuell geöffnete Dokument von der Klasse NotesUIDocument. Darin enthalten ist das Objekt Document von der Klasse NotesDocument, das uns schon von den Backendagenten bekannt ist. Mit Ausnahme von Richtextfeldern können wir Änderungen am NotesDocument vornehmen, die auch sofort im NotesUIDocument sichtbar sind.
Nach Betätigen unserer Schaltfläche erhalten wir diese Messagebox als Ergebnis



Validierung per Script

Unsere Maske bekommt ein Feld mit dem Namen „Inhalt“, Typ Text, bearbeitbar. Per Script werden wir das Speichern des Dokuments verhindern, wenn das Feld leer ist.



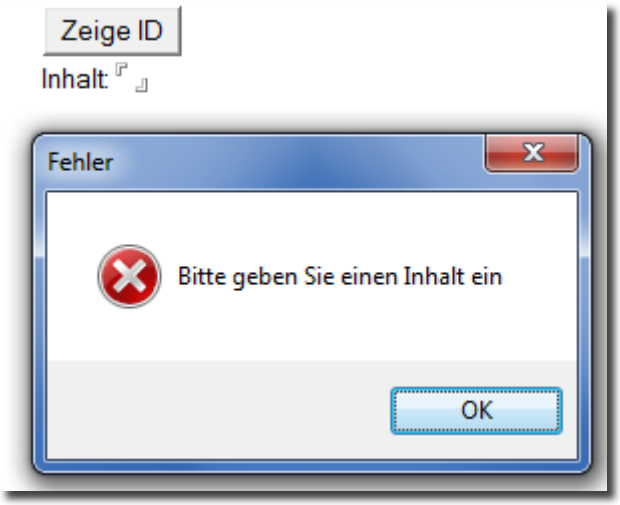
Erläuterungen:

In der Maske gibt es einige Events, in die wir mittels LotusScript eingreifen können. Zu nennen wären beispielsweise:

Postopen	nach dem Öffnen des Dokuments
Postrecalc	nach dem Aktualisieren (F9) des Dokuments
Querysave	vor dem Speichern des Dokuments
Queryclose	vor dem Schließen des Dokuments

Unser Validierungsscript erstellen wir im Querysave, es wird direkt vor dem Speichern des Dokuments ausgeführt. Der Einstieg erfolgt dieses Mal nicht über NotesUIWorkspace, da der Sub Querysave mit dem ersten Parameter Source bereits das NotesUIDocument übergeben wird. Über das NotesUIDocument greifen wir auf das NotesDocument zu und überprüfen den Wert des Feldes „Inhalt“. Ist der Wert leer, setzen wir den zweiten Parameter Continue auf False, wodurch das

Speichern des Dokuments abgebrochen wird.
Wir öffnen eines unserer Dokumente in der Datenbank und setzen es in den Bearbeitenmodus (STRG+B). Das Feld Inhalt ist leer. Beim Speichern des Dokuments (STRG+S) erhalten wir die Validierungsmeldung, das Dokument wird nicht gespeichert.



Dokumentenhistorie

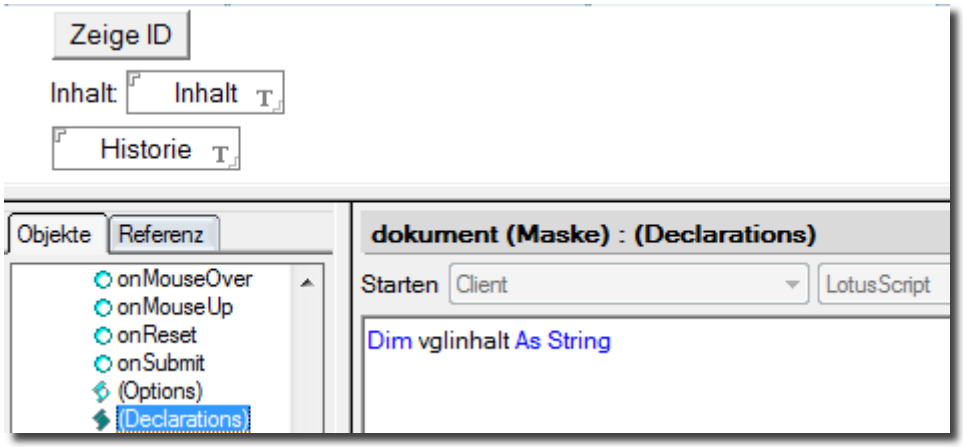
Abschließend bauen wir uns eine kleine Dokumentenhistorie. Alle Änderungen des Feldes Inhalt sollen in der Historie gespeichert werden. Zuerst benötigen wir ein neues Feld „Historie“, dieses Feld ist ein Textfeld, Berechnet beim Anlegen, Mehrfachwerte erlaubt, Trennzeichen der Mehrfachwerte ist Neue Zeile, Formel “”.

Sobald beim Speichern eine Änderung des Wertes von Inhalt festgestellt wird, erfolgt ein neuer Eintrag in der Historie.

Ablauf:

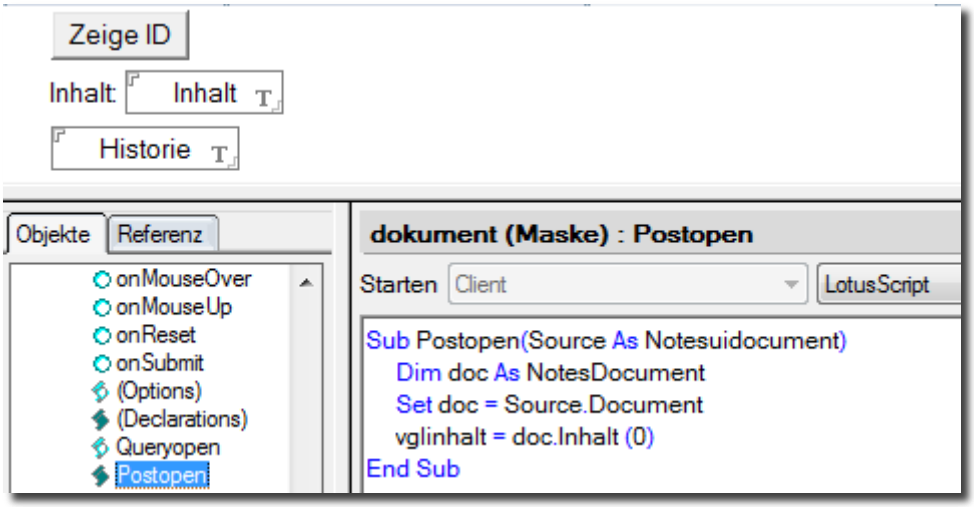
In den Declarations benötigen wir eine Variable, die den Vergleichswert des Inhaltes speichert. Die muss dort definiert sein, damit sie in allen Subs zur Verfügung steht.
Im Postopen wird die Variable mit dem aktuellen Feldinhalt gefüllt.
Im Querysave erfolgt ein Vergleich zwischen dem gemerkten Inhalt und dem aktuellen des Dokuments. Bei einer Abweichung wird die Historie um den entsprechenden Eintrag ergänzt und die Vergleichsvariable mit dem neuen Wert gefüllt, damit beim nächsten Speichern der zuletzt protokollierte Inhalt als Vergleichswert bereit steht.

Declarations



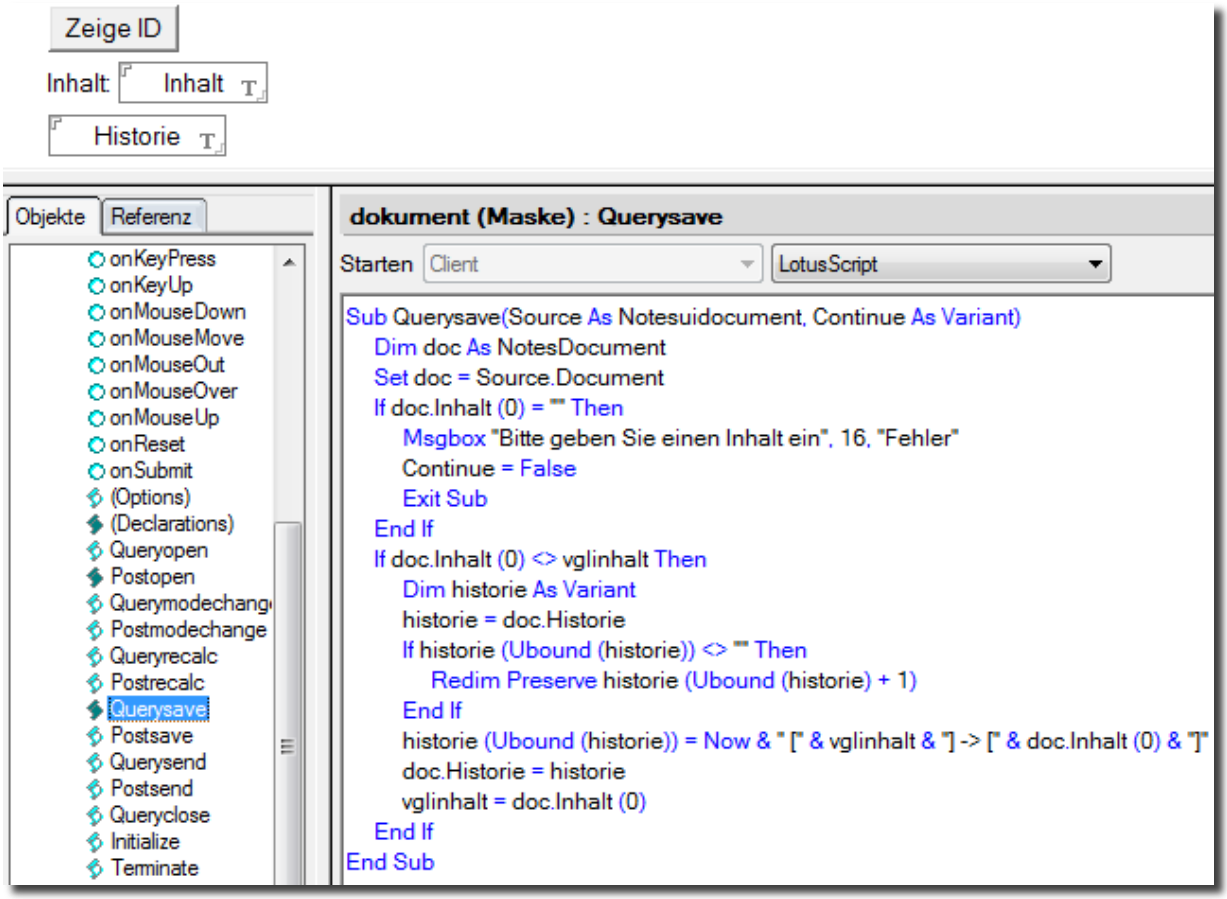
Es wird die globale Variable vglinhalt als String deklariert.

Postopen



Die globale Variable vglinhalt wird mit dem Wert des Feldes Inhalt des soeben geöffneten Dokuments belegt.

Querysave



Zuerst muss in der Validierung ein Exit Sub ergänzt werden, damit das Script bei fehlerhafter Validierung gestoppt wird.

Nur wenn der Wert des Feldes Inhalt nicht mit dem Wert in der globalen Variablen vglinhalt übereinstimmt, erfolgt ein neuer Eintrag in der Historie. Das Feld Historie wird ein Array mit mehreren Elementen, dessen Bearbeitung per Script nicht direkt im Feld möglich ist. NotesDocument.Historie (0) ist der Wert des ersten Elements, NotesDocument.Historie (1) der des zweiten usw.. Das Ändern der einzelnen Elemente muss in einer separaten Variablen erfolgen, das Zurückschreiben der Variablen in das Feld erfolgt dann mittels einer einzigen Zuweisung.

```
NotesDocument.Historie (0) = „erster Wert“
NotesDocument.Historie (1) = „zweiter Wert“
```

Diese Schreibweise ist nicht gestattet und führt zu einem Fehler. Es muss ein Umweg über eine Variable erfolgen:

```
variable = NotesDocument.Historie
variable verändern
NotesDocument.Historie = variable
```

Im Detail:

Dim historie As Variant

Es wird eine Variable historie als Variant deklariert. Die Deklarierung kann hier erfolgen, es wäre aber auch möglich, diese schon im oberen Teil bei den anderen Deklarierungen einzutragen. Der Datentyp Variant erlaubt die Aufnahme beliebiger Datentypen, wir benötigen ihn als Array.

historie = doc.Historie

Die Variable historie bekommt als Wert den Inhalt des Feldes (Items) Historie, wenn das Feld leer ist, handelt es sich um ein Array mit einem Element ohne Inhalt.

If historie (Ubound (historie)) <> "" Then

Ubound ist die obere Grenze des Arrays, angenommen historie hätte zwei Elemente, dann gäbe Ubound (historie) die Zahl 1 zurück (da das erste Element mit der 0 angesprochen wird). Wenn dann historie (1) nicht leer ist, ist die Bedingung erfüllt, um den nächsten Befehl auszuführen.

> Redim Preserve historie (Ubound (historie) + 1)
>
> Redim deklariert die Variable erneut, Preserve sorgt dafür, dass der bisherige Inhalt erhalten bleibt. Als Ergebnis der Aktion hat unser Array historie ein Element mehr als zuvor.

End If

historie (Ubound (historie)) = Now & " [" & vglinhalt & "] -> [" & doc.Inhalt (0) & "]"

In das höchste Element des Arrays historie schreiben wir den Protokolleintrag. Das höchste Element ist definitiv leer, denn wenn das bisher höchste Element nicht leer war, wurde mit Redim Preserve ein neues leeres Element angehängt .

doc.Historie = historie

Das veränderte Array historie schreiben wir jetzt zurück in das Feld Historie des Dokuments.

vglinhalt = doc.Inhalt (0)

Zuletzt bekommt die Vergleichsvariable vglinhalt den aktuellen Wert des Feldes Inhalt.

Wir öffnen eines der Dokumente unserer Datenbank im Bearbeitenmodus, ändern den Inhalt und speichern das Dokument mehrfach. Mit jeder Änderung von Inhalt bekommen wir beim Speichern einen neuen Eintrag in das Historienfeld.