Entwicklercamp 2010

Lotus knows.

Smarter software for a Smarter Planet.

Track 2, Session 5
Add-ons für Client/Designer auf Basis neuer Java APIs von Lotus Notes® 8.5.1!

Karsten Lehmann | Geschäftsführer Mindoo GmbH

NOTES & DOMINO ENTWICKLERCAMP

Agenda

- Einführung
- Lotus Notes mit Java erweitern
- Mehrwert für das klassische Lotus Notes
- Domino Designer mit Java erweitern
- Best Practices
- Q&A

Über uns

- Mindoo ist IBM Business Partner und Notes/Domino Design Partner
- Wir konzentrieren uns auf die „neuen“ Aspekte der IBM Lotus Notes-Entwicklung
  - Eclipse/Expeditor-Plugins und Rich-Client-Anwendungen
  - Composite Application Architekturen
  - LiveText Erweiterungen
  - Xpages-Anwendungen
- Karsten Lehmann und Tammo Riedinger
  - Gründer der Mindoo GmbH
  - Seit 2004 Entwickler der Applikation MindPlan® für die Haus Weilgut GmbH, Mindmapping und Projekt-Management für Lotus Notes, IBM Award Winner 2008
- Weitere Informationen: http://www.mindoo.de

[mindu:] Mindoo GmbH

Motivation

- „Notes kann auch das“ - ja, aber wieso so kompliziert?
  - Eclipse Entwickler mussten bisher zu viele Workarounds nutzen, um mit der klassischen Notes UI interagieren zu können
  - Man denke nur an “Nutzung der Notes.ini zum Datenaustausch” oder “Seiten öffnen, die sich selbst schließen, nur um LotusScript®-Code im QueryClose-Event auszuführen”
  - Es fehlte eine Java-API zur Ansteuerung der Notes-Oberfläche. Der Sinn der neuen APIs ist, diese Lücke zu füllen und den Entwicklern damit das Leben zu erleichtern
- Wir waren an den Diskussionen über die Features der neuen „Notes and Designer extensibility APIs“ beteiligt
  - Diskussionen mit IBM dev über API-Entwürfe bereits auf der Lotusphere 2009 und später in Konferenzschaltungen
  - Feedback zum Design / Berichte über Tests im Zuge des Design Partner Programms
- Wie immer bei ersten Releases gibt es noch Raum für Verbesserungen
  - Wir denken jedoch, dass diese APIs schon ein **riesiger Schritt** vorwärts sind

Achtung

- Die hier gezeigten Java APIs funktionieren nicht in Notes-Agenten. Dies is reine Eclipse-Entwicklung!

- Bitte gehen Sie noch nicht, falls Sie nur mit LotusScript entwickeln. Wir haben später auch noch etwas für Sie!

Achtung

- Wir können hier nicht alle APIs im Detail betrachten!
  - Wir werden uns stattdessen auf die wichtigsten Neuerungen konzentrieren: die neue Java UI API und die Designer Java-API
- Wir wollen Ihnen einen guten Eindruck vermitteln, was sie wirklich mit den Notes 8.5.1 APIs tun können
  - Hoffentlich verlassen Sie diesen Vortrag bereits mit einigen eigenen Idee
  - Wir haben ca. 10 Demos für Sie erstellt!
- Wir versuchen Sie heute nicht mit Code zu Tode zu langweilen!
- Verfolgen Sie stattdessen die kommende Serie von Artikeln mit Details zu den Demos in unserem Blog: http://blog.mindoo.com

Startvorbereitungen

- Die APIs und Demos basieren auf
  - Eclipse 3.4.2
  - Expeditor Toolkit 6.2.1
  - IBM Lotus Notes 8.5.1
- Installieren Sie Expeditor in Eclipse und setzen Sie Lotus Notes 8.5.1 als Ziel-Plattform

- Erstellen Sie ein Plugin-Projekt für Ihren eigenen Code
- Für die UI API müssen Sie die folgende Abhängigkeit definieren:
  - com.ibm.notes.java.ui
- Für die DDE API müssen Sie die folgenden Abhängigkeiten definieren:
  - com.ibm.designer.domino.ui.commons
  - com.ibm.designer.domino.ide.resources

Agenda

- Einführung
- Lotus Notes mit Java erweitern
- Mehrwert für das klassische Lotus Notes
- Domino Designer mit Java erweitern
- Best Practices
- Q&A

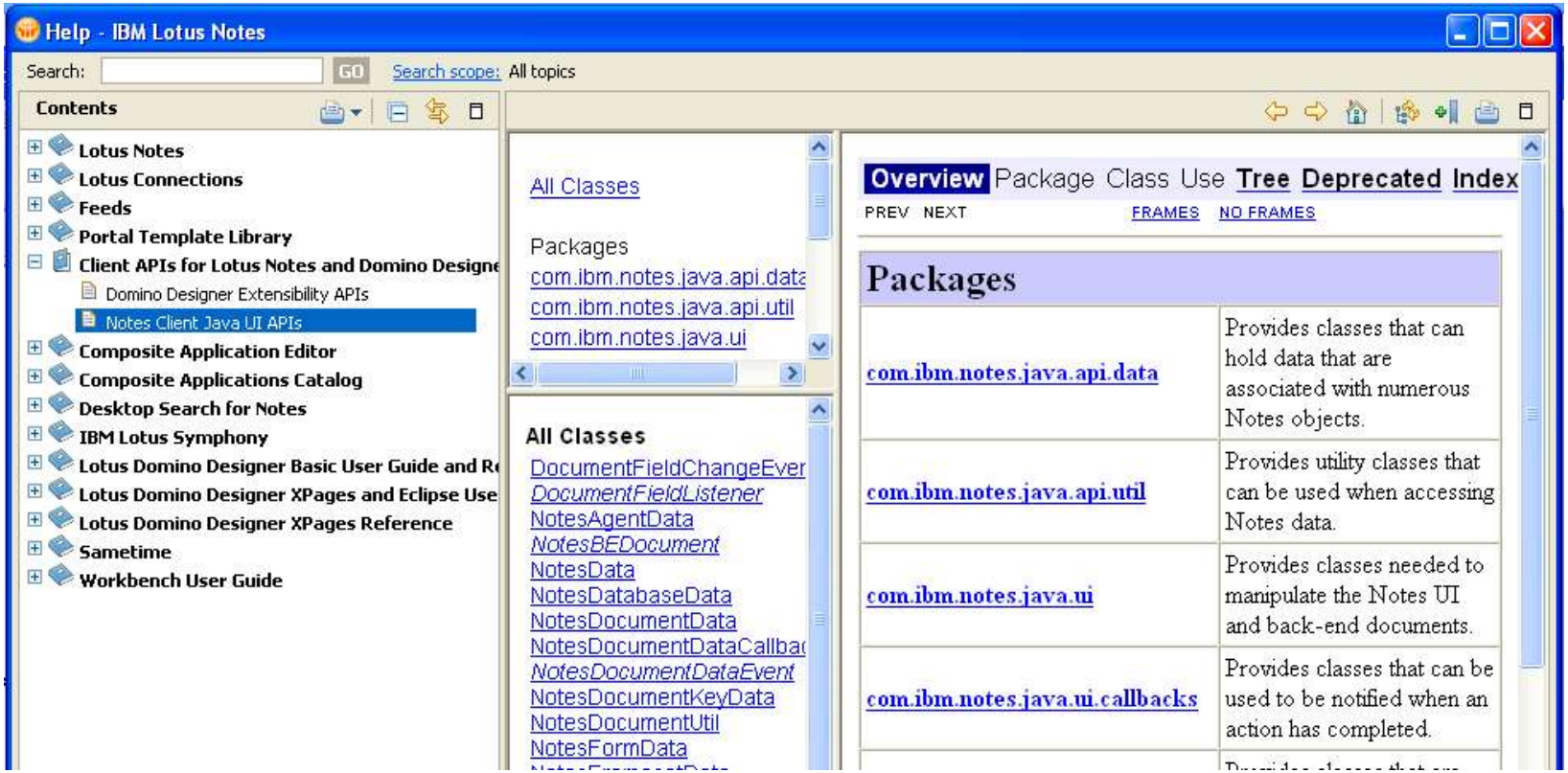
Java UI API – eine grobe Übersicht

- API ist neu in 8.5.1
  - Nicht supportete Vorschau bereits verfügbar in 8.5
- Verbessert die Integration zwischen Eclipse und dem klassischen Lotus Notes Client
  - Eclipse Plugins können endlich Informationen über den Zustand von Formularen und Ansichten des klassischen Clients bekommen
  - Neue Wege der Interaktion dieser beiden Welten
- Bestehende Funktionalität lässt sich nun einfacher nutzen
  - Öffnen von Design-Elementen
  - Drucken von Dokumenten und Ansichten aus Eclipse heraus
  - Erstellen neuer Dokumente und Vorbesetzen von Werten
  - Temporäre Dokumente, um Daten zu speichern und zu übertragen
  - Ausführen von Notes-Code in einem Hintergrund-Thread inklusive Speicher-Management (NotesSessionJob)

Java UI API – eine grobe Übersicht

- Ergänzt neue Funktionalitäten und verkleinert die Lücke zwischen Eclipse und dem klassischen Lotus Notes client
  - Lesen/Modifizieren von Dokumenten-Inhalt im Bearbeiten-Modus
  - Document Listener (Bearbeiten-Modus an/aus, Änderungen und Schließen von Dokumenten verfolgen)
  - Eclipse-Selection für fokussierte Felder und neue noch nicht gespeicherte Dokumente
  - Datenbank zu Workspace hinzufügen
  - Eingabe-Methoden von NotesUIWorkspace, z.B. um Datenbanken auszuwählen
  - Viele Eclipse Property Tester (eine Art "hide when" für Eclipse-Elemente wie z.B. Actions)
  - LotusScript-Agenten in UI-Thread ausführen (es können Dialoge angezeigt werden in den Agenten), Daten können bi-direktional übergeben und es können Callbacks installiert werden

Wo ist die Dokumentation?



Klein anfangen – Ihr erstes Beispiel

- Toolbar-Aktion zum Ändern eines Feldes in einem geöffneten Formular
- Nutzen Sie diesen Code in einer standard Eclipse Toolbar-Aktion:

```
NotesUIWorkspace ws = new NotesUIWorkspace();
NotesUIDocument uidoc = ws.getCurrentDocument();
if (uidoc != null) {
   NotesUIField field = uidoc.getField("Subject");
   if (field != null)
      field.setText("Hello World!");
}
```


Klein anfangen – Ihr erstes Beispiel

- Für Felder in Backend-Dokumenten:

```
NotesUIWorkspace ws = new NotesUIWorkspace();
NotesUIDocument uidoc = ws.getCurrentDocument();
if (uidoc != null) {
   NotesBEDocument beDoc = uidoc.getBEDocument();
   String oldValue = beDoc.getItemString("Flag");
   // do something here
   beDoc.setItemValue("Flag", "1");
   //optional to see your changes in the UI:
   uidoc.reload();
}
```

- Kein voller Zugriff auf das Dokument durch technische Restriktionen

Demo

- Mit Eclipse-Aktionen auf ein UI-Dokument zugreifen und dieses modifizieren:
  - Alternativer Address Picker für Mail-Adressaten
  - Ausfüllhilfe für Felder

Tiefer einsteigen - NotesUIWorkspace erkunden

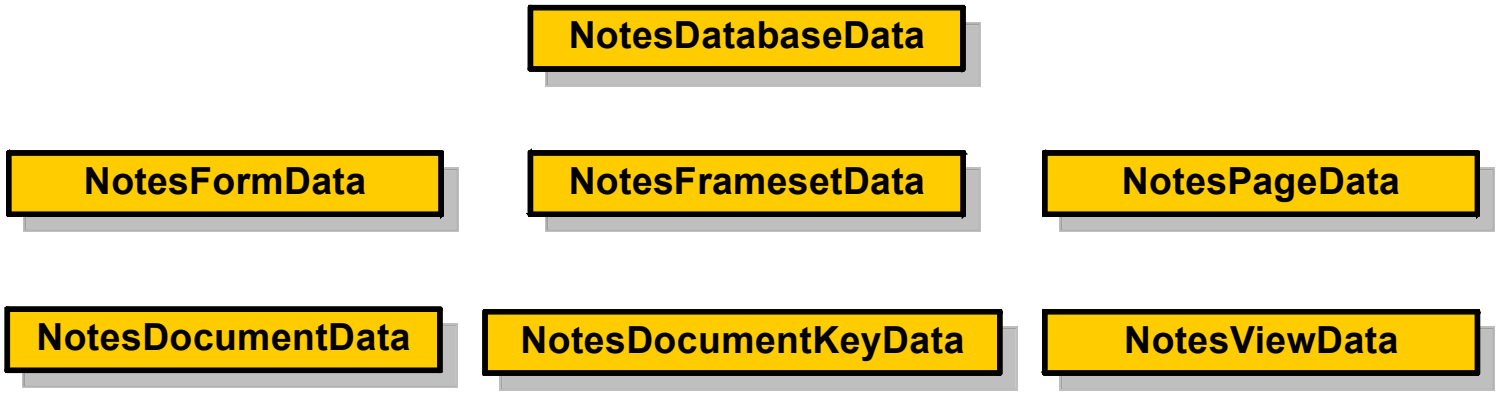
- Ein neues Dokument mit vorbesetzten Feldern erstellen

```
NotesUIWorkspace ws = new NotesUIWorkspace();
NotesDatabaseData dbData =
   new NotesDatabaseData("Server/Org", "main/jdoe.nsf");
NotesFormData formData =
   new NotesFormData(dbData, "Memo");
formData.addComposeItem("SendTo", "Peter Smith/Org");

ws.composeDocument(formData);
```


Tiefer einsteigen - NotesUIWorkspace erkunden

- Daten-Klassen werden in der API genutzt, um Daten zwischen Notes-Sessions im Speicher zu halten
- Daten können sicher zwischen API-Aufrufen übergeben und lokal gesichert werden
- In einigen Fällen fehlen einige Daten-Werte (z.B. der Pfad zu einer Datenbank)
  - Ist eine technische Beschränkung der API
  - Es können dann „open“-Methoden genutzt werden, um diese Daten zu laden
  - z.B. NotesDatabaseData.open(Session)



Tiefer einsteigen - NotesUIWorkspace erkunden

- Datenbank zum Workspace hinzufügen

```
NotesUIWorkspace ws = new NotesUIWorkspace();
NotesDatabaseData dbData =
    new NotesDatabaseData("Server/Org", "path/db.nsf");
ws.addDatabase(dbData);
```

- Wenn eine Datenbank bereits auf dem Workspace existiert, wird Ihre Kachel fokussiert

Demo

- Erstellung von Dokumenten mit vorbesetzten Feldern
- Bookmark/Workspace Add-on

Tiefer einsteigen - Agent in der UI aufrufen

- Bisher konnten LotusScript-Agenten nur im Backend ausgeführt werden
  - Keine Möglichkeit, UI-Änderungen von einem Agenten aus durchzuführen
- Neuheit in der API in 8.5.1
  - Neue Funktion, um einen Agent in der Notes UI auszuführen
  - z.B. um auf das aktuelle Dokument/die aktuelle Ansicht zuzugreifen und Dialoge anzuzeigen
  - Man kann sogar Agenten anderer Datenbanken ausführen!
- Nützlich für Funktionen, die bisher nicht Teil der API sind
  - Code in einen Lotusscript-Agenten schreiben
  - NotesUIWorkspace.runAgent aufrufen
  - Callback-Listener nutzen, um bei Abschluss des Agenten benachrichtigt zu werden
  - Daten-Austausch zwischen Eclipse und LotusScript

Verbesserte Eclipse-Selektion

- In 8.5: Eclipse-Selektion begrenzt auf bereits gespeicherte Dokumente aus Ansichten
  - Es wurden hauptsächlich Notes-URLs übergeben
  - Kein Zugriff auf "in-memory"-Dokumente
- In 8.5.1: Informationen auch über noch nicht gespeicherte Dokumente und sogar fokussierte Felder eines Formulars
  - Es kann verfolgt werden, was der Nutzer editiert
  - Einführung von Klassen, die mehr Informationen liefern als nur die Notes-URL (z.B. NotesUIElement, NotesUIDocument und NotesUIField)
  - Nutzt standard Eclipse-Konzept (Adapter), um zusätzliche Daten zu liefern!

Verbesserte Eclipse-Selektion

- Verwendung von IAdaptable auf Selektionen

```
// iterate over a selection and print the form-name
for(Iterator<?> i=((IStructuredSelection)
  selection).iterator(); i.hasNext(); ) {
   Object item=i.next();
   if (item instanceof IAdaptable) {
      NotesUIDocument doc=(NotesUIDocument)((IAdaptable)
        item).getAdapter(NotesUIDocument.class);

      if (doc!=null)
         System.out.println(doc.getForm());
   }
}
```


Demo

- Universelles kontext-basiertes Online Hilfe-System
- Aufruf von Agenten von Eclipse und Transfer von Daten

Agenda

- Einführung
- Lotus Notes mit Java erweitern
- Mehrwert für das klassische Lotus Notes
- Domino Designer mit Java erweitern
- Best Practices
- Q&A

Grenzen der API

- Klingt toll! Und wo ist der Haken?
  - Wir wollen nicht nur zeigen, was möglich ist, sondern auch was nicht möglich ist
- Nur ein kleiner Teil der LotusScript API
  - Neue Features werden mit künftigen Releases integriert
- Event Listener blockieren nicht bei Form/View-Events
  - Kein Ersatz für LotusScript Events, z.B. QuerySave
- Richtung nur Eclipse => klassisches Notes
  - Hilft nur der Eclipse-Welt, keine Verbesserungen für die Welt des klassischen Clients

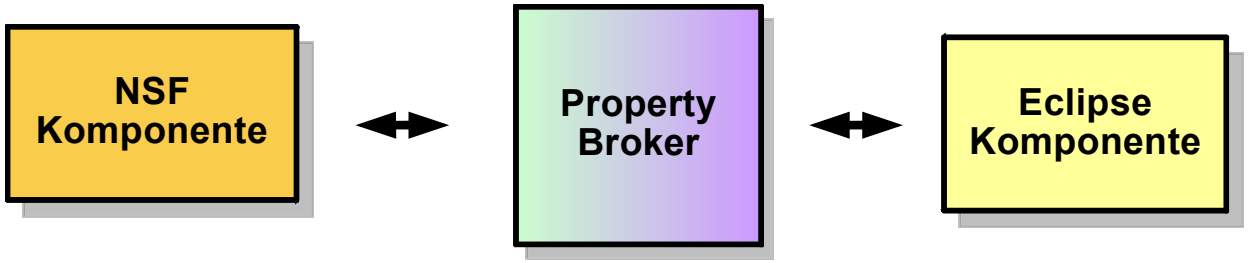
Was können wir tun, um beide Welten enger zusammen zu bringen?

- Wie können wir Eclipse-Funktionen von LotusScript aus nutzen?
- Drei mögliche Lösungen, um Eclipse vom klassischen Notes Client aus anzusprechen

Verbindung beider Welten

Lösung 1: Der Property Broker

- Einen String von LotusScript zum Property Broker senden
- Eclipse-Plugin ist beim Broker registriert und empfängt den String
- Asynchrone Lösung
  - Kein direkte Antwort, nur mittels Composite Application Wire
- Arbeitet lediglich in einer Composite Application



Verbindung beider Welten

Lösung 2: Erstelle eigenen URL Handler

- Undokumentiertes / nicht supportetes Feature seit 8.5
- Eigenen URL-Typ registrieren z.B. „myprotocol://”
- Java Handler wird aufgerufen, wenn eine URL geöffnet werden soll
  - Funktioniert für NotesUIWorkspace.URLOpen, @UrlOpen, Links im Richtext und der Adress-Zeile
  - Interessant auch für andere Einsatzzwecke
- Länge der URL ist begrenzt
  - Mehrfach aufrufen für größere Datenmengen
- Asynchrone Lösung
  - Arbeitet jedoch außerhalb von Composite Application

Demo

- Eigener URL Handler in Lotus Notes

Verbindung beider Welten

Lösung 3: Eigene Brücke erstellen!

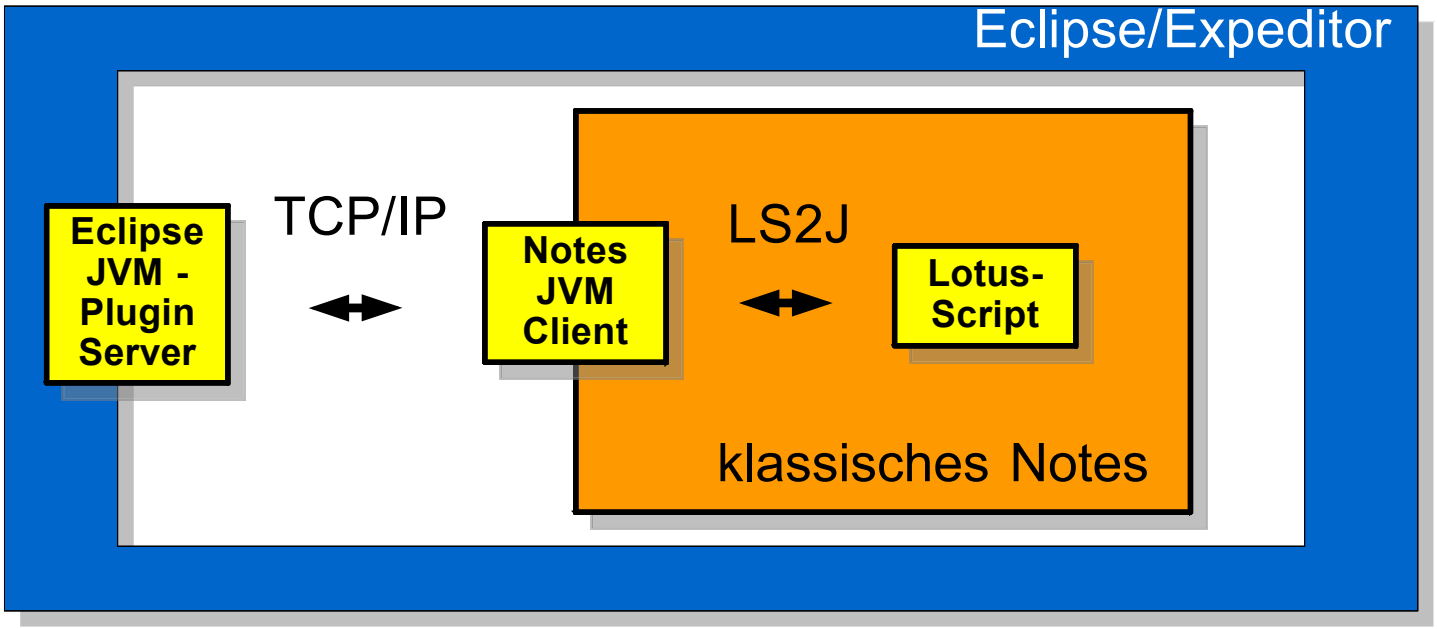
Idee:

- Zwei JVMs: Eclipse JVM und klassische Notes JVM
  - Keine direkte Verbindung zwischen Eclipse-Plugins und Notes-Agenten
- Benutze lokale Netzwerk-Kommunikation zwischen beiden
  - Öffne einen Server-Port in Eclipse, verbinde darauf von der klassischen JVM
- Benutze eigenes Protokoll oder Industrie-Standard wie z.B. Java RMI
  - Und rufe remote Eclipse-Plugin-Code vom klassischen Notes auf
- Optional: Nutze LS2J für den Zugriff auf die Remote-API in LotusScript

Verbindung beider Welten

Lösung 3: Eigene Brücke erstellen!

- Das Resultat ist beeindruckend
  - Kombination von LotusScript und Eclipse-Plugins eröffnet neue Entwurfs-Muster
  - Z.B. Hintergrund-Threads für lang andauernden LotusScript-Code oder Erzeugung von Eclipse-Reitern und Layouts von LotusScript on-the-fly



Demo

- Multi-Threading in LotusScript-Anwendung
- LotusScript-Zugriff auf die Eclipse-Workbench
- LotusScript erstellt neue Eclipse-Reiter-Layouts on-the-fly (Eclipse Perspective)

Agenda

- Einführung
- Lotus Notes mit Java erweitern
- Mehrwert für das klassische Lotus Notes
- Domino Designer mit Java erweitern
- Best Practices
- Q&A

Domino Designer Extensibility API

- Die DDE API ermöglicht Programmierung von Java Erweiterungen der Domino Designer IDE
  - Haupteinsatzzweck:
  - Reagieren auf Nutzer-Selektion -
    z.B. Anzeige zusätzlicher Daten in eigenen Ansichts-Bereichen (Eclipse Views)
  - Automatische Verarbeitung von Daten anbieten, z.B. Flags für alle selektierten Bilder setzen oder Code-Generatoren, die Design-Elemente erzeugen
- Einführung neuer Funktionalität
  - Design-Element Informationen der aktuellen Eclipse-Selektion
  - Basis-Daten von Design-Elementen oder Datenbanken setzen
  - Aktualisieren von Projekten/einzelnen Design-Elementen nach einer Änderung im Hintergrund (z.B. durch einen DXL-Import)
  - Datenbanken im DDE-Navigator öffnen
- Zusätzliche Erweiterbarkeit gewinnen durch Nutzung von Standard Eclipse APIs
  - Ein NSF-Projekt ist nur eine Erweiterung von Eclipse IProject

Eclipse-Selektion in DDE API Objekten umwandeln

- Konvertiert ein Eclipse IProject in ein DesignerProject
  - Ein IProject ist ein generisches Enwicklungs-Projekt in der Eclipse IDE

```
DesignerProject nsfProject =
   DesignerResource.getDesignerProject(iproject);
String dbServer = nsfProject.getServerName();
String dbPath = nsfProject.getDatabaseName();

//
//modify db design here, then notify DDE about changes
//
nsfProject.refresh();
```


Eclipse-Selektion in DDE API Objekten umwandeln

- Konvertiert eine Eclipse IResource in ein DesignerDesignElement
  - Eine IResource ist ein generisches Unterelement eines Eclipse IProjects

```
DesignerDesignElement de =
    DesignerResource.getDesignElement(iresource);
String oldName = de.getName();

//
//modify design element here, then notify DDE about changes
//
de.refresh();
```


Demo

- Eigene Eigenschaften für Notes Design-Elemente
- LotusScript.doc* Integration in Domino Designer
- Design-Manipulationen

*) Download von Lotusscript.doc V2 erhältlich unter: http://blog.lsdoc.org

Agenda

- Einführung
- Lotus Notes mit Java erweitern
- Mehrwert für das klassische Lotus Notes
- Domino Designer mit Java erweitern
- Best Practices
- Q&A

Generelle Empfehlungen

- Das Arbeiten mit Threads erlernen
  - Keine lang andauernden Operationen im UI-Thread ablaufen lassen!
  - Dies blockiert den gesamten Client!
  - Berechnungen in Hintergrund-Jobs auslagern → dann einen UIJob zum Update der UI nutzen

```
NotesSessionJob job = new NotesSessionJob("BG Operation") {
  protected IStatus runInNotesThread(
      Session session, IProgressMonitor monitor)
        throws NotesException {
      //compute something here
      return Status.OK_STATUS;
  }
};
job.schedule();
```


Generelle Empfehlungen

- Keine Notes-Objekte cachen
  - Kann zu schweren Problemen mit dem Memory-Management führen
- Die Notes Java API hat nur eine begrenzte Anzahl von Handles für Daten-Objekte
  - Und der eigene Code ist nicht alleine im Client!
  - Wenn möglich stets .recycle() aufrufen
- NotesSessionJob für den Notes-Zugriff nutzen
  - Wird im Hintergrund ausgeführt
  - Holt immer eine neue Session; ist sogar dann sicher, wenn die Notes ID geändert wurde
  - Recycled automatisch alle Notes-Objekte, die innerhalb der Session erzeugt wurden
    → Kopieren der Notes-Daten in eigene Objekte zur Zwischenspeicherung
  - UI API Daten-Klassen sind hingegen sicher (z.B. NotesDocumentData)

Zusammenfassung

- Eclipse-Entwickler erhalten zusätzliche Möglichkeiten, um mit der Lotus Notes UI interagieren zu können
  - Existierender LotusScript-Code kann wiederverwendet werden (Aufruf in UI-Agenten)
  - Kann für einen sanften Übergang von Notes-Code zu Java-Plugins genutzt werden
  - Raum für Verbesserungen in den APIs: Fehlende Klassen/Methoden gegenüber LotusScript
  - IBM-Entwicklung sollte auch weiter an den Backend APIs arbeiten (Notes.jar): z.B. Parameter-Übergabe an Backend-Agenten, Streaming von Anhängen und andere Features
- Klassische Notes-Entwicklung kann auch von den APIs profitieren
  - Durch eine Brücke zwischen LotusScript und Eclipse-Plugins können API-Funktionen auch von klassischen Formularen und Ansichten genutzt werden
  - Interessante neue Entwurfs-Muster wie Multi-Threading in LotusScript-Anwendungen und Verwendung von Eclipse UI-Komponenten (Perspective, View)
- DDE kann jetzt ebenso erweitert werden
  - Selektion von Design-Elementen nutzen, um Design zu modifizieren und z.B. Code/Design-Generatoren zum DDE hinzuzufügen
- Fazit: Lotus Notes ist cool :-)

Vielen Dank!

Zeit für Fragen