Lotusphere 2010

Lotus knows.

Smarter software for a Smarter Planet.

BP203
Leverage the New Java APIs in IBM® Lotus Notes® 8.5.1!

Karsten Lehmann, Tammo Riedinger | CEOs Mindoo GmbH

lotusknows.com

Agenda

- Introduction
- Extending Lotus Notes in Java
- Added value for classic Lotus Notes
- Extending Domino Designer in Java
- Best practices
- Q&A

About us

- Mindoo is IBM Business Partner and Notes/Domino Design Partner
- Focused on the „new“ IBM Lotus Notes development areas
  - Eclipse/Expeditor plugins and rich client applications
  - Composite Application architectures
  - LiveText extensions
  - Xpages applications
- Karsten Lehmann and Tammo Riedinger
  - Founders of Mindoo
  - Since 2004 developers of the MindPlan® application,
    Mindmapping and Project Management for Lotus Notes, IBM Award Winner 2008
- More company information:
  http://www.mindoo.com

[mindu:]
Mindoo GmbH

Motivation

- „Notes can do that too“ - yes, but why must it be so complicated?
  - Until now, Eclipse developers had to use far too many workarounds to interact with the classic Notes UI
  - Think of “using Notes.ini for data exchange” or “opening pages that close themselves, just to run some LotusScript® code in the QueryClose event”
  - The purpose of the new APIs is to make this gap a little smaller and life quite a bit easier
- We participated in the discussion about required features for the new Notes and Designer extensibility APIs
  - Discussion with IBM dev about API draft at Lotusphere 2009 and conference calls
  - Design feedback / test reports within the Design Partner program
- As always for a first release, there is still room for improvements
  - But we think this is already a **huge step** forward!

Disclaimer

- The presented Java APIs do not work in Notes agents. This is pure Eclipse stuff!

- Don't leave yet, if you do pure LotusScript development. We also have something for you later on!

Disclaimer

- We cannot cover all APIs in detail in this session!
  - We'll focus on some "**hot**" areas instead:
    namely the new Java UI APIs and the Designer Java-API
  - See BP209 and AD103 for two one hour sessions about each API
- We hope to give you a good impression, what you can really do with the Notes 8.5.1 APIs.
  - Hopefully you will leave this session with some ideas of your own already
  - We have created up to 10 demos for you!
- We try not to bore you to death with code today!
- Watch our blog for an upcoming series with details about how our demos work instead:
  http://blog.mindoo.com

Getting started

- The following APIs and demos are based on
  - Eclipse 3.4.2
  - Expeditor toolkit 6.2.1
  - IBM Lotus Notes 8.5.1
- Install Expeditor into Eclipse and set Lotus Notes 8.5.1 as target platform

- Create a plugin-in project to develop your code
- For the UI API, add the following dependency
  - com.ibm.notes.java.ui
- For the DDE API, add two dependencies:
  - com.ibm.designer.domino.ui.commons
  - com.ibm.designer.domino.ide.resources

Agenda

- Introduction
- Extending Lotus Notes in Java
- Added value for classic Lotus Notes
- Extending Domino Designer in Java
- Best practices
- Q&A

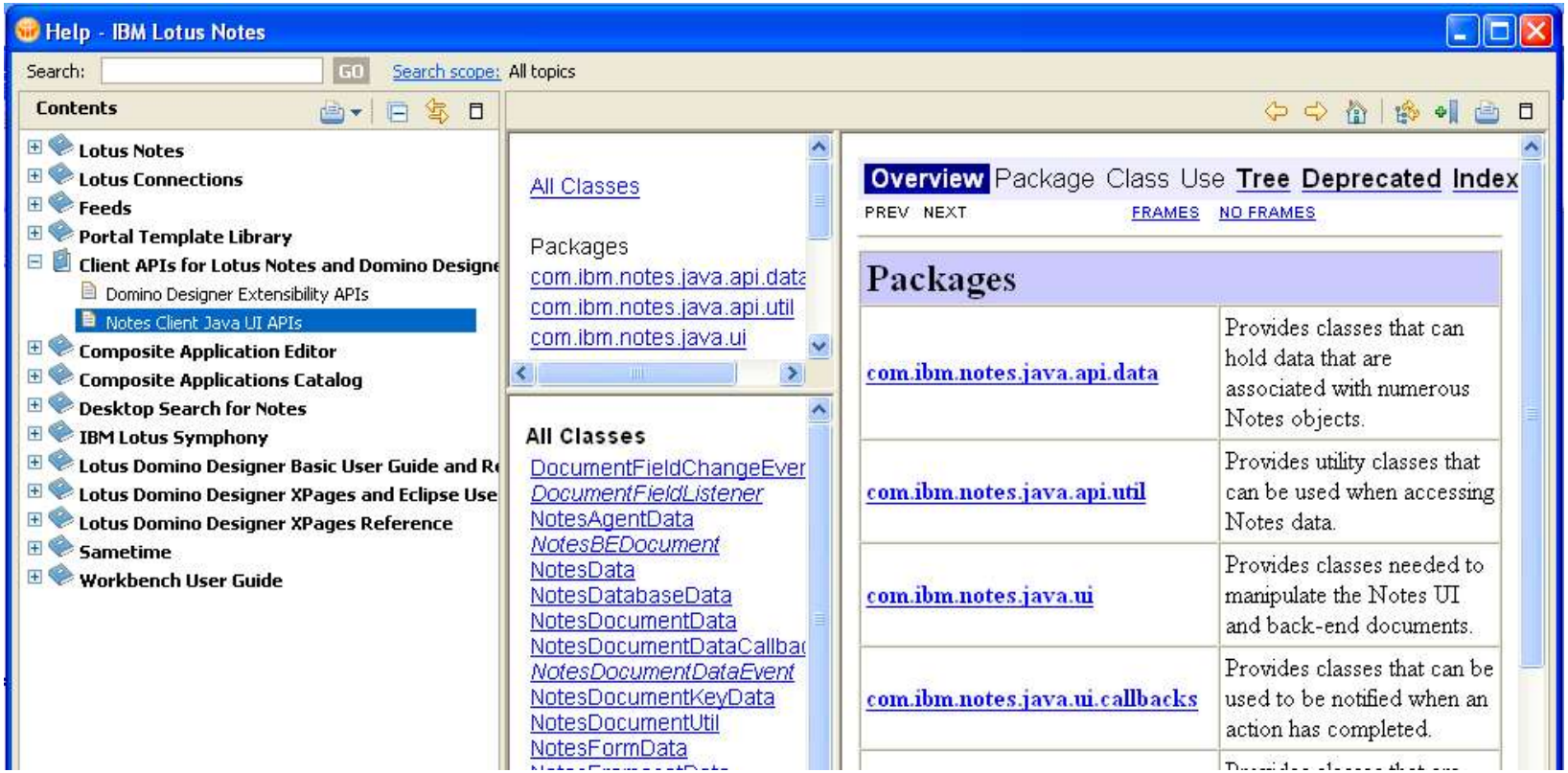
The Java UI API – a Management Summary

- API new in 8.5.1
  - Unsupported preview available in 8.5
- Further improves the integration between Eclipse and the classic Lotus Notes client
  - Eclipse plugins can finally receive information about the state of forms and views of the classic client
  - New ways of interaction between these two worlds
- Makes existing functionality easier to use
  - Opening of design elements
  - Printing documents and views from Eclipse
  - Compose a new document and fill it with default items
  - Getting a temporary document to store and pass data
  - Execute Notes code in a background thread with proper memory management (NotesSessionJob)

The Java UI API – a Management Summary

- Adds new functionality and further closes the gap between Eclipse and Lotus Notes
  - Read/modify contents of documents in edit mode
  - Document listener (detect edit mode on/off, modifications and document closing)
  - Get Eclipse selection for focused fields and new unsaved documents
  - Add database to workspace
  - Prompt methods of NotesUIWorkspace, e.g. to choose a database
  - Lots of Eclipse property testers (a kind of "hide when" for Eclipse elements like e.g. actions)
  - Execute LotusScript agents in the UI (they can display dialogs), pass data back and forth and attach callbacks

Where is the documentation?



Start small – Your first example

- Toolbar action that changes fields in an open form
- Add this to a standard Eclipse toolbar action:

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

Start small – Your first example

- For backend document fields:

```
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();
}
```

- Due to technical restrictions no full access to the document from Java

Demo

- Access and modify the UI document with Eclipse actions

Digging deeper - Exploring NotesUIWorkspace

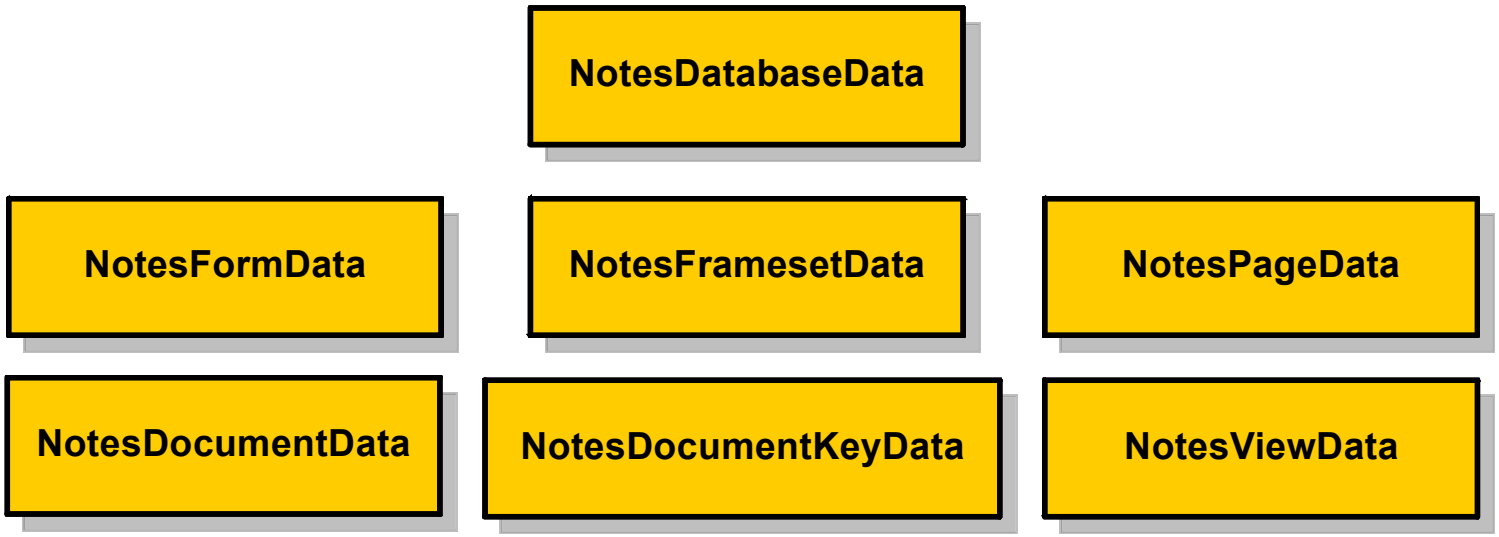
- Composing new documents with preset fields

```
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);
```

Digging deeper - Exploring NotesUIWorkspace

- Data classes are used in the API to store data between Notes session
- You can safely pass them between calls to the API and store them locally
- sometimes, data (like the database's filepath) is missing
  - This is due to technical restrictions
  - Use the open-method to let Notes fill in the missing fields
  - e.g. NotesDatabaseData.open(Session)



Digging deeper - Exploring NotesUIWorkspace

- Add databases to the workspace

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

- If the database is already on the workspace, it gets focused

Demo

- Compose document demo

Digging deeper - Calling agents in the UI

- Until now, you could only execute LotusScript agents in the backend
  - No way to change the Notes UI from an agent
- This has changed in 8.5.1
  - New function to execute an agent in the Notes UI
  - E.g. to access the current document/view and display dialogs
  - Even run agents from a different database!
- Handy for functions that are not yet part of the UI API
  - Put your code in a Lotusscript agent
  - Call NotesUIWorkspace.runAgent
  - Use a callback listener to get notified when the agent is done
  - Pass data between Eclipse and LotusScript

Improved Eclipse selection

- In 8.5: Eclipse selection limited to selected documents in a view and already saved documents
  - Based mainly on passing around Notes-URLs
  - No access to "in-memory" documents
- In 8.5.1: get information about unsaved documents and even about focused fields
  - You can track what the user is editing
  - Introduction of classes that can be queried directly for more information than just the URL (e.g. NotesUIElement, NotesUIDocument and NotesUIField)
  - Uses standard Eclipse concept (adapters) to provide additional data!

Improved Eclipse selection

- Using IAdaptable on selections

```
// 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

- Universal context-based online help system
- Calling agents from Eclipse and transfer data

Agenda

- Introduction
- Extending Lotus Notes in Java
- Added value for classic Lotus Notes
- Extending Domino Designer in Java
- Best practices
- Q&A

Limitations of the API

- Sounds great! But where is the catch?
  - We do not only want to show what's possible, but also what's not possible
- Only a small subset of the LotusScript API
  - New features will be added in future releases
- Event listeners not blocking form/view events
  - No replacement for LotusScript events, e.g. QuerySave
- Eclipse => classic Notes only
  - Does only empower the Eclipse world, no improvements for the classic client world

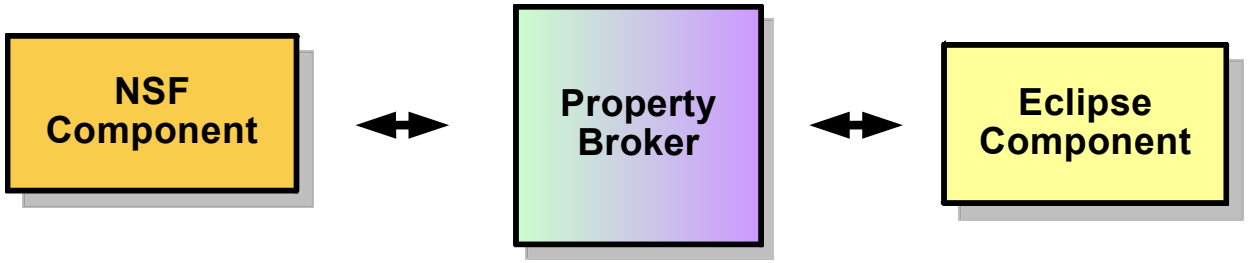
So how can we bring the classic client and Eclipse closer together?

- How can we leverage the Eclipse functions from LotusScript?
- Three possible solutions to access Eclipse from the Notes client

How to connect two worlds

Solution 1: The Property Broker

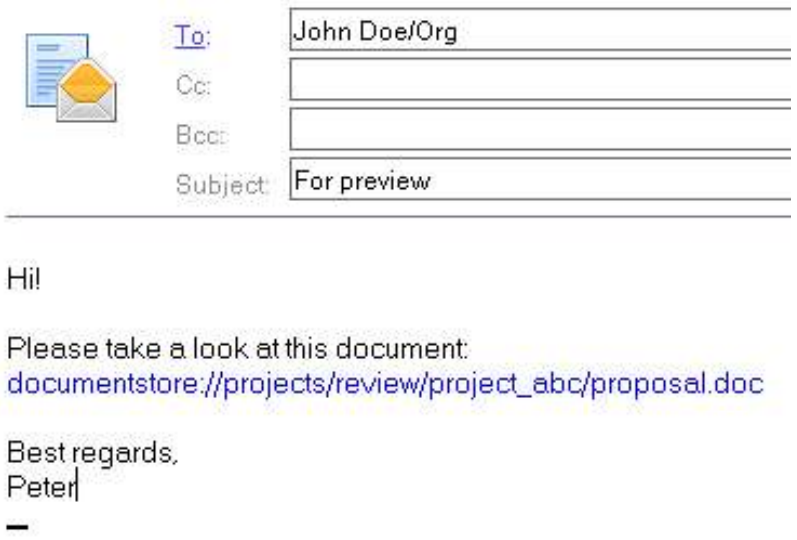
- Send a string from LotusScript to the Property Broker
- Eclipse plugin is registered at the Broker and receives the string
- Asynchronous solution
  - No direct anwer, only via Composite Application wire
- Does only work within a Composite Application



How to connect two worlds

Solution 2: Create your own URL handler

- Undocumented / unsupported feature since 8.5
- Register your own URL type myprotocol://
- Java handler is called whenever a URL should be opened
  - Works for NotesUIWorkspace.URLOpen, @UrlOpen, links in richtext and the address bar
  - Also interesting for other use cases
- Length of URL is limited
  - Call multiple times for more data
- Asynchronous solution
  - But works outside a Composite Application



Demo

- Custom URL handler in Lotus Notes

How to connect two worlds

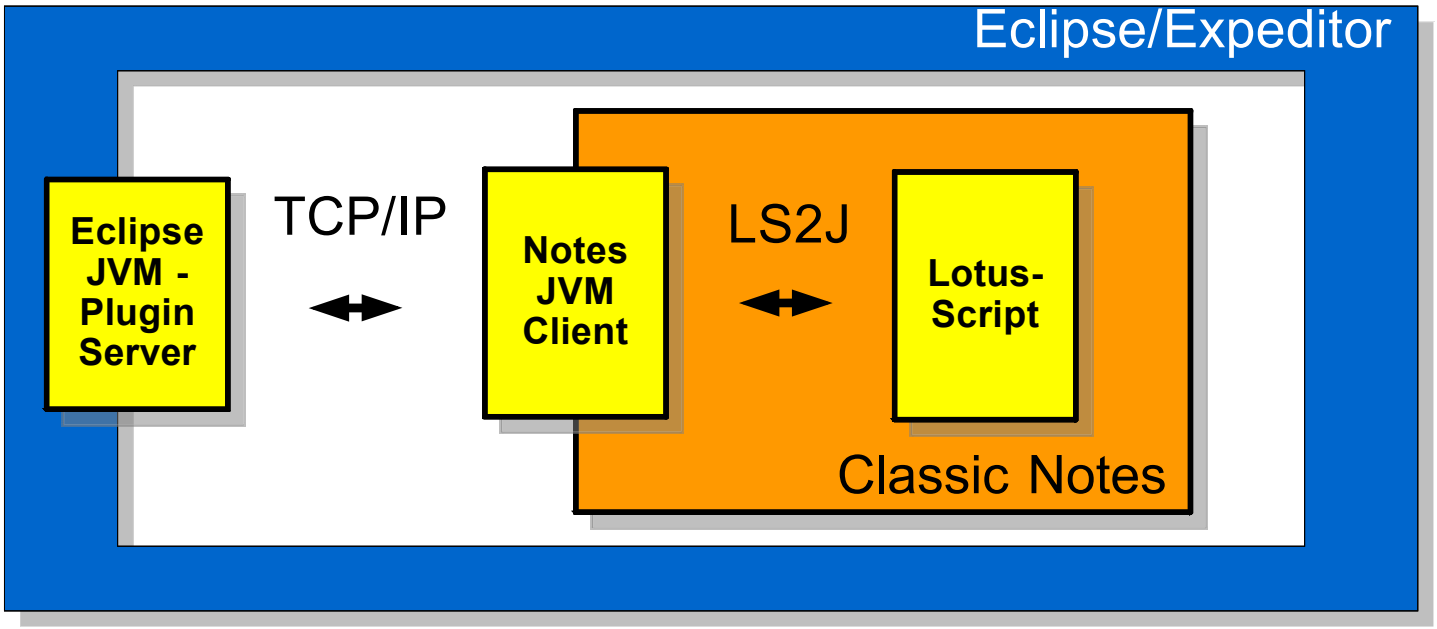
Solution 3: Build your own bridge!

- Two JVMs: Eclipse JVM and classic Notes JVM
  - No direct connection between Eclipse plugins and Notes agents
- Use local network communication between them
  - Open a server port in Eclipse, connect from the classic JVM
- Use your own protocol or industry standards like Java RMI
  - And remotely call Eclipse plugin code from classic Notes
- Optional: Use LS2J to use the remote Java API in LotusScript

How to connect two worlds

Solution 3: Build your own bridge!

- The result is pretty impressive
  - Combination of LotusScript and Eclipse plugins opens up new design patterns
  - E.g. background threads for long-running LotusScript code or creating Eclipse tabs and layouts from LotusScript on the fly



Demo

- Multithreaded LotusScript application
- LotusScript creating new Eclipse tab layouts on the fly (Eclipse Perspectives)

Agenda

- Introduction
- Extending Lotus Notes in Java
- Added value for classic Lotus Notes
- Extending Domino Designer in Java
- Best practises
- Q&A

Domino Designer Extensibility API

- The DDE API allows for programmatic Java extensions of the Domino Designer IDE
  - Main use cases:
  - React on the user selection -
    e.g. display additional data in your own display areas (Eclipse views)
  - Offer automated processing of data, e.g. set flags for all selected images
    or let code generators create the design
- Adds new functionality
  - Get design element information about the current Eclipse selection
  - Set basic design element and database data
  - Refresh the project/single design element after a backend change (e.g. DXL import)
  - Open databases in the DDE navigator
- Additional extensibility gained by leveraging the standard Eclipse APIs
  - An NSF project is an extended Eclipse IProject

Convert Eclipse selection into DDE API objects

- Convert Eclipse IProject into DesignerProject
  - An IProject is a generic development project in the 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();
```

Convert Eclipse selection into DDE API objects

- Convert Eclipse IResource into DesignerDesignElement
  - An IResource is a generic subelement of an Eclipse IProject

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

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

Demo

- Custom properties for Notes design elements
- LotusScript.doc* integration into Domino Designer

*) Download of Lotusscript.doc V2 available at: http://blog.lsdoc.org

Agenda

- Introduction
- Extending Lotus Notes in Java
- Added value for classic Lotus Notes
- Extending Domino Designer in Java
- Best practices
- Q&A

General advice

- Learn to work with threads
  - Don't do long running operations in the UI thread!
  - This blocks the whole client!
  - Do calculations in background jobs, then use a UIJob to update the UI:

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

General advice

- Don't cache Notes objects
  - Can lead to severe memory issues
- The Notes Java API only has a limited amount of handles for data objects
  - And you are not alone in the client
  - Call .recycle() whenever possible
- Use NotesSessionJob for your Notes access
  - Executes in the background
  - Grabs a fresh session every time, safe even if the Notes ID has changed
  - Automatically recycles all the Notes objects created within the session
  - Copy the Notes data into your own objects
  - UI API data classes are safe (e.g. NotesDocumentData)

Summary

- Eclipse developers get additional ways to interact with Lotus Notes UI
  - Existing Lotusscript code can be reused by calling it in UI agents
  - Can be used for a smooth transition of Notes code to Java plugins
  - Room for improvements in the APIs: Missing classes/methods compared to Lotusscript
  - IBM dev should also work on the backend APIs (Notes.jar):
    e.g. improve parameter passing to backend agents, attachment streaming and other features
- Classic Lotus Notes development can also benefit from the new APIs
  - By building a bridge between LotusScript and Eclipse plugins, the API functions can also be used from classic forms and views
  - Interesting new design patterns like multithreaded LotusScript applications and Eclipse UI control
- DDE can now be extended as well
  - Leverage the design element selection to modify design, add code/design generators to DDE
- Conclusion: Notes is cool :-)

Thank you!

Time for Q&A

Legal Disclaimer

The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

IBM, the IBM logo, Lotus, Lotus Notes, Notes, Domino and Lotusphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.