



EntwicklerCamp 2014

Track 1, Session 2:

Fehlerbehandlung in Notes

Gelsenkirchen, 17. März 2014

Innovative Software-Lösungen

www.assono.de



Thomas Bahn

- Diplom-Mathematiker, Universität Hannover
- seit 1997 entwickle ich mit Java und relationalen Datenbanken
- seit 1999 mit Notes/Domino zu tun: Entwicklung, Administration, Beratung und Schulungen
- regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen zu IBM Lotus Notes/Domino und Autor für THE VIEW



 **tbahn@assono.de**
 **www.assono.de/blog**
 **04307/900-401**

 **assono**
IT-Consulting & Solutions



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



Fehler vermeiden: defensive Programmierung

- Fehler vorhersehen, abfragen und – wo möglich – sinnvoll reagieren
- Beispiel:

```
Set doc = view.GetDocumentByKey(key)
MessageBox doc.Title(0)
```

- Was passiert, wenn das erwartete Dokument fehlt?
- Besser defensiv:

```
Set doc = view.GetDocumentByKey(key)
If Not doc is Nothing Then
    MessageBox doc.Title(0)
Else
    ' Fehler melden, Vorgabe-Dokument anlegen, ...
End If
```



Warum Fehler trotzdem immer passieren können

- erwartete Ressourcen sind nicht vorhanden oder erreichbar, z. B.
 - Dokument gelöscht
 - Datenbank verschoben
 - Festplatte voll
 - (anderer) Server heruntergefahren
- Fehler auf anderer Ebene, z. B.
 - Hardware kaputt
 - Betriebssystemfehler
 - Netzwerkproblem
 - Fehler im Notes-Client oder Domino-Server
- Fehler „sitzt vor dem Bildschirm“ 😊



Fehlerbehandlung ist mehr als Fehlerprotokollierung...

- Erster Schritt: Merken, dass ein Fehler aufgetreten ist
- Und dann?
- erst einmal protokollieren
- Und dann?
- Manchmal kann man sinnvoll reagieren:
 - Benutzer fragen (z. B. andere Datei auswählen)
 - kurz warten und Aktion wiederholen
(kann z. B. bei kurzen Netzwerkaussetzern helfen)
 - automatisch auf Cluster-Partner wechseln
 - ignorieren (Dokument löschen: ist nicht mehr da \Rightarrow egal)



Fehlerkontext ermitteln

- Was protokollieren?
- Alles, was helfen könnte, den Fehler zu identifizieren oder zumindest nachzustellen
 - Fehlernummer, Fehlertext
 - Fehlerposition (Zeile, Prozedur, Agent, Stack-Trace)
 - aktuelle Anwendung (Datenbank, Server, ...)
 - aktueller Kontext (Dokument, Ansicht)
 - aktueller Benutzer (Name, Rechte, Rollen, ...)
 - Versionen (Anwendung, Notes-Client, Domino-Server, ...)
 - Systemzustand (freies RAM, freier Festplattenplatz, ...)
- im Zweifelsfall: so viel wie möglich, ohne dass es die Benutzer stört oder die Datenbank platzen lässt



Fehler einsammeln mit/ohne Benutzerunterstützung

- Soll der Benutzer in die Protokollierung einbezogen werden?
- Pro
 - „Was hast du gerade gemacht, als der Fehler auftrat?“
 - Gefühl der Kontrolle
- Contra
 - klicken Dialoge einfach weg
 - geheime Informationen könnten so verbreitet werden
- Unsere „Best Practice“
 - immer im Hintergrund protokollieren
 - zusätzlich Benutzer fragen, ob eine E-Mail verschickt werden soll



Agenda

- Motivation
- **Formelsprache**
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



Formelsprache

- nur rudimentäre Möglichkeiten, auf Fehler zu reagieren
- keine zentrale Fehlerbehandlung möglich
- bei einigen Fehlern wird die Ausführung der Formel einfach abgebrochen oder das Dokument gar nicht geöffnet (z. B. Fehler in Werte-Formel)
- Testen auf Fehler mit @ISError und @IfError



@IsError

- Typischerweise als @If-Bedingung
- Beispiel:

```
_wert := @DbLookup("" : "NoCache"; ""; _view; _key; 2);  
@If(  
    @IsError(_wert);  
    Vorgabewert;  
    _wert  
)
```

- Typische Reaktionen
 - weiter mit Vorgabewert
 - Benutzer melden/fragen, z. B. mit @Prompt
 - Kombinationen mittels @Do



@IfError

- „syntaktischer Zucker“ (Kurzschreibweise)

@IfError(zu prüfender wert; Fehlerwert/-reaktion)

statt

```
@If(  
    @IsError(zu prüfender wert);  
    Fehlerwert/-reaktion;  
    zu prüfender wert  
)
```



@IfError (forts.)

- In Version 6 eingeführt, aber fehlerhaft implementiert, deshalb „deprecated“ [sic!] seit Version 7
- Probleme bei mathematischen Formeln und Gleichungen
- Ich benutze @IfError trotzdem gerne bei @DbColumn, @DbLookup u. ä.



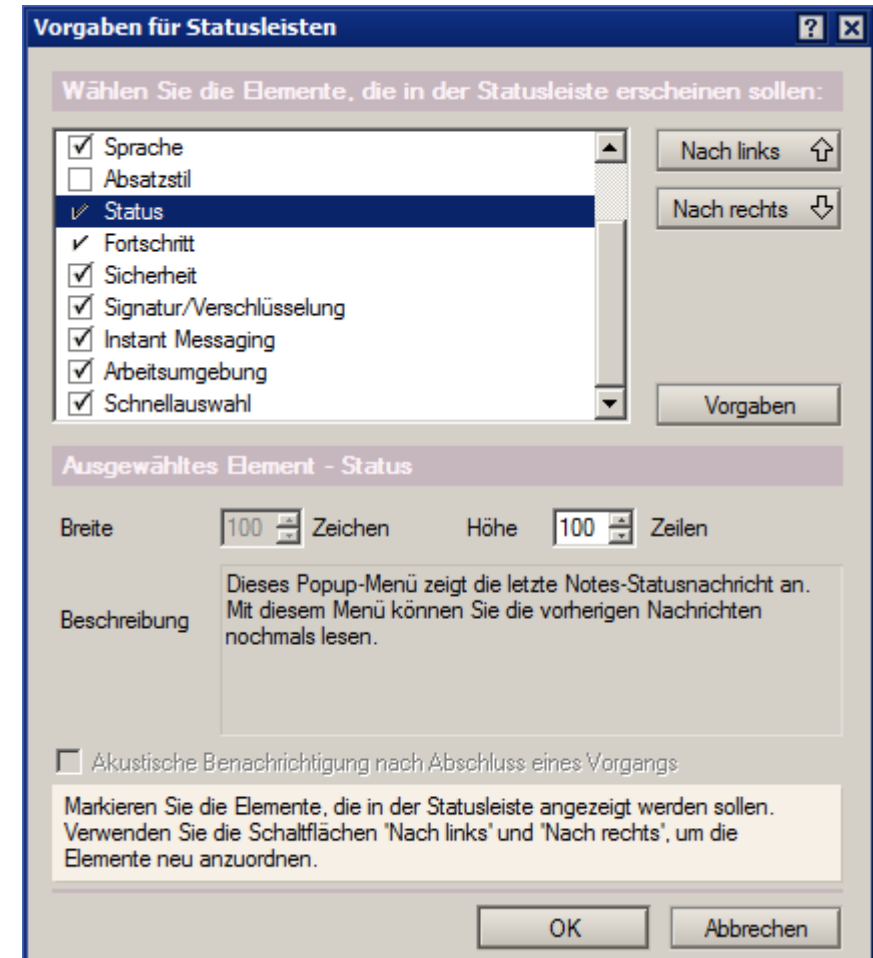
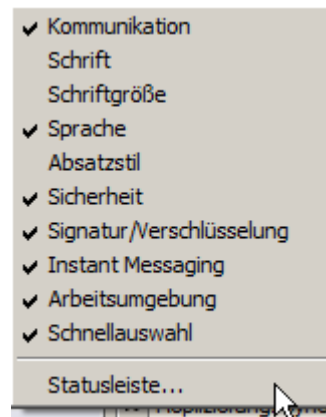
Defensive Formelsprache-Programmierung

- Testen mit
 - @IsTime – ist es ein Zeit-/Datumswert?
 - @IsNumber – ist es eine Zahl?
 - @Elements – wie viele Elemente hat die Liste?
 - @IsAvailable – gibt es das Feld überhaupt?
- Zahlen- bzw. Zeit-/Datumswert auf „Ist es leer?“ prüfen
 - @If(@Text(var) = ""); ...
- Fehler ausgeben
 - @Prompt([OK]; @Text(fehlerhafterwert))
 - @StatusBar(@Text(fehlerhafterwert))



Mehr Statuszeilen

- Im Basic-Client:
 - Im Kontextmenü der Statuszeile (Rechts-Klick):
 - Statusleiste...
 - Status
 - Höhe: 100 Zeilen





Protokollierung der Statuszeile aktivieren

- Im Standard-Client geht das leider nicht mehr
- Seit Version 7 neue notes.ini-Einstellung für Client:
LogStatusBar=1
www-10.lotus.com/ldd/dominowiki.nsf/dx/logstatusbar
- Protokolliert Statuszeilen-Ausgaben im lokaler log.nsf



Agenda

- Motivation
- Formelsprache
- **LotusScript**
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



LotusScript

- zentrale Fehlerbehandlung möglich
- Fehler steigen auf (wie Luftblasen) in der Prozedur-Aufruf-Hierarchie bis sie „gefangen“ werden
- Unbehandelte Fehler zeigt der Notes-Client in einer Dialogbox an



On Error

- On Error [fehlernummer]
 {GoTo label | Resume Next | GoTo 0}
- fehlernummer
 - optional
 - nur bei genau diesem Fehler reagieren (sonst bei allen)
- GoTo label
 - Ausführung beim Sprungziel fortführen
- Resume Next
 - Fehler ignorieren, mit nächster Zeile weiter machen
- GoTo 0
 - keine Fehlerbehandlung (deaktiviert früheres On Error)



Resume

- Innerhalb einer Fehlerbehandlung (nach GoTo Label)
- Resume 0
 - Ausführung fehlerhafter Zeile wiederholen
- Resume Next
 - Programmausführung mit nächster Zeile fortfahren
- Resume Label
 - an Sprungziel Label weiter machen
- End
 - Script-Ausführung sofort komplett abbrechen



Fehlernummern

- viele Fehlerkonstanten definiert in
 - Iserr.Iss
 - Isxbeerr.Iss
 - Isxuierr.Iss
- leider häufig „generischer“ Fehler 4000
 - Abbruch durch Benutzer
 - Feld zu groß
 - Netzwerkprobleme
 - ...



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



Basis-Fehlerdetails

- Err
 - Fehlernummer
- Er1
 - Fehlerzeile
- Error
 - Fehlertext



Mehr Fehlerdetails: GetThreadInfo

- **GetThreadInfo(infoID)**
 - gibt Informationen zum aktuellen Thread zurück
 - LSI_THREAD_LINE – aktuelle Zeile
 - LSI_THREAD_PROC – aktuelle Prozedur
 - LSI_THREAD_MODULE – aktuelles Modul
 - LSI_THREAD_VERSION – LotusScript-Version
 - LSI_THREAD_LANGUAGE – Spracheinstellung
 - LSI_THREAD_COUNTRY – Landeseinstellung
 - LSI_THREAD_TICKS – Zeitpunkt in Ticks
 - LSI_THREAD_TICKS_PER_SEC – Ticks pro Sekunde
 - LSI_THREAD_PROCESS_ID – aktuelle Prozess-ID
 - LSI_THREAD_TASK_ID – aktuelle Task-ID
 - LSI_THREAD_CALLPROC – aufrufende Prozedur
 - LSI_THREAD_CALLMODULE – aufrufendes Modul



Noch mehr Fehlerdetails: LSI_Info

- LSI_Info(infoID)
 - undokumentierte, nicht von IBM unterstützte Funktion
 - nicht thread-safe, kann zu Server-Abstürzen führen
www-304.ibm.com/support/docview.wss?uid=swg21237286
 - Warnung seit Domino 9:

You are using LSI_INFO in LotusScript which is an undocumented feature only used in a controlled environment, and is unsupported

LSI_INFO(14) was found and is known to cause memory corruption, leading to server instability. Please remove all uses of this from your application.



Noch mehr Fehlerdetails: LSI_Info (forts.)

- LSI_Info lässt sich weitgehend durch GetThreadInfo ersetzen, außer:
 - LSI_Info(14) – Stack Trace (seit Version 6)
 - LSI_Info(50) – LotusScript memory allocated
 - LSI_Info(51) – LotusScript memory allocated from OS
 - LSI_Info(52) – LotusScript blocks used
- Unsere Best Practice:
LSI_Info nur im Notes-Client benutzen:

```
If Not session.IsOnServer Then  
    stackTrace = Split(LSI_Info(14), Chr$(10))  
End If
```



LSI_Info(14) – Stack Trace

- LSI_Info(14) gibt einen String zurück
- mit Zeilenumbrüchen getrennte Zeilen
- jede Zeile besteht aus (Kommata getrennt):
 - Thread-ID
 - Prozedurname
 - Zeile
- Beispiel:
 - *F89C618, PRINTSTACKTRACE, 6
 - *F89C618, PROZEDUR_INNEN, 3
 - *F89C618, PROZEDUR_AUSSEN, 4
 - *F89C618, INITIALIZE, 4



Weitere Kontextinformationen

- aktuelle Datenbank
 - `session.CurrentDatabase`
- aktueller Benutzer
 - `session.UserName`
 - `session.EffectiveUserName`
- aktuelle Berechtigungen
 - `currentDB.CurrentAccessLevel`
 - `Evaluate(|@UserRoles|)`



Weitere Kontextinformationen (forts.)

- aktueller Agent
 - `session.CurrentAgent`
- Plattform
 - `session.Platform`
- Versionen
 - `session.NotesBuildVersion`
 - `session.NotesVersion`



Version der Anwendung/Schablone

- gespeichert in Items des gemeinsamen Felds \$TemplateBuild
 - \$TemplateBuild: Versionsnummer
 - \$TemplateBuildDate: Zeitpunkt
 - \$TemplateBuildName: Name der Schablone
 - \$TemplateBuildNr: (optional) eindeutige Build-Nummer
 - \$TemplateBuildTool: (optional) Build-Werkzeug
- Beispiel deutsche Mail-Schablone Notes 9.0.1:
 - \$TemplateBuild: "9.0"
 - \$TemplateBuildDate: 31.01.2013 12:17:27 EST
 - \$TemplateBuildName: "StdR9Mail"
 - \$TemplateBuildTool: "CICO"



Version der Anwendung/Schablone (forts.)

- \$TemplateBuild-Note holen:

```
Function GetTemplateBuildDoc(db As NotesDatabase) As NotesDocument
```

```
    Set GetTemplateBuildDoc = Nothing
```

```
    Set noteColl = db.CreateNoteCollection(False)
```

```
    noteColl.SelectSharedFields = True
```

```
    Call noteColl.BuildCollection
```

```
    noteID = noteColl.GetFirstNoteID
```

```
    For i = 1 To noteColl.Count
```

```
        Set doc = db.GetDocumentByID(noteID)
```

```
        Set item = doc.GetFirstItem("$Title")
```

```
        If item.Text = "$TemplateBuild" Then
```

```
            Set GetTemplateBuildDoc = doc
```

```
            Exit Function
```

```
        End If
```

```
        noteID = noteColl.GetNextNoteId(noteID)
```

```
    Next
```

```
End Function
```



Version der Anwendung/Schablone (forts.)

- \$TemplateBuild-Note auswerten:

```
Dim doc As NotesDocument
```

```
Set doc = GetTemplateBuildDoc(db)
```

```
If Not doc Is Nothing Then
```

```
    templateBuild = Cstr(doc.GetItemValue("$TemplateBuild")(0))
```

```
    templateBuildDate = Cstr(doc.GetItemValue("$TemplateBuildDate")(0))
```

```
    templateBuildName = Cstr(doc.GetItemValue("$TemplateBuildName")(0))
```

```
    templateBuildNr = Cstr(tbDoc.GetItemValue("$TemplateBuildNr")(0))
```

```
End If
```




Fehler werfen mit Error

- `Error fehlerNummer [, fehlerText]`
- `fehlerNummer`
 - frei wählbare Ganzzahl (Integer)
 - sinnvoll > 5.000, oberhalb der Standard-Fehlernummern
- `fehlerText`
 - optional
 - genauere Fehlerdetails mitgeben
- Best Practice:
 - Konstanten für eigene Fehlernummer zentral definieren und einheitlich verwenden

```
Public Const ERRNO_DB_OPEN_ERROR% = 11101
```



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - **Beispiele**
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



1. Beispiel: Löschen eines abhängigen Dokuments

- Dokument holen mit `db.GetDocumentByUNID(unid)`
- Wenn es kein Dokument mit der UNID gibt:
"Not matching the UNID to a document in the database raises `IsERR_NOTES_BAD_UNID (4091)`."
- Beispielfall: Prozedur soll Rechnungsdokument mit Rechnungsposition-Dokumenten löschen
- Wenn Rechnungsposition-Dokument nicht mehr vorhanden, einfach ignorieren und mit dem nächsten weiter machen



1. Beispiel (forts.)

```
Sub LöscheRechnungsposition(unid As String)
    Dim doc As NotesDocument
    On Error 4091 GoTo ignoreError ' IsERR_NOTES_BAD_UNID
    Set doc = Nothing
    Set doc = currentDB.GetDocumentByUNID(unid)
    If Not doc Is Nothing Then
        Call doc.Remove(True)
    End If
    Exit Sub
ignoreError:
    Resume Next
End Sub
```



2. Beispiel: Import aus einer Datei

- Dateiname mit Pfadangabe steht in der Konfiguration.
- Wenn es die Datei nicht gibt, soll der Benutzer eine andere auswählen können.
- Mit `Dir$` kann man prüfen, ob die Datei existiert.
- Aber wenn schon das übergeordnete Verzeichnis nicht existiert, wird Fehler 76 (`ErrPathNotFound`) geworfen.



2. Beispiel: Import aus einer Datei

```
Sub Import(dateiName As String)
    Dim dateiExistiert As Boolean
    On Error 76 GoTo errorHandlerFilepathNotFound
    dateiExistiert = (Dir$(dateiName) <> "")
    Do While Not dateiExistiert
        ' Benutzer andere Datei auswählen lassen
        dateiExistiert = (Dir$(dateiName) <> "")
    Loop
    ' importiere Datei
Exit Sub

errorHandlerFilepathNotFound:
    dateiExistiert = False
    Resume Next
End Sub
```



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - **Fehlerbehandlungsstrategie**
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



Fehlerbehandlungsstrategie

- Fehler werden die Aufruf-Hierarchie hoch gereicht, wenn sie nicht mit `On Error` „gefangen“ werden.
- sinnvolle Fehlerbehandlung meist nur direkt/nah an der Quelle möglich
 - Beispiel Dateizugriffsfehler: Benutzer andere Datei wählen lassen, danach noch einmal versuchen
- allgemeine Fehlerbehandlung, z. B. Protokollierung möglichst zentral, ganz „oben“



Fehlerbehandlungsstrategie (forts.)

- kein Resume, wenn der Fehler nicht behandelt wurde
- außer „ganz oben“, wenn
`GetThreadInfo(LSI_THREAD_PROC) =`
`GetThreadInfo(LSI_THREAD_CALLPROC)`
- auch Exit Function/Exit Sub „frisst“ den Fehler nach einem `On Error GoTo label`



Fehlerprotokollierung

- Wohin?
 - E-Mail-Benachrichtigung
 - Pro: hohe Aufmerksamkeit
 - Contra: Vergänglichkeit, keine Historie
 - gleiche Datenbank
 - Pro: nahe am Kontext, zusammenhängende Historie
 - zentrale Datenbank
 - Pro: eine einzige Stelle zu überwachen
 - Contra: unübersichtlich, überladen
- Unsere Best Practice:
 - immer protokollieren in aktueller Datenbank
 - Benutzer fragen, ob zusätzlich E-Mail-Benachrichtigung



Defensive Programmierung mit LotusScript

- Option Declare
 - Variablen müssen deklariert werden
 - verhindert versehentliche Verschreiber
 - seit Version 6 im Designer voreinstellbar
- Ergebnisse von Funktionsaufrufen prüfen, z. B.

```
Set doc = view.GetDocumentByKey(key)
If Not doc is Nothing Then
    MessageBox doc.Title(0)
Else
    ' Fehler melden, Vorgabe-Dokument anlegen, ...
End If
```

- Übergabeparameter in Prozeduren prüfen



Defensive Programmierung mit LotusScript (forts.)

- Bedingungen prüfen z. B. mit Hilfsfunktion Assert:
<http://www.ibm.com/developerworks/lotus/library/ls-DebugLS2/>

```
Sub Assert(condition As Boolean, message As String)
  Dim callerName As String
  If Not condition Then
    callerName = GetThreadInfo(LSI_THREAD_PROC)
    Stop ' in case the user has the debugger on.
    MsgBox "Assert failure in " & callerName & ": " & _
      message, 16, MSG_TITLE
    End ' abort execution of this script
  End If
End Sub

Assert Datatype(cutoffDate) = V_DATE,
      "cutoffDate must be a date/time value"

Assert cutoffDate < Today,
      "cutoffDate must be earlier than today"
```



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte

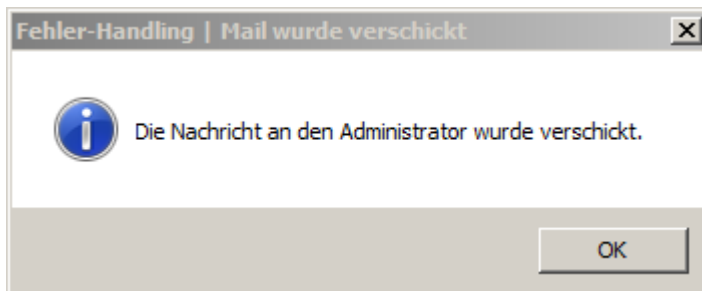
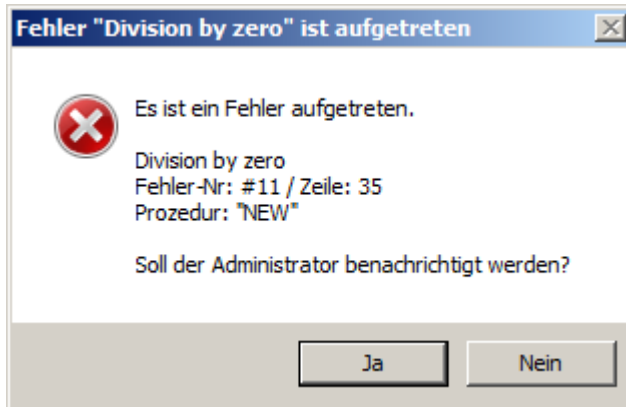



Fehlerbehandlung im assono Framework 2

- Unsere Best Practice:
 - zentrale Liste von Fehlerbehandlungsroutinen
 - jede Routine (ErrorHandler) prüft, ob sie zuständig ist
 - jede Routine gibt zurück, ob der Fehler abschließend behandelt wurde oder wie weiter verfahren werden soll
 - jede Routine temporär (de-)aktivierbar
 - zentrale Skript-Bibliothek "Error handling utils" installiert allgemeine Behandlungsroutinen im Initialize
 - zusätzliche Fehlerbehandlungen können in Skript-Bibliotheken oder einzelnen Routinen installiert und genutzt werden
 - siehe <http://www.openntf.org/p/assono+Framework+2>



Bei einem Fehler...






assCRM 2 (Schablone) - Fehler in der Procedure "NEW"
assCRM 2 (Schablone) An: tbahn
Gesendet von: **Thomas Bahn**

Es ist ein Fehler aufgetreten.

Technischer Fehler

Datenbank: assCRM 2 (Schablone)
!!D:\Notes\Projekte\assono\assCRM21.ntf 
Replik-ID: C125780D0032B0DF
[Notes:///C125780D0032B0DF](#)

Schablonen-Version
Name: assCRM-2
Build: 2.1.1
Datum: 21.02.2014
Build-Nr: 6

Fehlernummer: 11
Division by zero
Fehler-Level: ERROR
Zeile 35 von der Prozedur "NEW"

Trace: Zeile 35 in Prozedur "NEW"
Zeile 18 in Prozedur "CREATEOPPTYCONTROLLER"
Zeile 20 in Prozedur "QUERYOPEN"

User-Informationen
Benutzername: CN=Thomas Bahn/O=assono
Effektiver Benutzername: CN=Thomas Bahn/O=assono

Zugriffslevel:



Bei einem Fehler... (forts.)

Protokoll

Allgemeine Angaben

Zeitpunkt	17.03.2014 08:26:21
Absender	Standard logging handler
Zusammenfassung	Fehler #11 ist aufgetreten

Technischer Fehler

Fehlernummer	11
Text	Division by zero
Level	ERROR
Prozedur	NEW
Zeile	35
Stack-Trace	Zeile 35 in Prozedur "NEW" Zeile 18 in Prozedur "CREATEOPPTYCONTROLLER" Zeile 20 in Prozedur "QUERYOPEN"

Verknüpfungen (Fehlerkontext)

[Datenbank](#) | [Benutzer](#) | [System](#) | [Schablone](#) | [Properties](#)

Datenbankinformationen

Titel	assCRM 2 (Schablone)
Server	
Dateiname und -Pfad	D:\Notes\Projekt\assono\assCRM21.ntf
Replik-ID	C125780D0032B0DF
Zugriffslevel	Manager



Bei einem Fehler... (forts.)

Datenbank	Benutzer	System	Schablone	Properties
Benutzerinformationen				
Benutzername	Thomas Bahn/assono			
Effektiver Benutzername	Thomas Bahn/assono			
Benutzerrollen	[Entwickler], [Jeder], [KontaktNeu], [APNeu], [FirmaNeu], [FirmaLeser], [APLeser], [KontaktBearb], [Admin], [Verwalter], [FirmaBearb], [APBearb], [KontaktLeser], [WvIBearb], [WvILeser], [WvINeu], [Weihnachten], [BeschwerdeBearb],			

Datenbank	Benutzer	System	Schablone	Properties
Systeminformationen				
Eingestellte Sprache	de			
Engestelltes Land	0			
Plattform	Windows/32			
Notes-Version	Release 9.0.1 October 14, 2013			
Notes-Build-Version	405			
LotusScript-Version	5.0.0.08E			
Zeitpunkt des Fehlers	235278804			
Zeitliche Auflösung	1000			
Aktueller Prozess	34472			
Aktueller Task				
Reservierter Speicher LS (Bytes)				
Reservierter Speicher OS (Bytes)				
Genutzer Speicher (Blöcke)				

Datenbank	Benutzer	System	Schablone	Properties
Schabloneninformationen				
Name	assCRM-2			
Build	2.1.1			
Datum	21.02.2014			
Build	6			



Anwendung des Frameworks

- Verwendungsbeispiel:

```
Sub XYZ
```

```
  If Not IsDebugMode Then On Error Goto errorHandler
```

```
  ...
```

```
  Exit Sub
```

```
errorHandler:
```

```
  If HandleError() = RESUME_NEXT_LINE Then Resume Next  
  Call RethrowError()
```

```
End Sub
```

- RethrowError sorgt dafür, dass unbehandelte Fehler hoch gereicht werden.
- IsDebugMode über Konfiguration steuerbar, erlaubt Nutzung des Debuggers ohne Fehlerbehandlung



Anwendung des Frameworks (forts.)

- Minimal-Version (z. B. Skript-Bibliothek):

Use "Error handling utils"

```
Sub Initialize
```

```
    Call CreateAndRegisterStandardLoggingHandler()
```

```
    Call CreateAndRegisterStdMessageBoxHandler()
```

```
    If Not IsDebugMode Then On Error Goto errorHandler
```

```
    ...
```

```
Exit Sub
```

```
errorHandler:
```

```
    If HandleError() = RESUME_NEXT_LINE Then Resume Next
```

```
    Call RethrowError()
```

```
End Sub
```



Und jetzt in automatisch...



Benutzervorgaben

Code

- Domino Designer
 - LotusScript-Editor
 - Codeschablonen**
 - Java
 - JavaScript
 - Web

Codeschablonen

Schablone automatisch in neue Codeelemente aufnehmen

Codeelement:

- Design Element
- Class
- Type
- Sub**
- Function
- Property Get
- Property Set

Codeschablone:

```
/*  
 * .  
 *  
 * @param xxx .  
 *  
 * @author ${author} <${user}@assono.de>  
 * @version  ${year}-  
 */  
  
If Not IsDebugMode Then On Error Goto errorHandler  
  
Exit Sub  
  
errorHandler:  
If HandleError() = RESUME_NEXT_LINE Then Resume Next  
Call RethrowError()
```

OK Abbrechen



Fehlerbehandlung im assono Framework 2 (forts.)

- weitere HandleError-Varianten:

`HandleErrorWithContext(CreateErrorContext(doc, message))`

`HandleErrorWithLevel(level)`

`HandleErrorWithLevelWithContext(level, errorContext)`

`HandleErrorBackEnd()`

`HandleErrorBackEndWithContext(errorContext)`

- Fehler-Level:

- `ERROR_LEVEL_WARN`
- `ERROR_LEVEL_ERROR`
- `ERROR_LEVEL_FATAL`



ErrorHandler

- ErrorHandler definieren Reaktionen auf Fehler
- Attribute
 - Status: aktiv/nicht aktiv
 - Priorität: bestimmt die Reihenfolge der Aufrufe
 - RunOnce: darf der ErrorHandler mehrfach für den gleichen Fehler aufgerufen werden



ErrorHandler (forts.)

- vordefinierte ErrorHandler
 - EmailNotification
 - sendet E-Mail mit Fehlerdetails und -kontext
 - IgnoreErrorHandler
 - ignoriert einen bestimmten Fehler
 - LoggingHandler
 - schreibt Fehlerdetails in ein Protokoll (siehe Logger)
 - MessageBoxHandler
 - zeigt dem Benutzer eine Fehlermeldung
 - WebAgentErrorHandler
 - WebFormErrorHandler
- ErrorHandler sind beliebig erweiterbar durch eigene Unterklassen!



Eigene ErrorHandler

- Unterklassen von ErrorHandler brauchen nur zwei Methoden zu überschreiben:

Private Function CheckResponsibility(errorDetails As ErrorDetails) As Boolean

Private Function DoHandleError(errorDetails As ErrorDetails) As Integer

- CheckResponsibility soll True zurückgeben, wenn der ErrorHandler sich für diesen Fehler interessiert; nur dann wird DoHandleError aufgerufen
- DoHandleError behandelt den Fehler und gibt zurück, wie es weiter gehen soll:



Eigene ErrorHandler (forts.)

- (Do)HandleError-Rückgabewerte:
 - ERROR_NOT_HANDLED
der nächste ErrorHandler soll sich darum kümmern
 - RESUME_SAME_LINE
 - RESUME_NEXT_LINE
 - RESUME_AT_LABEL
 - EXIT_FUNCTION
 - END_PROGRAM



Logger

- Logger definieren, wo und wie protokolliert wird
- Attribute
 - Status: aktiv/nicht aktiv
 - minimaler Level, für den protokolliert wird
 - maximale Anzahl Protokolleinträge zur gleichen Fehlerart innerhalb eines definierten Zeitfensters
- vordefinierte Logger
 - DBLogger
 - protokolliert in eine Notes-Datenbank
 - EMailLogger
 - sendet eine E-Mail mit Fehler-Informationen



Eigene Logger

- Logger beliebig über Unterklassen erweiterbar, z. B. SMSLogger, SQLDBLogger, windowsEventsLogger oder webserviceLogger
- Unterklassen von Logger brauchen nur eine Methode zu überschreiben:

Private Sub doLog(errorDetails As ErrorDetails)

- doLog schreibt ins Log (wohin auch immer)



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- **Java**
- JavaScript
- XPages
- Open Source-Projekte



Java

- detailliertes Konzept für Fehler und Ausnahmefälle
- Klassenhierarchie
 - Throwable (abstrakte Oberklasse)
 - Error
 - Exception
 - RuntimeException
- und ganz viele weitere Unterklassen



Errors

- unvorhersehbar, eher „von außen“
- selten sinnvoll zu behandeln
- sollen auch nicht gefangen werden:
„An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.“
- Beispiele:
 - `OutOfMemoryError`
 - `StackOverflowError`
 - `NoClassDefFoundError`
 - `ThreadDeath`



Exceptions

- prinzipiell zumindest vorhersehbar, eher „von innen“
- sinnvolle Behandlung manchmal möglich
- Beispiele:
 - `IllegalAccessException`
 - `IOException`
 - `RemoteException`
 - `TimeoutException`
 - Spezialfall: `RuntimeException`



Checked Exceptions vs. RuntimeExceptions

- Checked Exceptions sind alle Ausnahmen, die nicht RuntimeException oder deren Unterklassen sind.
- Checked Exceptions **müssen** gefangen werden, sonst gibt es einen Fehler beim Kompilieren.
- RuntimeExceptions werden in der Regel nicht gefangen.



Fehlerbehandlung mit try – catch

- Code, der einen Fehler werfen könnte, wird in einem try-Block eingeschlossen, der mögliche Fehler im catch gefangen und ggf. behandelt

```
try {  
    ...  
} catch (Exception e) {  
    ...  
}
```

- mehre Catches möglich, vom Speziellen zum Allgemeinen, der erste passende „gewinnt“

```
} catch (IOException ioe) {  
    ...  
} catch (Exception e) {  
    ...  
}
```



try – catch – finally

- Manchmal muss Code unabhängig davon ausgeführt werden, ob ein Fehler auftrat oder nicht, z. B. um Ressourcen freizugeben.
- Dafür gibt es den optionalen `finally`-Block:

```
try {  
    ...  
} catch (Exception e) {  
    ...  
} finally {  
    ... // wird auf alle Fälle ausgeführt  
}
```



Eigene Exceptions

- Für fachliche Fehler ~~kann~~ sollte man eigene Ausnahme-Unterklassen definieren, z.B.
 - CustomerInvalidException
 - CustomerLoadingException

```
class KeywordNotFoundException extends RuntimeException {
    private String keyword;
    public KeywordNotFoundException (String message,
                                     String keyword) {
        super(message);
        this.keyword = keyword;
    }
    public String getKeyword() {
        return this.keyword;
    }
}
```



Eigene Exceptions (forts.)

- Ob von `Exception` oder `RuntimeException` abgeleitet wird, ist „Geschmackssache“ und wird in der Java-Community immer wieder heiß diskutiert. 😊
- Eigene, sinnvoll benannte Ausnahme-Klassen erhöhen die Lesbarkeit und Wartbarkeit des Codes und erlauben eine einfache, gezielte Behandlung von Fehlern.



Hochreichen von Fehlern

- Checked Exceptions müssen gefangen werden, auch wenn man sie nicht sinnvoll behandeln kann.
- Best Practice: Hochreichen als RuntimeException

```
try {  
    ...  
} catch (Exception e) {  
    throw new RuntimeException(e);  
}
```

- natürlich nur, wenn man nicht sinnvoll reagieren kann
- mit `e.getCause()` kann man später auf das innere Ausnahme-Objekt zugreifen



NotesException

- NotesException erweitert
org.omg.CORBA.UserException
- ist also eine Checked Exception
- Deklaration der Klasse:

```
public final class NotesException extends UserException {  
    public int id;  
    public String text;  
    public NotesException();  
    public NotesException(int _id, String _text);  
}
```



NotesError

- NotesError ist **keine** Unterklasse von Error [sic!], sondern ein Interface
- Container für Fehler-Konstanten
- Deklaration des Interfaces:

```
interface NotesError {  
  
    const long NOTES_ERR_ERROR = 4000;  
    const long NOTES_ERR_SYS_OUT_OF_MEMORY = 4001;  
    const long NOTES_ERR_SYS_LOAD_OUT_OF_MEM = 4002;  
    const long NOTES_ERR_SYS_FILE_NOT_FOUND = 4003;  
    const long NOTES_ERR_SYS_DICT_NOT_ON_PATH = 4004;  
  
    ...  
}
```



Behandlung von Notes-Ausnahmen

- Fallunterscheidung unelegant durch if-Kaskade statt getrennter catch-Blöcke

```
try {  
    ...  
} catch (NotesException e) {  
    if (e.id == NotesError.NOTES_ERR_SYS_FILE_NOT_FOUND)  
        // lass den Benutzer eine andere Datei wählen o.ä.  
    else  
        // Behandlung anderer Notes-Ausnahmen  
} catch (Exception e) {  
    // Behandlung sonstiger Ausnahmen  
}
```




Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- **JavaScript**
- XPages
- Open Source-Projekte



JavaScript

- Fehlerbehandlung ähnlich wie bei Java
- keine Klassen, deshalb genau ein catch-Block

```
try {  
    ...  
} catch (e) {  
    ...  
} finally {  
    ...  
}
```

- finally-Block wie bei Java optional



throw

- throw ausdruck wirft neuen Fehler
 - ausdruck kann ein String sein
 - oder eine Zahl
 - oder ein (Fehler-)Objekt
- Beispiele

```
throw 'invalid value for parameter month: ' + month;
```

```
throw new InvalidParameterError('month', month);
```



Error-Typen

- Neben Error gibt es folgende vordefinierten Fehler:
 - EvalError: Fehler bei Ausführung von eval()
 - RangeError: numerischer Wert oder Parameter außerhalb des erlaubten Bereichs
 - ReferenceError: ungültige Referenz
 - SyntaxError: Syntaxfehler beim Parsen von eval()-Code
 - TypeError: Variable oder Parameter hat falschen Typ
 - URIError: ungültiger Wert bei encodeURIComponent() oder decodeURI()



Eigene Fehler-Objekte

- Man kann eigene Fehler-Objekte mit zusätzlichen Attributen erstellen, z. B. für fachliche Fehler

```
function InvalidParameterError(parameter, value) {  
    this.parameter = parameter;  
    this.value = value;  
    this.message = 'Der Parameter ' + this.parameter +  
        ' hat den ungültigen Wert ' + this.value;  
  
    this.toString = function() {  
        return this.message;  
    }  
}
```



Nicht behandelte Fehler hoch reichen

- Wie in Java sollte man Fehler, die man gefangen hat, aber nicht behandeln konnte, neu werfen:

```
try {  
    ...  
} catch (e) {  
    if (e instanceof InvalidParameterError) {  
        ...  
    } else {  
        // cannot handle this exception, so rethrow  
        throw e;  
    }  
}
```



window.onerror

- zentrale Fehlerbehandlung im Browser installieren:

```
window.onerror = function (message, url, line) {...}
```

- Funktion wird bei Fehlern vom Browser aufgerufen
 - message: Fehlermeldung
 - url: aktuelle Seite
 - line: aktuelle Zeile
- Rückgabewerte
 - true: Fehler ist behandelt, nichts weiter machen
 - false: Fehler ist nicht behandelt, der Browser zeigt dem Benutzer die Standard-Fehlermeldung an



Beispiel für Fehlerbehandlung im Browser

- Beispiel

```
function trapError(message, url, line) {  
    var error = new Error(message);  
    error.location = url + ', line:' + line;  
    logJSError(error);  
    return (!SHOW_ERRORS);  
}  
  
window.onerror = trapError;
```




Beispiel für Fehlerbehandlung im Browser (forts.)

```
function logJSError(jsError) {  
var url = '/' + webpath + '/JSFehlerProtokoll?CreateDocument';  
var postArguments = 'Error=' + escape(jsError.name)  
+ '&Message=' + escape(jsError.message)  
+ '&Location=' + escape(jsError.location)  
+ '&Referer=' + escape(location.href)  
+ '&AppCodeName=' + escape(navigator.appCodeName)  
+ '&AppName=' + escape(navigator.appName)  
+ '&AppVersion=' + escape(navigator.appVersion)  
+ '&CookieEnabled=' + escape(navigator.cookieEnabled)  
+ '&Language=' + escape(navigator.language)  
+ '&Platform=' + escape(navigator.platform)  
+ '&UserAgent=' + escape(navigator.userAgent)  
+ '&JavaEnabled=' + escape(navigator.javaEnabled())  
+ '&Cookie=' + escape(document.cookie);  
... // Ajax-Call, erstellt JSFehlerProtokoll-Dokument auf Server  
}
```



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- Open Source-Projekte



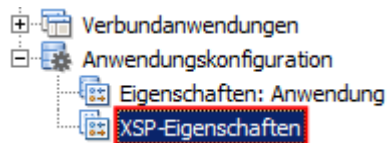
XPages

- je nach verwendeter Programmiersprache:
 - siehe JavaScript für CSJS (Client Side JavaScript) und SSJS (Server Side JavaScript)
 - siehe Java



Eigene Fehlerseite für XPages-Anwendungen

- Fehlerseite wird über die XSP-Eigenschaften innerhalb der Anwendungskonfiguration festgelegt:



XPage-Eigenschaften

Motivstandardeinstellungen

Anwendungsmotiv:	Plattform-Standard-einstellung
<input type="checkbox"/> Im Web überschreiben:	Anwendungsstandard-einstellung
<input type="checkbox"/> Unter Notes überschreiben:	Anwendungsstandard-einstellung
<input type="checkbox"/> Mobile-Motiv für XPages mit folgendem Prefix verwenden:	m_
Mobile-Motiv:	Mobile-Standard-einstellung
<input type="checkbox"/> Unter iOS überschreiben:	Anwendungsstandard-einstellung
<input type="checkbox"/> Unter Android überschreiben:	Anwendungsstandard-einstellung
<input type="checkbox"/> Benutzeragent debuggen:	

Fehlerbehandlung

<input type="checkbox"/> XPage-Laufzeitfehlerseite anzeigen	
Fehlerhafte Seite:	Fehlerseite

Zeitlimits

Allgemein | Persistenz | Seitengenerierung | Quelle



Eigene Fehlerseite für XPages-Anwendungen (forts.)

- „XPage-Laufzeitfehlerseite“ sehr informativ für Entwickler
- Eigene Fehlerseite ist benutzerfreundlicher und im gewünschten Design.
- Fehlerseite wird nur beim Full-Refresh angezeigt.
- Bei Partial-Refresh erscheint ein Fehlerdialog.
- Fehler kann über `requestScope.error` werden.
 - Rückgabewert ist ein Exception-Objekt.



Eigene Fehlerseite für XPages-Anwendungen (forts.)

```
try {
    var stackTrace = "";
    var error = requestScope.error;
    var message = "";

    if (error != null) {
        var trace = error.getStackTrace();
        message = '<div class="...">' + error.toString() + '</div>\n';

        while (typeof(error) != "NotesException" && error.getCause() != null) {
            trace = error.getStackTrace();
            error = error.getCause();
            message += '<div class="...">caused by ' + error.toString() + '</div>\n';
        }
        for( var i = 0; i < trace.length; i++) {
            stackTrace += trace[i] + '<br>\n';
        }
        return message + '<div class="...">' + stackTrace + '</div>';
    } else {
        return "No error object found";
    }
} catch(e){
    print(e.toString());
}
```



xp:eventHandler onError

- Im xp:eventHandler kann CSJS-Code definiert werden, der im Fehlerfall aufgerufen werden soll.

```
<xp:eventHandler
  event="onclick"
  submit="true"
  refreshMode="partial"
  refreshId="panel1">
  <xp:this.action>
  <![CDATA[#{javascript:thereIsNoSuchFunction();}]]>
  </xp:this.action>
  <xp:this.onError>
  <![CDATA[console.log(arguments);
    XSP.error("Es ist ein Fehler aufgetreten.\nBitte
      informieren Sie den Benutzerservice.");]]>
  </xp:this.onError>
</xp:eventHandler>
```



Agenda

- Motivation
- Formelsprache
- LotusScript
 - Fehlerdetails und –kontext ermitteln
 - Beispiele
 - Fehlerbehandlungsstrategie
 - Fehlerbehandlung im assono Framework 2
- Java
- JavaScript
- XPages
- **Open Source-Projekte**



Open Source-Projekte

- OpenLog ist sehr beliebtes Projekt auf OpenNTF.org
www.openntf.org/p/OpenLog
- Ergänzung speziell für XPages
www.openntf.org/p/XPages+OpenLog+Logger
- Flow
flow.openntf.org/
- alle drei beschränkt auf Fehler**protokollierung**
- assono Framework 2, darin die "Error handling utils"
www.openntf.org/p/assono+Framework+2



Fragen?

- jetzt stellen – oder später:

 tbahn@assono.de

 www.assono.de/blog

 04307/900-401



- Folien unter:
www.assono.de/blog/d6plinks/EC-2014-Fehlerbehandlung-in-Notes