

LotuspHERE[®] 2009



CREATED WITH LOTUS[®] SYMPHONY™

BP106

From IBM Lotus[®] Notes[®] Java[™] Developer to Lotus[®] Notes[®] 8 Plug-in Developer

Mikkel Heisterberg / Senior Solution Architect / Intravision

Lotusphere[®] 2009



Who I am

- With IntraVision (IBM premier business partner out of Denmark) – we're in the showcase
- 32 years old
- Live in wonderful, wonderful Copenhagen (DK)
- Domino NEXT design partner / writer for THE VIEW / working with Notes since v. 3.x
- LotusScript.doc / TwitNotes sidebar plug-in
- Strong proponent of Java and OOP
- Blog: lekkimworld.com / Twitter: [lekkim](https://twitter.com/lekkim)



Who you are

- How many with Java experience? (hope for all of you)
- How many have developed in Eclipse? (hope for most of you)
- How many know what a plug-in is? (hope for many of you)
- How many have developed for Eclipse? (hope for some of you)
- How many have written and deployed a plug-in?
- How many would like to develop a plug-in?
- How many are here to sleep/relax/tweet/blog? - I usually hate that! :-)

I assume...

- You are proficient in Java or at least can tell the difference between a class, an abstract class and an interface
 - ▶ Knowing anonymous inner classes would be good
- You have working knowledge of the Notes/Domino Java API
- You know your way around Eclipse
- You are running Notes 8.0.x/8.5 Standard
- You are ready to extend Notes 8 beyond belief!

Agenda

- Developing Java in Notes 5 -> 7
- The framework basics
 - ▶ Eclipse / OSGi / Lotus Expeditor[®], anatomy of a plug-in
 - ▶ SWT / JFace
 - ▶ Extension points
 - ▶ Threading / UI interaction
- Notes specifics
 - ▶ Data access
 - ▶ Extension points
- Demos (as many as time permits)
- Q & A

Java in Notes 5 -> 7

- Not much to talk about...
 - ▶ Issues with Java version
 - ▶ Security manager issues
- With Notes 8 we're on Java 5 though the Notes/Domino API is still lagging behind
- JavaCompilerTarget notes.ini setting
 - ▶ JavaCompilerTarget=1.5
 - ▶ See lekkimworld.com for more info on JavaCompilerTarget

Be aware

- Notes 8 plug-in development != Notes development
- Plug-in development is Java development and should be approached as such
- Watch out for threading issues
- Java and Eclipse skills are needed (some XML as well)
- Once you learn the ropes you're all set to exploit the platform

Getting started – 1, 2, 3, ready!

Two options for getting your Eclipse IDE ready for Notes

- 1 Configure Eclipse manually to work with Notes (see lekkimworld.com)
- 2 Use the Lotus Expeditor Toolkit (highly recommended)

Using the Lotus Expeditor Toolkit

- 1 Download Lotus Expeditor Toolkit
- 2 Download Eclipse
- 3 Follow the instructions on installing Expeditor toolkit into Eclipse

Follow the instructions in the toolkit and in the IBM InfoCenter

Eclipse Math

Eclipse = a generic **workbench** + IDE plug-ins

Lotus Expeditor = Eclipse 3.2

- the IDE specific plug-ins
- + a bunch of IBM Lotus plug-ins

Notes 8 = Lotus Expeditor + even more plug-ins

- some other plug-ins
- + Notes plug-ins with native code

* Notes 8.5 is based on Lotus Expeditor 6.2 which is based on a newer Eclipse

Using set theory



Even more Eclipse math

Eclipse = a generic workbench + IDE plug-ins

Workbench = perspective = views + editors

Eclipse = Java + UI layer

UI layer = SWT + JFace

SWT = Standard Widget Toolkit
= Java UI layer reusing of native widgets

JFace = coarse grained components built on SWT
(e.g. Viewers)

Plug-ins 101 – what's a plug-in?

plug-in = code + declarations
= Java + XML + OSGi manifest
= Java + plugin.xml + manifest.mf

Plug-in facts

- A plug-in is
 - ▶ a piece of managed code that plugs into stuff
 - ▶ a piece of managed code that may have other plug-ins plug into it (think of an electric socket and plugs)
- A plug-in does not
 - ▶ have to contain Java code
 - ▶ have to have a UI – you can write plug-ins that are never seen by the end user

Plug-in anatomy

- plugin.xml
 - ▶ Defines what you extend ("the plugs") and optionally new extension points for others to extend ("the sockets")
- META-INF\MANIFEST.MF
 - ▶ Exported packages
 - ▶ Required packages / required bundles
 - ▶ ID, version etc.
- Your stuff
 - ▶ Code (not always required)
 - ▶ Images, resources, property files

OSGi

- From Eclipse 3.1 OSGi became a part of Eclipse
 - ▶ Bundles are to OSGi what plug-ins are to Eclipse
 - ▶ Think of the two terms as being synonyms – the terms are used interchangeably
- OSGi is the framework for managing the plug-ins, their state etc.
 - ▶ INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING, UNINSTALLED
- Most important is the OSGi console which runs underneath Eclipse including Lotus Expeditor and Lotus Notes
 - ▶ Try starting your Notes client using the -console command line switch

```
<Notes dir.>/framework/rcp/rcplauncher.exe -config notes -console
```

A bit on widgets

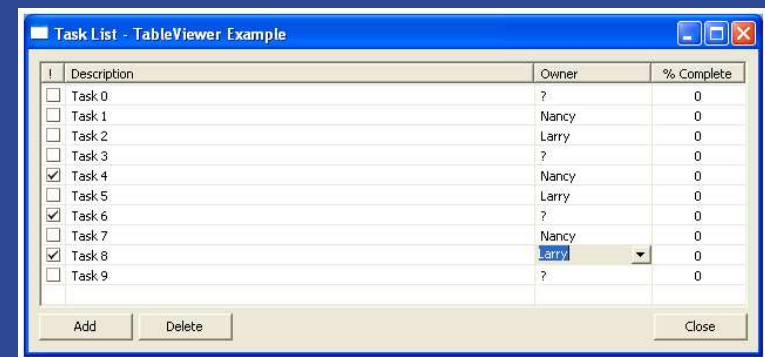
- The SWT library holds the low level widgets
 - ▶ Label, Button, Text etc.
 - ▶ org.eclipse.swt.widgets package
 - ▶ Beware of AWT look alikes! (use filters in Eclipse to not display type-ahead for AWT-classes)
- Remember that the SWT widget classes are only wrappers around the native OS widgets
 - ▶ Some acquire system resources and needs to be disposed e.g. Image - if you create it you dispose it!
 - DisposeListener to the rescue
 - ▶ Disposing a parent will dispose its children

JFace

- Coarse grained components from viewers to wizards
- I suggest you read up on JFace viewers (org.eclipse.jface.viewers package) as these can be used to display just about anything

- Parts

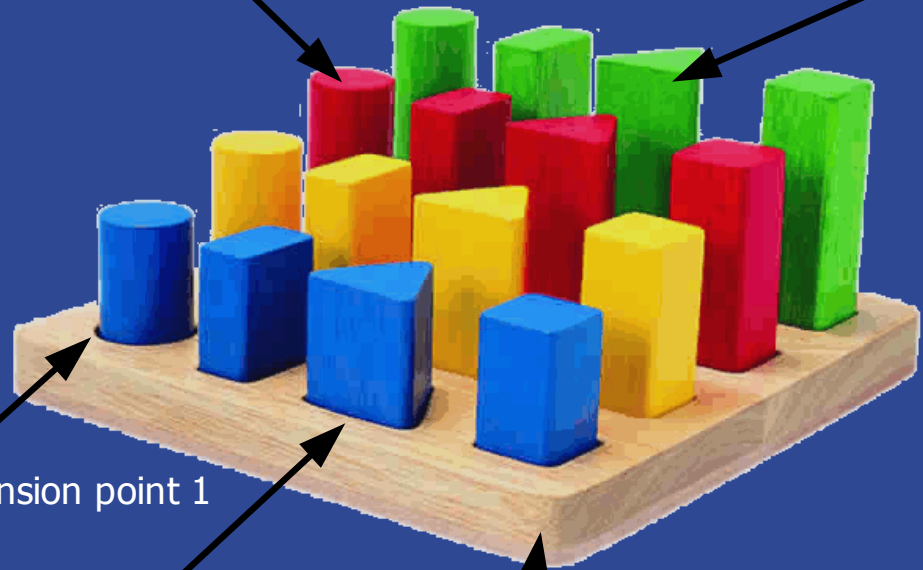
- ▶ Viewer
- ▶ ContentProvider
- ▶ LabelProvider
- ▶ Filter
- ▶ Sorter



Extension points – the analogy

A round peg = an extension to extension point 1

A triangular peg = an extension to extension point 2



The round holes = extension point 1

The triangular holes = extension point 2

The board = the platform / another plug-in



Notes extension points

- `com.ibm.rcp.ui.shelfViews`
 - ▶ Demo: `com.ls09.bp106.sidebar`
- `com.ibm.rcp.ui.launcherSet`
 - ▶ Demo: `com.ls09.bp106.perspective`
- `com.ibm.rcp.search.engines.searchEngines` / `com.ibm.rcp.search.ui.searchBarSets`
 - ▶ Demo: `com.ls09.bp106.search`
- `com.ibm.rcp.ui.controlSets`
 - ▶ Demo: `com.ls09.bp106.bars`
- ...
- ...

How many extensions do you see?

The screenshot shows the IBM Lotus Notes interface with several red circles highlighting specific elements:

- A circle around the "Push me! Drop down..." button in the top toolbar.
- A circle around the "Open Lotusphere 2009 Demo Perspective" and "Open Lotusphere 2009 Animation Perspecti..." items in the left sidebar.
- A circle around the "Lotusphere 2009: Image...", "Lotusphere 2009: Extension Points", "Lotusphere 2009: Preferences", "Lotusphere 2009: Memento", "Lotusphere 2009: Threading", "Lotusphere 2009: Drag'n'drop", "Lotusphere 2009: dnd2pdf", and "Lotusphere 2009: Monét" items in the main content area.
- A circle around the "Lotusphere 2009" icon in the bottom right corner.

A white box in the center contains the calculation: $2 + 1 + 8 + 2 + 1 = 14$.

Extend...

- Most of the Notes client, besides the core, is a collection of plug-ins
- You extend the client by using the extension points provided by other plug-ins in the client
- You use an extension point by adding some XML to the plugin.xml file specifying which extension point(s) you're plugging into
 - ▶ Alternatively use the GUI plugin.xml editor in Eclipse
 - ▶ Unfortunately not all extension points are well described so copy/pasting from examples is usually a good idea

Extend... (example plugin.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension point="com.ibm.rcp.ui.shelfViews">
    <shelfView
      id="com.ls09.bp106.viewpart"
      region="TOP"
      showTitle="true"
      view="com.ls09.bp106.viewpart">
    </shelfView>
  </extension>
  <extension point="org.eclipse.ui.views">
    <category name="Lotusphere 2009" id="com.ls09.bp106" />
    <view
      category="com.ontime.ls09"
      class="com.ls09.bp106.MainViewPart"
      id="com.ls09.bp106.viewpart"
      allowMultiple="true"
      name="Lotusphere 2009" />
  </extension>
</plugin>
```

Hook
into
the
sidebar

The
view
part

...and prepare to be extended

- You prepare to be extended by defining extension points yourself
- Extension points can be divided into two categories:
 - ▶ The simple ("static") is a set of XML declarations which is parsed and handled in full by the "hosting" plug-in that is you only define the schema
 - ▶ The advanced ("dynamic") is where you supply some XML declarations and one or more classes to provide the actual functionality
 - You define the schema and an interface for the functionality you want extending plug-ins to provide
 - Remember to export the package where you place the interface
- When defining an interface be sure to use `IConfigurationElement.createExecutableExtension(String)` to instantiate the class being contributed (class loading)
- Demo: `com.ls09.bp106.extension`

...and prepare to be extended (example plugin.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  { <extension-point id="dynamicContribution" name="Dynamic contribution extension point"
    schema="schema/dynamicContribution.exsd"/>
  { <extension-point id="staticContribution" name="Static contribution extension point"
    schema="schema/staticContribution.exsd"/>
  { <extension point="com.ls09.bp106.extension.staticContribution">
    <staticContribution
      displayValue="Some value 1">
    </staticContribution>
  </extension>
  { <extension point="com.ls09.bp106.extension.staticContribution">
    <staticContribution
      displayValue="Some value 2">
    </staticContribution>
  </extension>
  { <extension point="com.ls09.bp106.extension.dynamicContribution">
    <dynamicContribution
      class="com.ls09.bp106.extension.Ls09DynamicContribution"
      displayValue="Lotusphere 2009">
    </dynamicContribution>
  </extension>
  { <extension point="com.ls09.bp106.demo.extension.dynamicContribution">
    <dynamicContribution
      class="com.ls09.bp106.extension.Notes8RocksDynamicContribution"
      displayValue="Notes 8 rocks!">
    </dynamicContribution>
  </extension>
</plugin>
```

...and prepare to be extended (code, "static")

```
// declarations
IConfigurationElement[] configs = null;

// get extension registry
IExtensionRegistry reg = Platform.getExtensionRegistry();

// get extension points from registry
configs = reg.getConfigurationElementsFor
            ("com.ls09.bp196.extension.staticContribution");

// loop configuration elements (you may have multiple extensions)
for (IConfigurationElement config : configs) {
    // get static value
    String displayValue = config.getAttribute("displayValue");
}
```

...and prepare to be extended (code, "dynamic")

```
// declarations
IConfigurationElement[] configs = null;

// get extension registry
IExtensionRegistry reg = Platform.getExtensionRegistry();

// get extension points from registry
configs = reg.getConfigurationElementsFor
            ("com.ls09.bp106.extension.dynamicContribution");

// loop configuration elements (you might have multiple extensions)
for (IConfigurationElement config : configs) {
    // get static value
    String displayValue = config.getAttribute("displayValue");

    // instantiate contribution
    IExtension ext = (IExtension)config.createExecutableExtension("class");

    // do stuff with the extension code
    ext.doStuff();
}
```

Remember to use `IConfigurationElement.createExecutableExtension(String)` method to instantiate the class – no `Class.newInstance()` as it will fail!

Threading

- All code runs in threads
- There is **one** thread where all events from the UI are handled (the “UI thread”)
- All code that interacts with the UI including setting values of widgets (e.g. `Label.setText()`) must run in the UI thread
- All code that runs based on UI events (i.e. click of a button) automatically runs in the UI thread
- Don't use the “UI thread” unless necessary – consider doing the processing in the background and then bringing the result to the UI
 - ▶ This may mean “breaking” out of the UI thread, doing the processing and then using the “UI thread” to update the UI once the result is known
 - ▶ Tying up the “UI thread” will render the UI unresponsive i.e. users will complain of their client “locking up”
 - ▶ The same UI thread is responsible for your plug-in and refreshing the inbox!

Threading (code 1, do UI stuff)

Run code in the UI thread using Job API

```
new UIJob("Updating text control using UIJob") {  
    public IStatus runInUIThread(IProgressMonitor monitor) {  
        // manipulate UI  
        ...  
  
        // return status  
        return Status.OK_STATUS;  
    }  
}.schedule();
```

Run code in the UI thread, calling thread continues to run

```
Display.getCurrent().asyncExec(new Runnable() {  
    public void run() {  
        // manipulate UI  
        ...  
    }  
});
```

Run code in the UI thread, calling thread is blocked until UI code completes

```
Display.getCurrent().syncExec(new Runnable() {  
    public void run() {  
        // manipulate UI  
        ...  
    }  
});
```

Threading (code 2, wrong – how not to do it!)

```
public void doStuff(Display display, final Text txt) {
    new UIJob("Indicate job starting") {
        public IStatus runInUIThread(IProgressMonitor monitor) {
            // update label
            txt.setText("Starting to run code...");

            // do long running task
            try {
                Thread.sleep(10000);
            } catch (Exception e) {
                System.out.println("Ignored exception");
            }

            // update label
            txt.setText("Done running code...");
            return Status.OK_STATUS;
        }
    }.schedule();
}
```

Threading (code 3, nice and correct)

```
public void doStuff(Display display, final Text txt) {  
    new UIJob("Indicate job starting") {  
        public IStatus runInUIThread(IProgressMonitor monitor) {  
            txt.setText("Starting to run code...");  
            return Status.OK_STATUS;  
        }  
    }.schedule();  
    try {  
        Thread.sleep(10000);  
    } catch (Exception e) {  
        System.out.println("Ignored exception");  
    }  
    new UIJob("Update UI after long running job") {  
        public IStatus runInUIThread(IProgressMonitor monitor) {  
            txt.setText("Done running code...");  
            return Status.OK_STATUS;  
        }  
    }.schedule();  
}
```

Update UI using UIJob
to run in the UI thread

Simulate performing
long running task

Update UI using UIJob
to run in the UI thread

Notes 8 specifics



Notes data access

- The Notes thread is still there and has to be respected
- Access to Notes data and Notes objects has to be done from the Notes thread
- Two ways
 - ▶ Statically initiate a thread like you have done so far using `NotesThread.sinitThread`
 - ▶ Use the job API and run your code within a `NotesJob` (asynchronously)
- A Session object may be obtained in two ways
 - ▶ `NotesFactory.createSession()`
 - ▶ `NotesPlatform.getInstance().getSession()` <-- recommended

Notes data access (code)

```
try {
    NotesThread.sinitThread();
    Session session = NotesFactory.createSession();
    DbDirectory dbdir = session.getDbDirectory(null);
    Database db = dbdir.openMailDatabase();
    System.out.println("Title: " + db.getTitle());
} catch (NotesException e) {
    e.printStackTrace();
} finally {
    NotesThread.stermThread();
}

new NotesJob("Read mail database title") {
    protected IStatus runInNotesThread(IProgressMonitor monitor) throws NotesException {
        // get session and access mail
        Session session = NotesPlatform.getInstance().getSession();
        DbDirectory dbdir = session.getDbDirectory(null);
        Database db = dbdir.openMailDatabase();
        System.out.println("Title: " + db.getTitle());

        // return
        return Status.OK_STATUS;
    }
}.schedule();
```

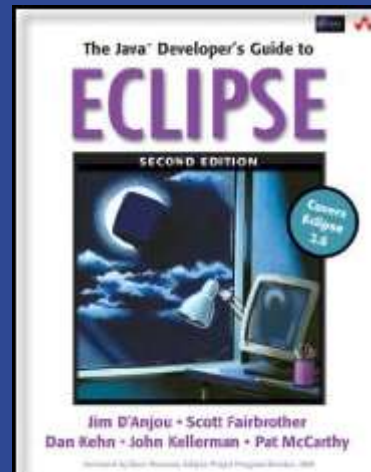
Notes data access and UI interaction (wrong)

```
public void createControl(final Composite parent) {
    final Text textField = new Text(parent, SWT.BORDER);
    new NotesJob("Read mail database title") {
        protected IStatus runInNotesThread(IProgressMonitor mon) throws NotesException {
            // get session and access mail
            Session session = NotesPlatform.getInstance().getSession();
            DbDirectory dbdir = session.getDbDirectory(null);
            final Database db = dbdir.openMailDatabase();

            // schedule UIJob to update UI
            new UIJob("Update UI") {
                public IStatus runInUIThread(IProgressMonitor monitor) {
                    try {
                        textField.setText(db.getTitle());
                    } catch (NotesException e) {}
                    return Status.OK_STATUS;
                }
            }.schedule();

            // return
            return Status.OK_STATUS;
        }
    }.schedule();
}
```

You cannot access the Notes database object from the UI thread!



Notes data access and UI interaction (correct)

```
public void createControl(final Composite parent) {
    final Text textField = new Text(parent, SWT.BORDER);
    new NotesJob("Read mail database title") {
        protected IStatus runInNotesThread(IProgressMonitor mon) throws NotesException {
            // get session and access mail
            Session session = NotesPlatform.getInstance().getSession();
            DbDirectory dbdir = session.getDbDirectory(null);
            Database db = dbdir.openMailDatabase();
            final String title = db.getTitle();

            // schedule UIJob to update UI
            new UIJob("Update UI") {
                public IStatus runInUIThread(IProgressMonitor monitor) {
                    textField.setText(title);
                    return Status.OK_STATUS;
                }
            }.schedule();

            // return
            return Status.OK_STATUS;
        }
    }.schedule();
}
```

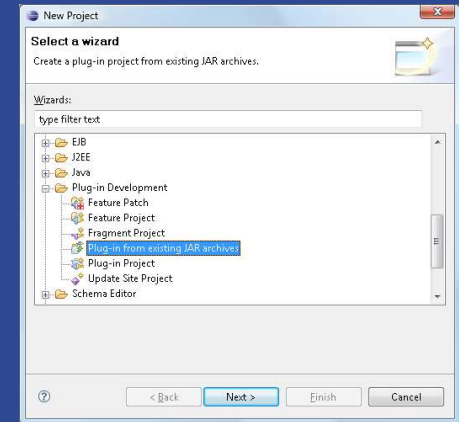
Get database title in Notes thread
and use it in the UI thread

Componentization

- “Applications” should be componentized into logically units i.e. multiple plug-ins (UI / business login)
- Componentizing is also useful for testing and for catering for different platforms and situations
 - ▶ Notes 8 (`NotesPlaform.getInstance().getSession()`)
 - ▶ Sametime (`NotesFactory.createSession()`)
- Think about separating generic parts from more specific parts
- No overhead from using multiple plug-ins vs. keeping it all in one all-encompassing plug-in
- Use extension points to tie your plug-ins together

Componentization - 3rd party code

- Third party code is best distributed as separate plug-ins as it lets you manage the jar-file independently from the plug-in code
 - ▶ Release new versions of the plug-in without redistributing the jar-file plug-in
- Easily, and best, imported into Eclipse using the "New Project"-wizard
 - ▶ Set jar-file version as the plug-in version in the manifest.mf
- Use the equivalent library distributed with Lotus Expeditor/Notes 8 whenever possible (especially for patched libraries such as Jakarta HttpClient)



Being Monet – painting it yourself

- If everything else fails – paint it yourself – remember you have complete control!
- GC (the graphics context)
 - ▶ You use the GC to paint all the basic shapes and text
 - GC.drawRectangle
 - GC.fillRectangle
 - GC.drawText
 - ...
 - ▶ You can set the foreground and background color
 - GC.setBackground(Color)
 - GC.setForeground(Color)
- The JFace viewer framework can still be a good start
- Use the paint event
 - ▶ Typed listener: PaintListener
 - ▶ Untyped listener: SWT.PaintItem

Being Monet – painting it yourself (cont.)

```
public void createPartControl(final Composite parent) {
    // set layout
    parent.setLayout(new FillLayout());

    // create canvas
    final Canvas canvas = new Canvas(parent, SWT.NO_BACKGROUND);

    // add paint listener to paint
    canvas.addPaintListener(new PaintListener() {
        public void paintControl(PaintEvent event) {
            Image img = new Image(event.display, canvas.getBounds());
            GC gc = new GC(img);
            ...
        }
    });
}
```

Demos

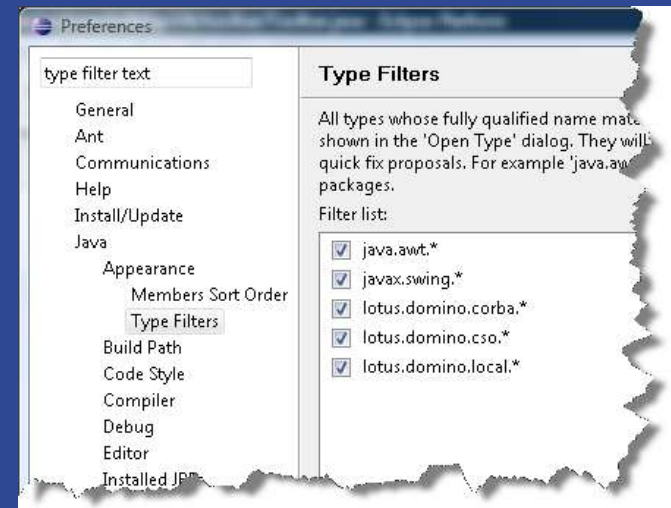
- ▶ com.ls09.bp106.monet
- ▶ com.ls09.bp106.perspective_animation

Tips and tricks

- Use a version qualifier (1.0.0.qualifier) when versioning your plug-ins / features (com.example.myplugin, version 1.0.4.qualifier --> com.example.myplugin_1.0.4.200809111219.jar)
- Be aware of setting version of required plugins to ensure compability
- Use `IConfigurationElement.createExecutableExtension` when instantiating classes from other plug-ins
- When having extensions provide UI elements be sure to isolate yourself from it by supplying a child Composite
- Be aware of threading
- Be aware of platform differences

More tips and tricks

- Use type-filters to avoid strange type-ahead behavior and wrong suggestions
- Remember to change the plug-in id when copy/pasting an entire plug-in
- Set execution environment in the manifest to ensure compatibility
- When in doubt resort to trial-and-error
- Hot code replacement works most of the time when Notes is started in debug mode from Eclipse



Resources

- My blog (lekkimworld.com)
- IBM Lotus Composite Application wiki (www.ibm.com/developerworks/wikis/display/appdev)
- Composite application blog (www.ibm.com/developerworks/blogs/page/CompApps)
- Eclipse.org articles (www.eclipse.org/articles)
- Book: The Java Developer's Guide to Eclipse (2nd Edition)

Related sessions

- BP111 (Dolphin South Hemisphere 1 - just after this session!)
 - ▶ Reports, Charts and Graphs 2.0
- AD305
 - ▶ Building Java Plugins for Lotus Applications
- SHOW103
 - ▶ "Sidebar Safari" -- Controlling and Extending the IBM Lotus Notes Sidebar with Policies, Widgets, Plug-ins and a New API
- AD213
 - ▶ It's Easy! Creating Composite Applications With IBM Lotus Notes, Browser and Java Components

Summary

- Sidebar app. dev. != Notes development
 - ▶ Plug-in development is Java development and should be approached as such
 - ▶ Beware of threading issues (UI / Notes)
 - ▶ Not always for the faint of heart
- Search for Eclipse / XPD code – not Notes code
- Go do it!
 - ▶ Explore! Extend! Exploit!

Q & A

- But...
 - Doesn't that mean...
 - How would you...
 - Could it be that...
-
- Come on up to the microphone and fire away!

Legal disclaimer

© IBM Corporation 2008. All Rights Reserved.

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.

IBM, the IBM logo, Lotus, Lotus Notes, Notes, Domino, Sametime, Expeditor 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.

All references to company names refer to a fictitious companies and are used for illustration purposes only.