JAVADDIN prototype

Version 1.5

Change Notes:
Important: delete all the previous code before reinstalling a new version of Javaddin.

Version 1.5:
- Minor bug fixes
- Dots OSGi profiles now support sites using plugins stored in an NSF database. With this new feature, you can post your plugin in an NSF database based on the UpdateSite.ntf template and have them loaded in dots.
- Added support for security allowing you to configure permissions for each site specified in the OSGi Profile. This feature is using the OSGi Conditional Permission admin service which also supports negative permissions. As an example of OSGi permission, you can prevent plugins to have dependencies on a predetermined set of plugins.
- Remote Control support: you can configure the DOTS OSGi profile to accept remote OSGi console commands, on a port of your choosing. For security purposes, only requests initiated from the same machine are authorized. This is a great feature to, for example, drive a dots tasklet from an XPage application.
- All the new features in 1.5 have been tested with version 3.6.1 of eclipse OSGi. It is recommended to upgrade the OSGi plugins from http://www.eclipse.org/download

Version 1.4:
- Changed the javaddin executable name to dots and updated documentation accordingly
- Added support for OSGi profiles allowing multiple instances of dots to be run and configured
- Bug fix reported on openNTF: runOnStart attribute was not always being triggered right away.
- Changed the name of the list command as it can cause conflicts in some case (when Declarative services plugin is installed). The new command is " tell dots tasklist"
- Added C/C++ binaries for linux32 and linux64

Version 1.3:
- Added support for IBM Lotus Domino Debug plugin
- Better detection of hung tasklets, with automatic stack trace reporting.
- New AbstractServerTaskExt that provides easier support for keyed arguments
- New MultiTask extension point to declaratively create new tasklets by combining existing one. This extension point also allow dynamic arguments using substitution variables.
- New ITaskFilter interface to programmatically allow a tasklet to be loaded or not.
- Can now use regular expressions to list tasklets
- Notes.ini variables to start the tasklet container in debug mode
- C/C++ binaries now also include Win64 version

Version 1.2:
- More supported Extension Manager events (see complete list in section)
- Support for NSFHooks
- OSGi console is now integrated to Javaddin (no need for external telnet)
- New extension point to support aliases
- New extension point to automatically start a plugin
- New debug command to dump thread java stacks
- Bug fixes:
  - OSGi framework was not properly closed upon main task termination
  - Javaddin task is not visible in task list

Version 1.1:
- Tasklets are now running on a different thread than the main java thread. This improves responsiveness of javaddin commands while a long running tasklet is in progress

- Tasklets now support the “triggered” attribute that let a task run based on an Extension Manager event. This feature requires the Javaddin extension manager library to be installed. Note: only a subset of the extension manager events are currently supported in javaddin 1.1. Based on feedback, I will add support for more events as appropriate.
- Tasklets progress can now be monitored and/or canceled. An IProgressMonitor object has been added to the run method signature. Tasklets implementations can provide progress monitoring via the IProgressMonitor argument, they can also be notified if a task has been canceled.
- New javaddin arguments:
  - tell javaddin status: give the status of the current tasks
  - tell javaddin cancel: cancel all the current tasks

Introduction

This document provides information about the Javaddin prototype and its overall architecture including how to install it and run the samples. It also provides information about the source code for each of its components.

Note: it is assumed that you have unzipped the javaddin_prototype.zip in a directory of your choice. This zip file contains the following subdirectories: sources, build.

Javaddin is an experimental generic domino addin task that lets users create Domino server tasks using Java OSGi plugins by creating a tasklet container. A tasklet is a lightweight server adding task written in Java and declared as an OSGi extension point.

There are many benefits in using a java framework like OSGi to write Domino Server tasks:

- Declarative task scheduling
- Built-in mechanism to access server task user arguments.
- List of tell commands to query the OSGi framework.
- Multi-platform support
- Extensibility
- Easy deployment
- Better tooling with Eclipse IDE and its first class Java and plugins editors and Java debugger.
- ...

OSGi itself is a java based framework for running unit of code called bundles. It is usually associated with Eclipse, so a lot people might be surprised that it is used as a server technology. However at its core, OSGi is only a thin runtime that governs how bundles are loaded/stopped (lifecycle), their dependencies, how their services are exposed, etc... You can find more information about OSGi at http://www.osgi.org/Main/HomePage.

Overall architecture

As explained in the diagram below, the javaddin generic server task launches a JVM and invokes code located in launcher.jar to launch the OSGi container.

The OSGi framework located in {dominobin}/javaddin/rcp/eclipse contains the core plugins to run a OSGi framework. The minimum set is comprised of the following list:

1. org.eclipse.update.configurator
2. org.eclipse.osgi
3. org.eclipse.equinox.preferences
4. org.eclipse.equinox.registry
5. org.eclipse.equinox.app
6. org.eclipse.equinox.common
7. org.eclipse.core.runtime.compatibility.auth
8. org.eclipse.core.jobs
9. org.eclipse.core.runtime
10. org.eclipse.core.contenttype

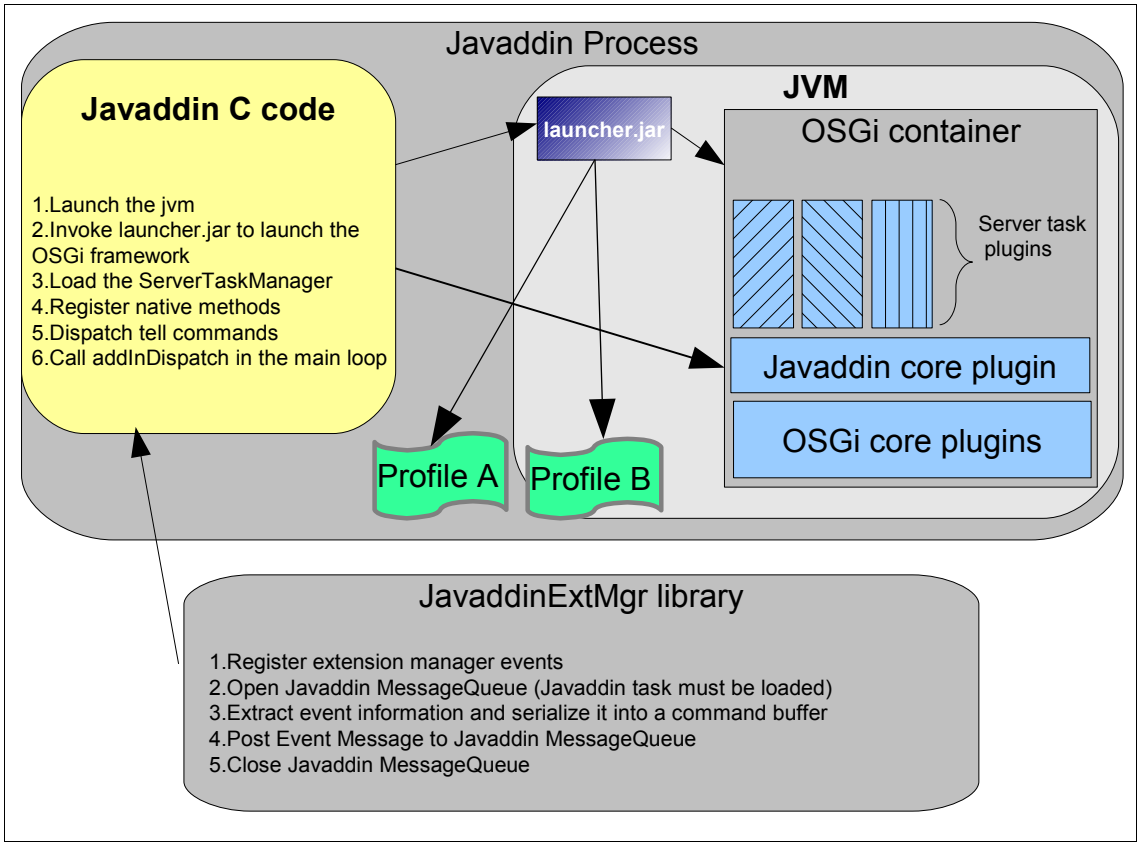
You should be able to simply copy these 10 plugins in the {dominobin}/javaddin/rcp/eclipse/plugins directory from any installed version of Eclipse SDK. If you do not have an eclipse SDK installed on your machine, you can download it from

http://www.eclipse.org/download (there are many flavors of the eclipse IDE in this page, if not sure pick the “Eclipse IDE for Java Developers”).

The “org.openntf.domino.javaddin” plugin located in {dominobin}/javaddin/shared/eclipse/plugins, is part of this prototype. This plugin is essential to communicate between the javaddin server task and the server tasks plugins. Among other things, it provides:

- The extension point that allows a user to register the server task
- The ServerTaskManager instance
- Tell commands implementation

A sample plugin called “org.openntf.javaddin.samples” is included in this prototype. This sample contains a few server tasks to illustrate the different configurations available e.g. Scheduled, runonce, run on startup, etc...



Source code:

The source code is included in the “**sources”** directory and contains the following files:

1. javaddin.cpp: contains the source code for the server task. To recompile it, you will need to install the Notes/Domino C Api Toolkit available here: http://www.ibm.com/developerworks/lotus/downloads/toolkits.html. Please refer to the Notes/Domino C Api toolkit documentation for instructions on how to compile javaddin.cpp.
   Since javaddin.cpp makes use of the jni API, you will also need to install a JDK. For example, you can download one at http://java.sun.com/javase/downloads/index.jsp.
2. JavaddinExtMgr.cpp: contains the source code for the extension manager library that is responsible for posting the events to the tasklet container for processing by the triggered tasklets.
3. JavaddinNSFHook.cpp: contains the source code the the NSFDBHOOK library that is

responsible for posting the NSFDbHook events to the tasklet container for processing by the triggered tasklets.

4. org.openntf.osgi.launcher: Eclipse project containing the source code for launcher.jar which is responsible for launching the OSGi framework. To compile it, it is recommended that you install an Eclipse IDE and load the project in your workspace. Eclipse IDEs can be install from http://www.eclipse.org/download.
5. org.openntf.javaddin.security: (since 1.5) Eclipse project containing the source code for the dots security hook
6. org.openntf.javaddin: Eclipse project containing the source code the core javaddin plugin
7. org.openntf.javaddin.sample: Eclipse project containing the source code for sample server tasks.

Built code:

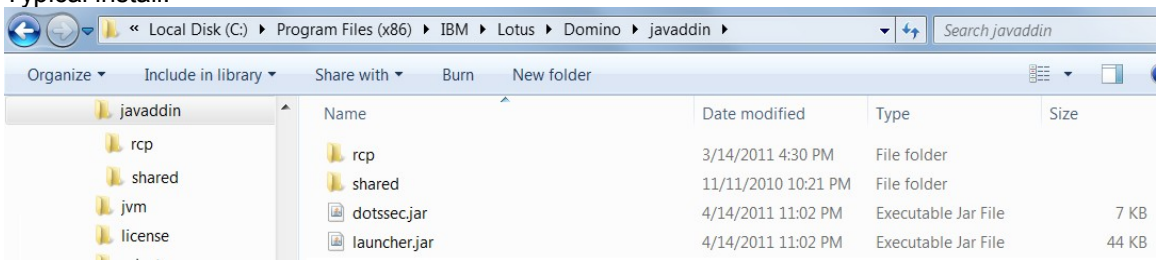
The built code is included in the "**build"** directory. In contains the following files:

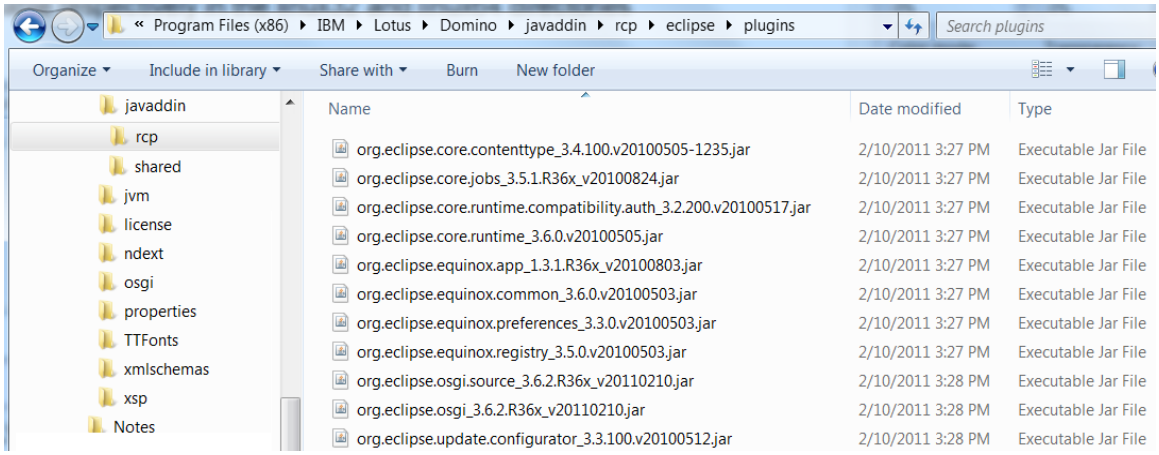
1. dots.exe: executable that must be installed in {dominobin} directory
2. javaddinExtMgr.dll: extension manager library that must be installed if you have triggered tasklets.
3. javaddinNSFHook.dll: NSF Hook library that must be installed if you have triggered tasklets.
4. launcher.jar: jar file that must be installed in {dominobin}/javaddin directory
5. dotssec.jar: jar file that must be installed in {dominobin}/javaddin directory
6. org.openntf.javaddin_1.0.0.XXXX.jar: jar file that must be installed in {dominobin}/javaddin/shared/eclipse/plugins directory
7. org.openntf.javaddin.sample_1.0.0.XXXX.jar: jar file that must be installed in {dominobin}/javaddin/shared/eclipse/plugins directory

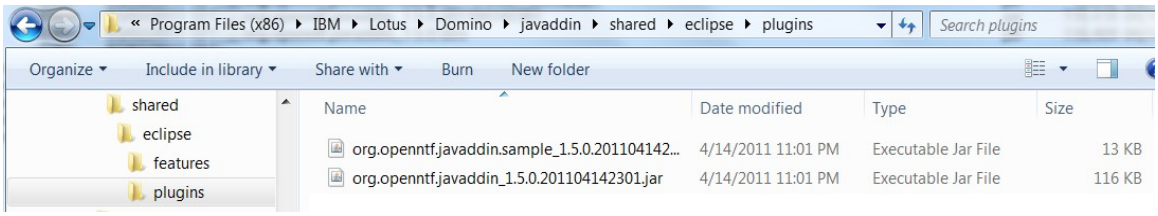
New in 1.3: built win64 versions of dots.exe, javaddinExtMgr.dll and javaddinNSFHook.dll are available in the win64 directory

New in 1.4: build linux32 and linux64 versions of dots, javaddinExtMgr and javaddinNSFHook libraries are available respectively in the linux32 and linux64 directories

Typical install:







To launch the Domino OSGi Tasklet Container, use the following command from the domino server console: “load dots”

New in 1.4: You can now optionally specify an OSGi profile as a command line argument using the -profile switch. This allows you to run multiple instances of the Domino OSGi tasklet container in the same server.
The syntax to specify an OSGi profile is: **load dots -profile [path]** where [path] is the file path to a local XML file. If the specified path is not absolute, then it is assumed to be relative to the {dominoData} directory:
e.g: load dots -profile myprofile.xml

The OSGi profile XML must follow the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<osgiProfile mqName="profile1" workspaceName="workspace1">
        <site url="file:mySite1/eclipse"/>
        <site url="file:mySite2/eclipse"/>
</osgiProfile>
```

osgiProfile element: root element for the profile
mqName attribute: [Must be unique across profiles] Name of the Message Queue. This name is used to identify the dots instance when executing commands: e.g. **tell profile1 run …**
workspaceName attribute: [Must be unique across profiles] Name used as the workspace directory located under {dominodata}/domino directory.
site element: Optional element(s) defining eclipse locations to be contributed to the runtime of the dots instance
url attribute: identifies the location of the eclipse location. Note: right now only file based url are supported (e.g. file:mySite1/eclipse). In the future, we'll also support eclipse locations stored in NSF databases.

Note:
If no profile is specified, then a default profile is used with the following values:
mqName: DOTS
workspaceName: workspace-javaddin
You can also modify the default values by creating an xml profile called dots.xml in your {dominodata} directory.

Extra commands added to manage profiles

- **show tasks**: gives a list of all the dots instances and their MQName
- **tell [profileMQName] showprofile:** print information on the specified profile.
- **tell [profileMQName] exit**: terminate the specified instance of dots. We've added this new command because using tell dots q would terminate all the instances of dots. Note: you can also use **tell [profileMQName] close**.

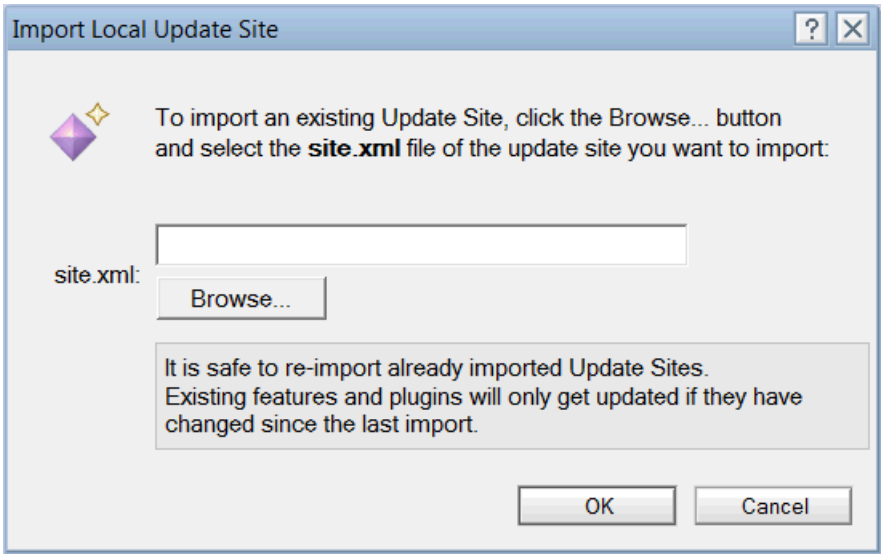
New in 1.5: There is a new “osginsf” protocol supported by the site url attribute. This allows you to

include plugins stored in an NSF database as follow: osginsf:<path to the nsf database>
Optionally: you can specify a list of “feature” elements to include. If none is specified, then all the features found in the database will be loaded. Note: in 1.5, the feature element only works with “osginsf” sites and not with “file” sites.
Optionally: you can specify a list of “security” elements to configure permissions per site (more info on security below in this document).
e.g.

```
<site url=”osginsf:myUpdateSite.nsf”>
    <feature id=”feature1”/>
    <feature id=”feature2”/>
</site>
```

Steps:

- Package your plugin(s) as an update Site. From eclipse, you will need to create a feature project and an Update Site project. Please refer to eclipse documentation for details on how to do that.
- Create myUpdateSite.nsf using the standard updateSite.ntf template.
- In MyUpdateSite.nsf, upload your update site using the “Import Local Update Site” buttom:



Security

New in 1.5, for each “site” element, you can optionally specify a “security” child element, which itself can have 2 types of child elements:

- “deny” element: list of permissions that are explicitly denied to the plugins of the site
- “allow” element: list of permissions that are explicitly allowed to the plugins to the site.

Note: you can disable security altogether by using **disableSecurity=”true”** attribute in the osgiProfile root element.

Although you could specify both deny and allow element, in practice it make more sense to specify only one of them based on the level of security you want to assign. Specifying only deny means that the plugins have all permissions except the one declared in deny. In turn, specifying only allow means that the plugins have no permission except the one declared in allow.

Each permission must be declared on a separate line. The syntax is similar to the Java 2 permission syntax:

<<*Permission class name*>> “<<*target name*>>” “<<*actions*>>”

Note: <<actions>> can be combined using a comma separator

for example:

```
<security>
    <deny>
       java.net.SocketPermission "www.ibm.com" "connect"
       org.osgi.framework.BundlePermission "org.openntf.javaddin" "require"
```

```
        java.io.FilePermission "&lt;&lt;ALL FILES&gt;&gt;" "read"
    </deny>
</security>
```

or

```
<security>
    <allow>
        org.osgi.framework.BundlePermission "org.osgi.*" "require"
        org.osgi.framework.BundlePermission "*" "provide,host,fragment"
        java.io.FilePermission "/local/notesdata/*" "read"
    </allow>
</security>
```

Every permission class define their own action, please refer the java sdk documentation for a complete list of all the permissions and their actions.

What's interesting here is that you can also specify Permissions dedicated to OSGi plugins and used by the OSGi Conditional Permission Admin Service (for more information on this service, please refer to http://www.osgi.org). These permissions allow you to control things like “preventing plugins to have a dependency on a particular plugin” or “preventing plugins to be a fragment to a particular plugin”, etc...

The following table gives a complete list of all the OSGi Permissions and their associated actions:

Permission Name	Actions	Target	Method
org.osgi.framework.AdminPermission	class	None	Bundle.loadClass
	execute	None	Bundle.start Bundle.stop StartLevel.setBundleStartLevel
	extensionLifecycle	None	BundleContext.installBundle for extension bundles Bundle.update for extension bundles Bundle.uninstall for extension bundles
	lifecycle	None	BundleContext.installBundle Bundle.update Bundle.uninstall
	listener	None	BundleContext.addBundleListener for SynchronousBundleListener BundleContext.removeBundleListener for SynchronousBundleListener
	metadata	None	Bundle.getHeaders Bundle.getLocation
	resolve	None	PackageAdmin.refreshPackages PackageAdmin.resolveBundles
	resource	None	Bundle.getResource Bundle.getResources Bundle.getEntry Bundle.getEntryPaths Bundle.findEntries Bundle resource/entry URL creation
	startlevel	None	StartLevel.setStartLevel StartLevel.setInitialBundleStartLevel
	context	None	Bundle.getBundleContext
org.osgi.framework.BundlePermission	provide	Bundle symbolic name. Wildcards may be used. For example: org.osgi.example.bundle org.osgi.example.*	
	require		
	host		
	fragment		
org.osgi.framework.PackagePermission	Export (deprecated use exportonly instead)	Fully qualified package name for example: org.osgi.service.http	
	exportonly		
	import		

org.osgi.framework.ServicePermission	get	Target: name of the service
	register	The register action allows a bundle to register a service on the specified names. The get action allows a bundle to detect a service and get it.

Note: not specifying any “security” element to a site has a different meaning based on the site protocol. For sites using the “file” protocol, the plugins run unrestricted. However, for sites using the “osginsf” protocol, then a set a default allowed permissions is automatically assigned to the plugins. The default allowed permisssions for “osginsf” sites are specified in the nsf.policy file located in the org.openntf.osgi.launcher package:

```
//List of default permissions for any NSF Based plugins
java.lang.RuntimePermission "stopThread"

// allows anyone to listen on un-privileged ports
java.net.SocketPermission "localhost:1024-" "listen"

// "standard" properies that can be read by anyone
java.util.PropertyPermission "java.version" "read"
java.util.PropertyPermission "java.vendor" "read"
java.util.PropertyPermission "java.vendor.url" "read"
java.util.PropertyPermission "java.class.version" "read"
java.util.PropertyPermission "os.name" "read"
java.util.PropertyPermission "os.version" "read"
java.util.PropertyPermission "os.arch" "read"
java.util.PropertyPermission "file.separator" "read"
java.util.PropertyPermission "path.separator" "read"
java.util.PropertyPermission "line.separator" "read"

java.util.PropertyPermission "java.specification.version" "read"
java.util.PropertyPermission "java.specification.vendor" "read"
java.util.PropertyPermission "java.specification.name" "read"

java.util.PropertyPermission "java.vm.specification.version" "read"
java.util.PropertyPermission "java.vm.specification.vendor" "read"
java.util.PropertyPermission "java.vm.specification.name" "read"
java.util.PropertyPermission "java.vm.version" "read"
java.util.PropertyPermission "java.vm.vendor" "read"
java.util.PropertyPermission "java.vm.name" "read"

java.util.PropertyPermission "javax.realtime.version" "read"

//OSGi specific permissions
org.osgi.framework.BundlePermission "org.eclipse.*" "require"
org.osgi.framework.BundlePermission "org.osgi.*" "require"
org.osgi.framework.BundlePermission "org.openntf.javaddin" "require"

org.osgi.framework.PackagePermission "org.eclipse.*" "import"
org.osgi.framework.PackagePermission "org.osgi.*" "import"
org.osgi.framework.PackagePermission "lotus.domino" "import"
```

For some OSGi framework permissions, you may not get a visible exception in the console. For example, if a bundle doesn't have the correct require BundlePermission, then this bundle will not load correctly and will have the installed state:

```
> tell david diag com.ibm.nsfbased.tasklet
[1E88:0002-108C] 04/07/2011 12:24:50 PM  [DAVID] initial@osginsf:dotsupdateSite.nsf/3E08D9B640C1A739852578690052F858/com.ibm.nsfbased.taskle
t_1.0.0.201104042235.jar [8]
[1E88:0002-108C] 04/07/2011 12:24:50 PM  [DAVID]
[1E88:0002-108C] 04/07/2011 12:24:50 PM  [DAVID]
[1E88:0002-108C] 04/07/2011 12:24:50 PM  [DAVID] Missing Permission: (org.osgi.framework.BundlePermission org.openntf.javaddin require)
[1E88:0002-108C] 04/07/2011 12:24:50 PM  [DAVID]
```

Remote Controller

New in 1.5: you can specify a remoteCommands element as a child of the osgiProfile root element as follow:

```
<osgiProfile>
            …
            <remoteCommands type=”embedded” port=”1234”/>
            ….
</osgiProfile>
```

type: indicate the type remoteCommands. Only “embedded” is supported today, which open a socket on port specified by the port attribute and listen to remote OSGi console commands requests.

port: port number used by the remote Command controller. If none specified, then DOTS will use any free port. A transient file called port.txt containing the port number in use will be created for the duration of the session. This file is located in the workspace directory for the dots instance. Remote applications can open this file to know which port to use to issue remote requests.

For security reasons, only request coming from the local machines are authorized.

The following shows a sample client code to issue remote OSGi commands to DOTS:

```
private static String sendDotsRequest(String command) throws Exception {
    //1234 is the port specified in the profile
    Socket socket = new Socket("localhost", 1234 );
    socket.getOutputStream().write( command.getBytes() );
    socket.shutdownOutput();
    socket.setSoTimeout( 3000 );  //Timeout to 3 s
    InputStream is = socket.getInputStream();
    StringBuilder sb = new StringBuilder(1024);
    int c;
    while ( ( c = is.read()) != -1 ){
           sb.append( (char)c );
    }
    socket.close();
    return sb.toString();
}
```

Dots tell commands:

help:	Display this help
close/exit:	Close the current instance of the Domino OSGi Tasklet Container
tasklist:	Display list of OSGi server tasks
run [taskid]:	Run the task once specified by the task id
info [taskid]:	Display detailed information about the task
status:	Display information about the tasks being run
cancel:	Cancel the current task.

Examples:

```
> tell dots tasklist
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.sendMail
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    Sample task that send a summary email for all the databa
ses available on the server. Options: [emailaddress]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.multiTask1
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    null
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.triggered
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    Sample task triggered by Extension manager events
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.runOnStart
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    null
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.scheduled
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    Scheduled Task
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS]
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] id:     org.openntf.sample.listViews
[2154:0002-04D8] 03/02/2011 11:50:02 AM  [DOTS] description:    Sample task that list the views of a database. Options:
```

```
> tell dots info org.openntf.sample.scheduled
[2744:0002-1144] 02/28/2011 02:53:10 PM  [DOTS]
[2744:0002-1144] 02/28/2011 02:53:10 PM  [DOTS] id:     org.openntf.sample.scheduled
[2744:0002-1144] 02/28/2011 02:53:10 PM  [DOTS] description:    Scheduled Task
[2744:0002-1144] 02/28/2011 02:53:11 PM  [DOTS] Run on Start:   false
[2744:0002-1144] 02/28/2011 02:53:11 PM  [DOTS] Runs every 5 - minute
[2744:0002-1144] 02/28/2011 02:53:11 PM  [DOTS] Runs every 1 - day
```

```
> tell dots status
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]         Scheduled Runs : No task running
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]         Triggered Runs : Not running
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]
[2744:0002-1144] 02/28/2011 02:54:59 PM  [DOTS]         Manual Runs : org.openntf.sample.listViews [List the views] (9.8
04%)
> tell dots cancel
[2744:0002-1144] 02/28/2011 02:55:48 PM  [DOTS] 1 task(s) cancelled
```

New in 1.3: you can now use regular expressions as an argument to the list command e.g.
tell dots list org.openntf.*

Description of the org.openntf.javaddin.task extension point:

A detailed description of the "org.openntf.javaddin.task" extension point is availble in the pde_schema_org-openntf-javaddin-task_preview.html file included in the distribution zip file.

Server Addin Task

Identifier: org.openntf.javaddin.task

Since: [Enter the first release in which this extension point appears.]

Description: [Enter description of this extension point.]

Configuration Markup:

```
<!ELEMENT extension (task+)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id    CDATA #IMPLIED
  name  CDATA #IMPLIED
>

<!ELEMENT task (run)*>
<!ATTLIST task
  class       CDATA #REQUIRED
  id          IDREF #REQUIRED
  description CDATA #IMPLIED
  runOnStart  (true | false)
  triggered   (true | false)
  filter      CDATA #IMPLIED
>
```

- **class** - Fully qualified class name, must extend org.openntf.javaddin.task.AbstractServerTask.
- **id** - Unique id for this task
- **description** - Task description
- **runOnStart** - Indicates if the task should be run once when the javaddin task starts. True by default
- **triggered** - If true, this task will be triggered by extension manager event. RunWhen contains the information about the event.
- **filter** - Optional class that must implement the org.openntf.javaddin.task.ITaskFilterITaskFilter interface to dynamically prevent this task from loading in the container.

```
<!ELEMENT run EMPTY>
<!ATTLIST run
  every CDATA #REQUIRED
  unit  (second|minute|day)
>
```

- **every** - Schedule Frequency for the server task
- **unit** - Frequency unit for the Task Can be second, minute or day. If day, then "every" attribute is not used and the task will run once a day.

The "org.openntf.javaddin.task" extension point requires a class that must extends org.openntf.javaddin.task.AbstractServerTask. In this class, the user must override the following methods:

```
        /**
         * @param serverTaskInfo
         * Called before the task is ready to run
         * Note: a default implementation is provided in AbstractServerTask, subclasses
must call super if overriding this method     */
        public void init( ServerTaskInfo serverTaskInfo );

        /**
         * @param runWhen
         * @param args: command line arguments
         * @param monitor
         * @throws NotesException
         */
        public void run(RunWhen runWhen, String[] args, IProgressMonitor monitor) throws
NotesException;

        /**
         * Called when the task instance is being discarded
         * Task should recycle any created backend objects
         */
        public void dispose() throws NotesException;
```

The IProgressMonitor argument can be used to notify the tasklet container of the progress of the current task, but also to be notified if the user wants to cancel the current task. The ListViewExample tasklet illustrate how a long running tasklet uses the IProgressMonitor:

```
/* (non-Javadoc)
	* @see
org.openntf.javaddin.task.IServerTaskRunnable#run(org.openntf.javaddin.task.RunWhen, java.lang.String[], org.eclipse.core.runtime.IProgressMonitor)
	*/
	public void run(RunWhen runWhen, String[] args, IProgressMonitor monitor)
throws NotesException {
		....
			Vector<?> views = db.getViews();
			monitor.beginTask( "List the views", views.size() );
			logMessage("List of views for db: " + db.getTitle() );
			for ( Object view : views ){
				if ( monitor.isCanceled() ){
					break;
				}
				logMessage("\t" + ((View)view).getName() );
				try {
					Thread.sleep( 1000 );
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				monitor.worked( 1 );
			}
		...
	}
```

For more information, please look at the sample tasks provided in “org.openntf.javaddin.sample” plugin.
New in 1.3: You can now use the AbstractServerTaskExt as the base class of your tasklet, which give you support for keyed arguments via the following methods:

```
	/**
	 * @param key
	 * @return: the value for the keyed argument
	 */
	public String getKeyedArgument( String key )
	 /**
	 * @param n
	 * @return: the Nth arguments as it was passed to the task by the user
	 */
	public String getNthArgument( int n )
```

Description of the org.openntf.javaddin.multitask extension point:
New in 1.3, the new alias extension point allows you to create a new task by combining existing tasks. This can be very useful for example if you need to build automated test suites that run multiple tasks
For example:

```
<extension point="org.openntf.javaddin.multitask">
	<multiTask
		id="org.openntf.sample.multiTask1">
		<taskRef id="org.openntf.sample.sendMail">
			<param name="to" value="john.smith@acme.com"/>
			<param name="cc" value="fred.manager@acme.com"/>
		</taskRef>
		<taskRef id="org.openntf.sample.sendMail">
			<param name="to" value="ted.smith@acme.com"/>
			<param name="bcc" value="bill.vp@acme.com"/>
		</taskRef>
		<taskRef id="org.openntf.sample.sendMail">
			<param name="to" value="al.smith@acme.com"/>
		</taskRef>
	</multiTask>
</extension>
```

A multitask extension point is composed of one or more taskRef elements that point to an existing task. For each taskRef element, you can pass one or more param (key/value pair), these params element can also be dynamic using the following notation: {var}. Dynamic variables can be resolved using the argResolver extension point (see org.openntf.sample plugin for an example on how to use the argResolver extension point)

Description of the org.openntf.javaddin.alias extension point:

New in Javaddin 1.2.0, the new alias extension point allows you to configure a task with predefined arguments

For example:

```
<extension point="org.openntf.javaddin.alias">
   <alias id="sendMailtoJohn"
        targetTask="org.openntf.sample.sendMail"
        description="Send a mail to John">
        <arg name="johndoe@acme.com" value=""/>
   </alias>
</extension>
```

You can check the list of aliases currently installed by using the following command:
tell javaddin aliases

Description of the org.openntf.javaddin.startup extension point:

New in Javaddin 1.2.0, you can automatically start a plugin by using the org.openntf.javaddin.startup extension point.

For example:

```
    <extension point="org.openntf.javaddin.startup">
       <startup class="org.openntf.sample.Startup"/>
    </extension>
```

The Sartup class must implement the org.openntf.javaddin.startup.IStartup interface:

```
   public interface IStartup {
        /**
         * Method called just after the plugin has been activated
         */
        public void earlyStartup();
   }
```

Support for extension Manager events:

New in Javaddin 1.1, you can use the "triggered" attribute when declaring the tasklet in the plugin.xml. When doing so, the task run method will be called when one of the supported extension manager events occur. It is important to understand that the task is not called synchronously when the even occurs, rather a message containing information about the event is posted to Javaddin task container. In turn, Javaddin will service the event as soon as possible. So you cannot use this mechanism to affect the outcome of the extension manager, but it provides a nice mechanism to trigger a task based on an event rather than polling.

To work, you must install the javaddin extension manager library provided in the distribution zip by adding the following line to notes.ini:

EXTMGR_ADDINS=javaddinExtMgr.dll

The TriggeredExample tasklet illustrate how to use "triggered" attribute:

In plugin.xml:

```
    <task id="org.openntf.sample.triggered"
    description="Sample task triggered by Extension manager events"
    class="org.openntf.javaddin.sample.TriggeredExample"
    runOnStart="false"
    triggered="true">
   </task>
```

When the task is event triggered, the RunWhen object contains an IextensionManagerEvent object that can be used to access information about the event.

In TriggeredExample.java:

```
public void run(RunWhen runWhen, String[] args, IProgressMonitor monitor)throws
NotesException {
    if ( runWhen.getUnit() != RunUnit.triggered ){
            //We only support triggered events
```

```
                return;
        }

        IExtensionManagerEvent event = runWhen.getExtensionManagerEvent();
        if ( event.getEventId() == IExtensionManagerEvent.EM_NSFNOTEUPDATEXTENDED ){
           NSFNoteUpdateExtendedEvent updateEvent = (NSFNoteUpdateExtendedEvent)event;
           ...

        }else if ( event.getEventId() == IExtensionManagerEvent.EM_NSFDBCREATE ){
           NSFDbCreateEvent dbCreateEvent = (NSFDbCreateEvent)event;
           logMessage( "Trigger event: EM_NSFDBCREATE with dbPath = " +
dbCreateEvent.getDbPath() );

        }else if ( event.getEventId() == IExtensionManagerEvent.EM_NSFDBDELETE ){
           NSFDbDeleteEvent dbDeleteEvent = (NSFDbDeleteEvent)event;
           logMessage( "Trigger event: EM_NSFDBDELETE with dbPath = " +
dbDeleteEvent.getDbPath() );

        }else if ( event.getEventId() == IExtensionManagerEvent.EM_NSFNOTEDELETE ){
           NSFNoteDeleteEvent noteDeleteEvent = (NSFNoteDeleteEvent)event;
           logMessage(
                   MessageFormat.format( "Trigger event: EM_NSFNOTEDELETE with dbPath = {0}
and NoteId {1}",
                   noteDeleteEvent.getDbPath(), noteDeleteEvent.getNoteId()
                   )
           );
        }
}
```

If you want to know the list of events currently supported, please look in the javaddinExtMgr.cpp source code provided in the distribution zip

New in Javaddin 1.2, enhanced list of supported extension manager events. Please look in the source code of each event class to know the available fields.

Extension Manager Event	Associated Java class
EM_NSFDBCLOSE	NSFDbCloseEvent
EM_NSFNOTEUPDATE	NSFNoteUpdateExtendedEvent
EM_NSFNOTEUPDATEXTENDED	NSFNoteUpdateExtendedEvent
EM_NSFDBCREATE	NSFDbCreateEvent
EM_NSFDBDELETE	NSFDbDeleteEvent
EM_NSFNOTECREATE	NSFNoteCreateEvent
EM_NSFNOTEDELETE	NSFNoteDeleteEvent
EM_ADMINPPROCESSREQUEST	AdminPProcessRequestEvent
EM_NSFNOTEOPEN	NSFNoteOpenEvent
EM_NSFNOTECLOSE	NSFNoteCloseEvent
EM_NSFNOTEOPENBYUNID	NSFNoteOpenByUNIDEvent
EM_FTINDEX	FTIndexEvent
EM_FTSEARCH	FTSearchEvent
EM_NIFFINDBYKEY	NIFFindByKeyEvent
EM_NIFFINDBYNAME	NIFFindByNameEvent
EM_NIFOPENNOTE	NIFOpenNoteEvent
EM_NIFREADENTRIES	NIFReadEntriesEvent
EM_NIFUPDATECOLLECTION	NIFUpdateCollectionEvent
EM_NSFDBCOMPACT	NSFDbCompactEvent
EM_NSFDBDELETENOTES	NSFDbDeleteNotesEvent
EM_NSFDBWRITEOBJECT	NSFDbWriteObjectEvent
EM_NIFOPENCOLLECTION	NIFOpenCollectionEvent
EM_NIFCLOSECOLLECTION	NIFCloseCollectionEvent
EM_NSFDBRENAME	NSFDbRenameEvent

EM_NSFDBREOPEN	NSFDbReopenEvent
EM_NSFDBOPENEXTENDED	NSFDbOpenExtendedEvent
EM_NSFNOTEOPENEXTENDED	NSFNoteOpenExtendedEvent
EM_TERMINATENSF	TerminateNSFEvent
EM_NSFNOTEDECRYPT	NSFNoteDecryptEvent
EM_NSFCONFLICTHANDLER	NSFConflictHandlerEvent
EM_MAILSENDNOTE	MailSendNoteEvent
EM_CLEARPASSWORD	ClearPasswordEvent
EM_SCHFREETIMESEARCH,	SCHFreeTimeSearchEvent
EM_SCHRETRIEVE,	SCHRetrieveEvent
EM_SCHSRVRETRIEVE,	SCHSrvRetrieveEvent
EM_NSFDBCOMPACTEXTENDED,	NSFDbCompactEvent
EM_NSFDBCOPYNOTE,	NSFDbCopyNoteEvent
EM_NSFNOTECOPY,	NSFNoteCopyEvent
EM_NSFNOTEATTACHFILE,	NSFNoteAttachFileEvent
EM_NSFNOTEDETACHFILE,	NSFNoteDetachFileEvent
EM_NSFNOTEEXTRACTFILE,	NSFNoteExtractFileEvent
EM_NSFDBCOPY,	NSFDbCopyEvent
EM_NSFDBCREATEANDCOPY,	NSFDbCreateAndCopyEvent
EM_NSFDBCOPYACL,	NSFDbCopyACLEvent
EM_NSFDBCOPYTEMPLATEACL,	NSFDbCopyTemplateACLEvent
EM_NSFDBCREATEACLFROMTEMPLATE	NSFDbCreateACLFromTemplateEvent
EM_FTDELETEINDEX	FTDeleteIndexEvent
EM_FTSEARCHEXT,	FTSearchExtEvent
EM_NAMELOOKUP,	NameLookupEvent
EM_NSFNOTEUPDATEMAILBOX,	NSFNoteUpdateMailBoxEvent
EM_AGENTOPEN,	AgentOpenEvent
EM_AGENTRUN,	AgentRunEvent
EM_AGENTCLOSE,	AgentCloseEvent
EM_SECAUTHENTICATION,	SECAuthenticationEvent
EM_NAMELOOKUP2,	NameLookup2Event
EM_NSFADDTOFOLDER,	NSFAddToFolderEvent
EM_ROUTERJOURNALMESSAGE,	RouterJournalMessageEvent
EM_SMTPCONNECT,	SMTPConnectEvent
EM_SMTPCOMMAND,	SMTPCommandEvent
EM_SMTPMESSAGEACCEPT,	SMTPMessageAcceptEvent
EM_SMTPDISCONNECT,	SMTPDisconnectEvent
EM_NSFARCHIVECOPYNOTES,	NSFArchiveCopyNotesEvent
EM_NSFARCHIVEDELETENOTES,	NSFArchiveDeleteNotesEvent
EM_MEDIARECOVERY_NOTE,	MediaRecoveryEvent
EM_NSFNOTECIPHERDECRYPT,	NSFNoteCipherDecryptEvent
EM_NSFNOTECIPHEREXTRACTFILE	NSFNoteCipherExtractFileEvent
EM_NSFDBNOTELOCK	NSFDbNoteLockEvent
EM_NSFDBNOTEUNLOCK	NSFDbNoteUnlockEvent
HOOK_EVENT_NOTE_UPDATE	NSFHookNoteUpdateEvent
HOOK_EVENT_NOTE_OPEN	NSFHookNoteOpenEvent

Support for NSFDBHOOK events:

New in Javaddin 1.2, similar to extension manager events, the "triggered" attribute can also be used for NSFDBHOOK events. To enable it, you must install the javaddin NSFDBHOOK library provided in the distribution zip by adding the following line to notes.ini:

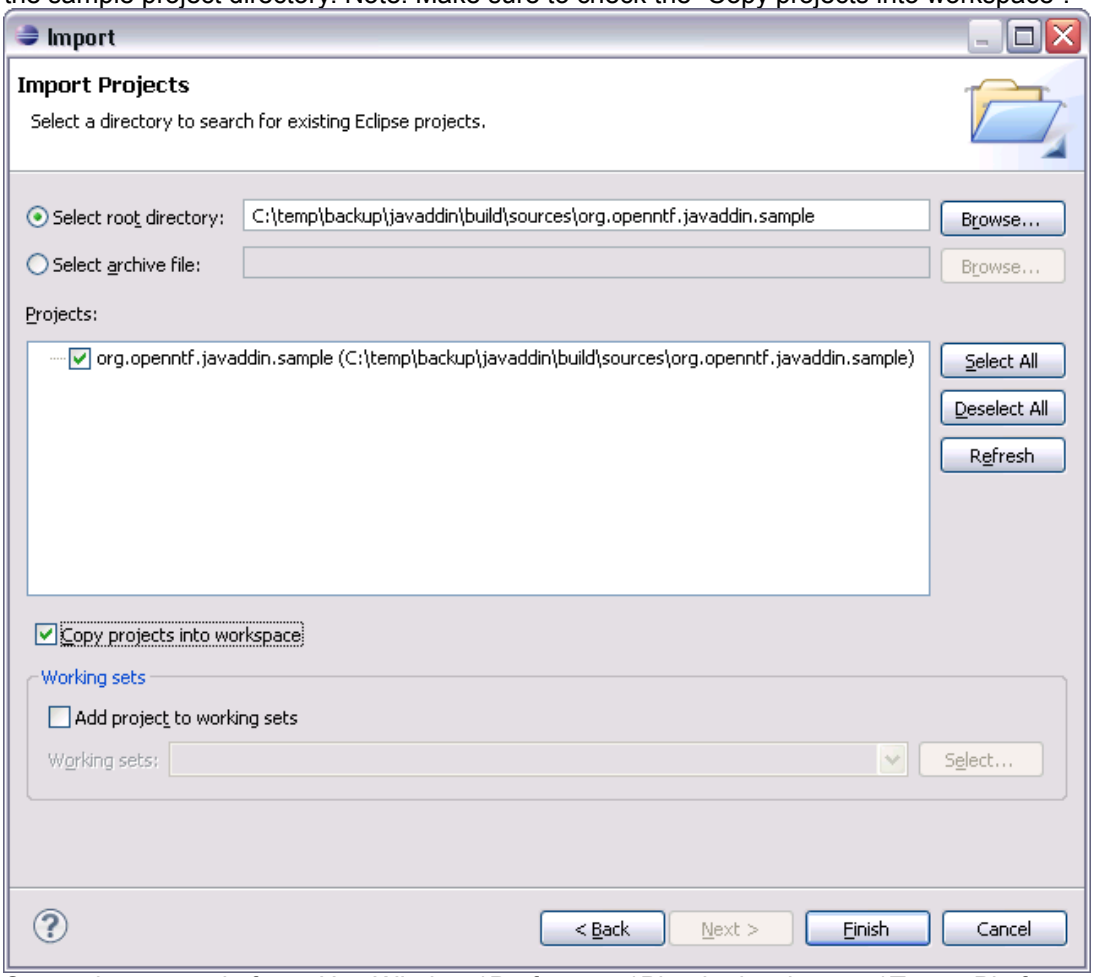
NSF_HOOKS=javaddinNSFHook

NSFDBHOOK Event	Associated Java class
HOOK_EVENT_NOTE_UPDATE	NSFHookNoteUpdateEvent
HOOK_EVENT_NOTE_OPEN	NSFHookNoteOpenEvent

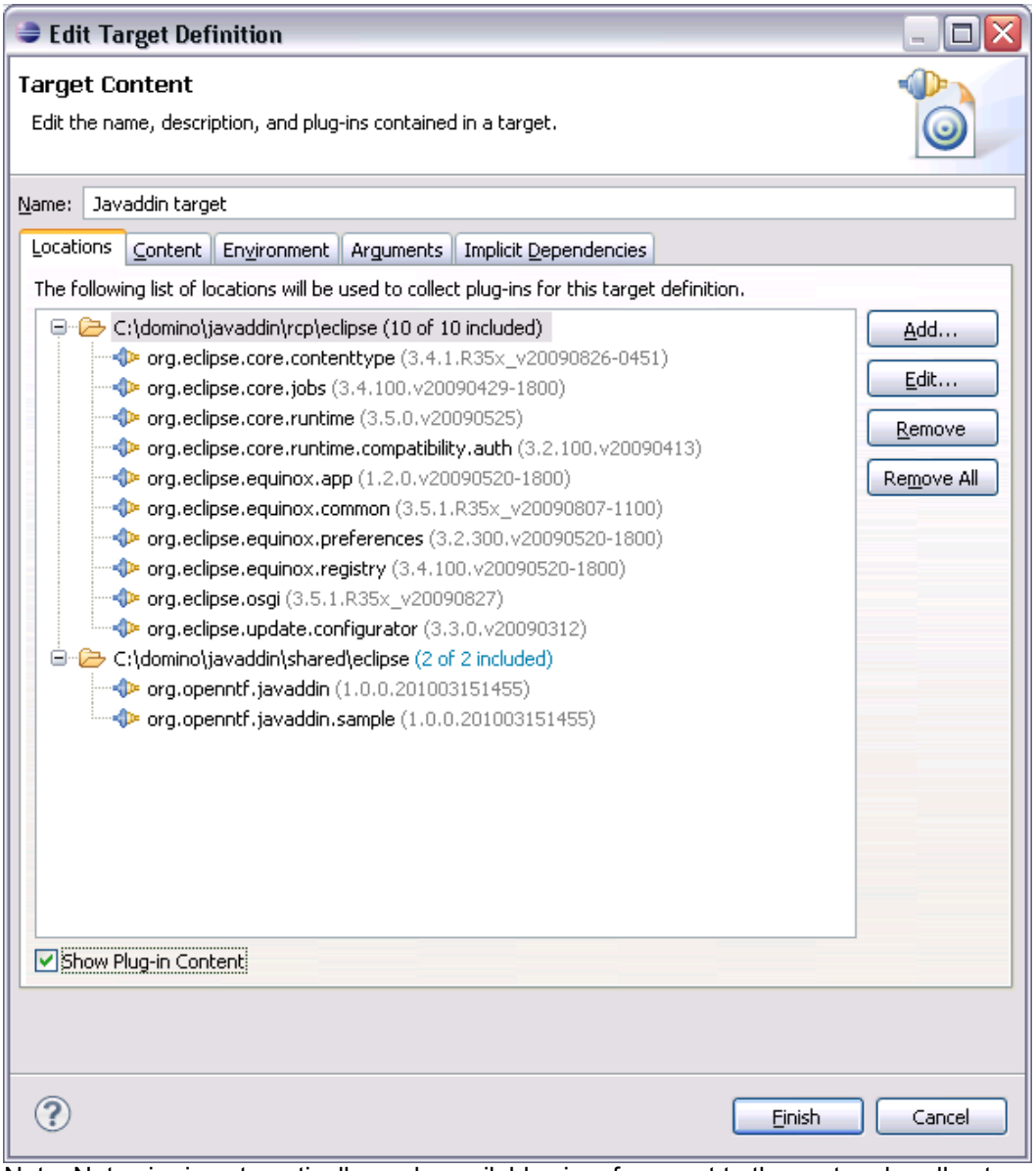
Setting up the dev environment to work with the sample plugin:

The recommended version for the Eclipse sdk : 3.5.1

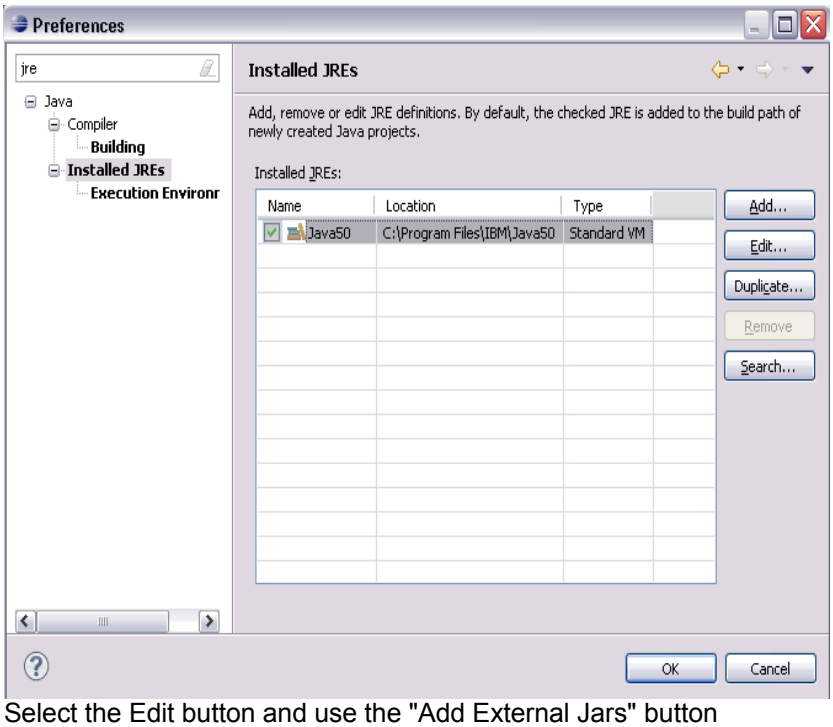
1. Import the source code for the sample project org.openntf.javaddin.sample located in the sources directory: use file\import menu, and select "General\Existing projects into Workspace". In the next page use "select root directory:" radio button, then use the browser button to load the sample project directory. Note: Make sure to check the "Copy projects into workspace".



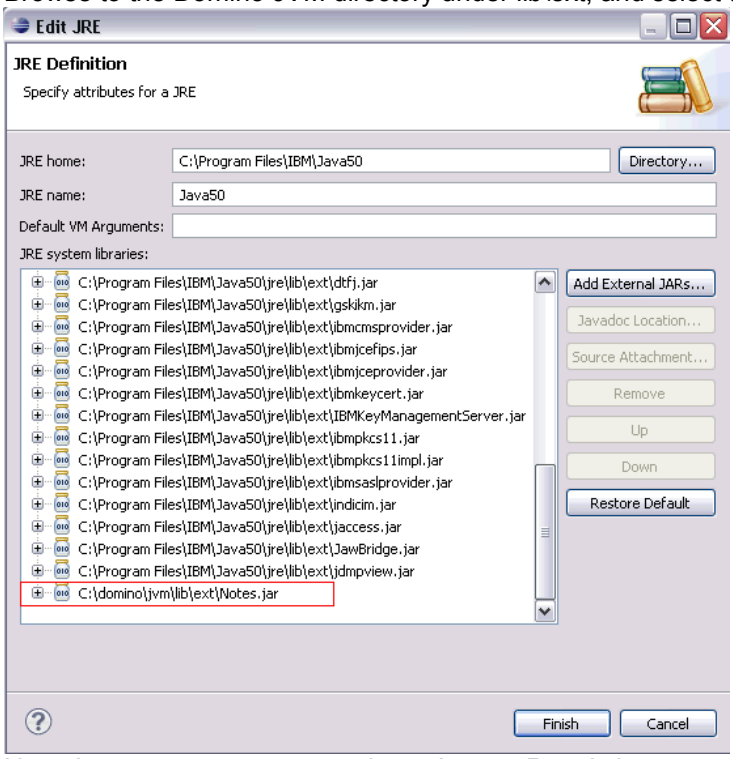
2. Set up the target platform: Use Windows\Preferences\Plug-in development\Target Platform and add the following 2 locations {dominobin}/javaddin/rcp/eclipse and {dominobin}/javaddin/shared/eclipse. The following screen shot shows how your target platform should look like:



3. Note: Notes.jar is automatically made available via a fragment to the system bundle at runtime, but not at compile time. To fix this problem, you need to add the notes.jar to the JRE and add a dummy plugin to the target platform that exports the lotus.domino package.
    1. Add Notes.jar to the JRE settings: Use Windows/Preferences and type JRE in the filter text

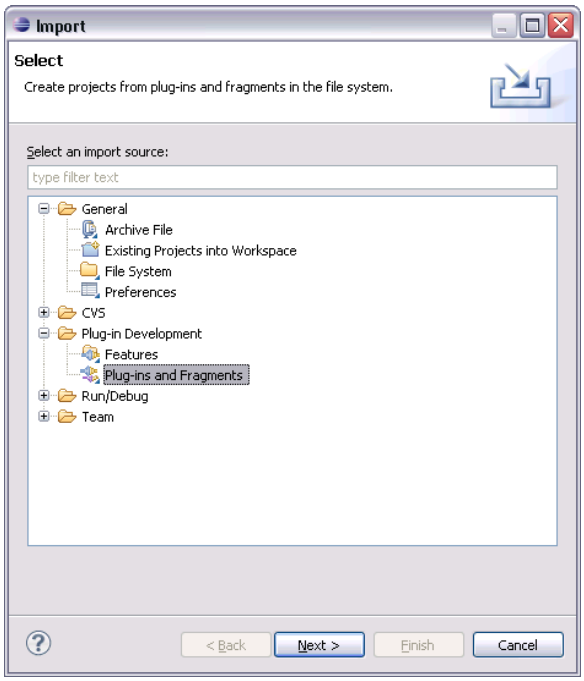


Select the Edit button and use the "Add External Jars" button
Browse to the Domino JVM directory under lib\ext, and select the Notes.jar:

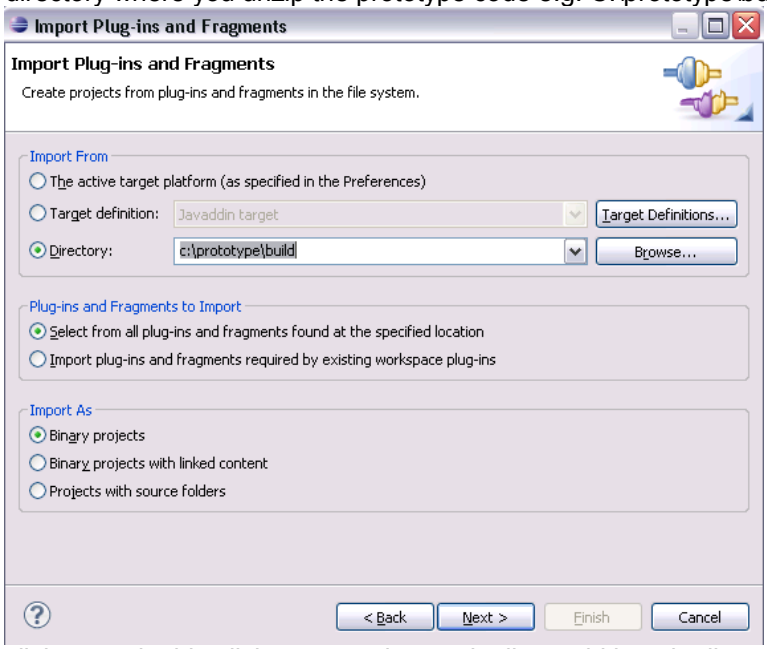


Note: In some cases, you may have Access Restriction errors on lotus.domino packages, this is because some plugin specify an Execution-Environment of 1.5 in the manifest. To fix this problem, you can do the following: use Windows\preferences\Java\Compiler\Errors-Warnings\Deprecated and resticted API and select Ignore in the Forbidden references (access rules) field

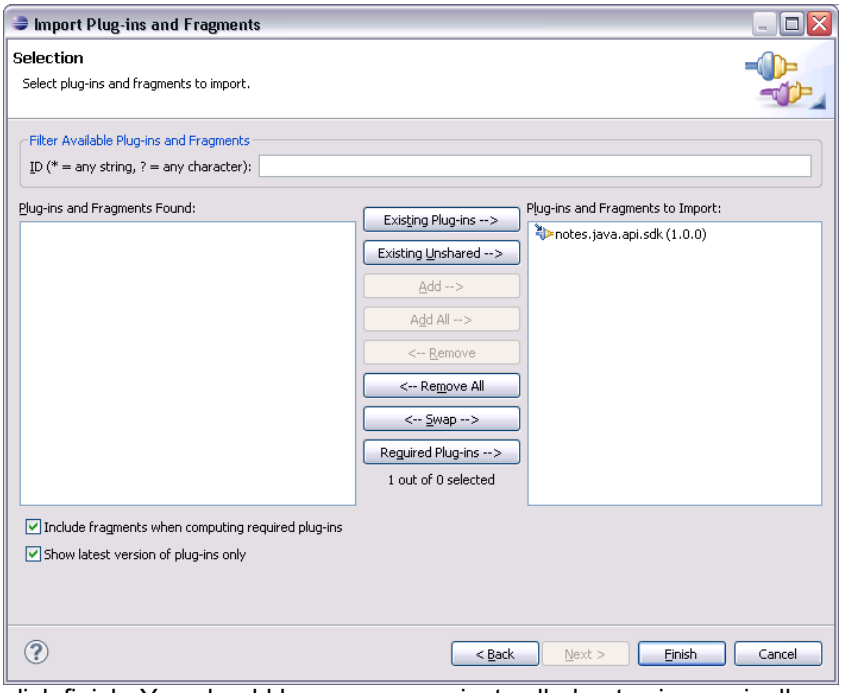
2. Import the “notes.java.api.sdk” plugin located in the build directory: use file\import menu and select “Plug-ins and Fragments”:



Click next. In the Import from section, use “Directory” radio button and select the build directory where you unzip the prototype code e.g. C:\prototype\build:

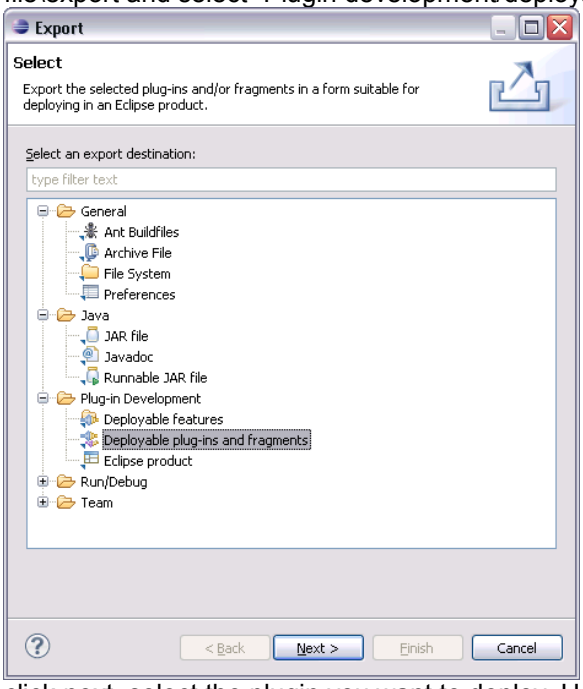


click next, double click on notes.java.api.sdk to add it to the list on the right:



click finish. You should have a new project called notes.java.api.sdk on your workspace.

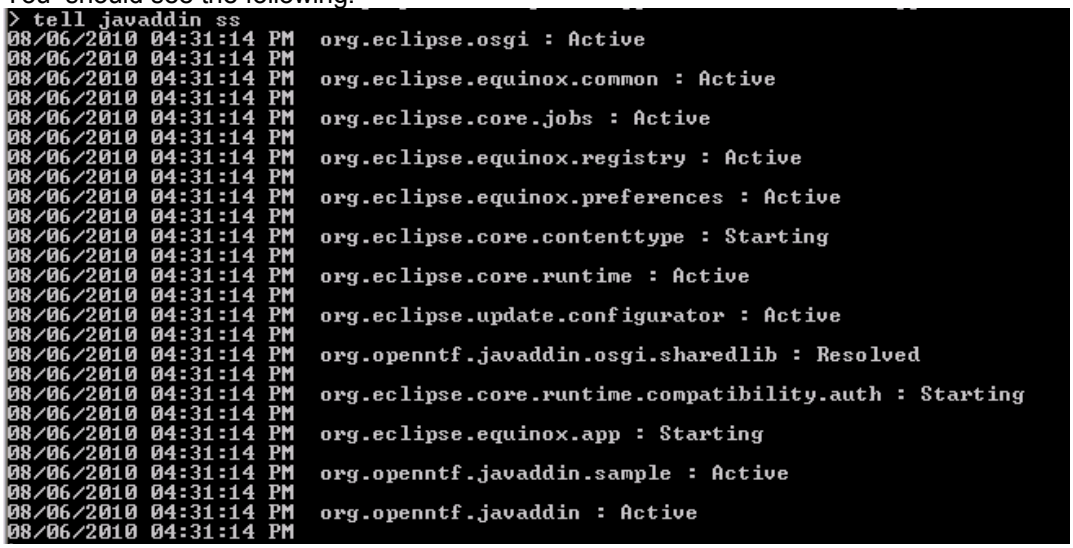
3. Click OK. You should now compile the sample project without error. The next step is to deploy the project as a jar (using file\export deployable plugins and fragments). Deploy the jar in {dominodata}/domino\workspace-javaddin\applications\eclipse\plugins: Use file\export and select “Plugin development/deployable plug-ins and fragments”



click next, select the plugin you want to deploy. Use the “directory” radio button and specify “c:/{dominobin}/javaddin/shared/eclipse” as the directory. (Note: the export dialog box will automatically add the plugins directory so you don't have to specify it.

4. At this point, the sample plugin is deployed, you must restart the javaddin task to be loaded again.
5. You can verify that it is loaded correctly by using the osgi console.

In the domino server console, simply type:
tell dots ss
You should see the following:



If all goes well, you should see the org.openntf.javaddin.sample plugin in the list.
Type the following command: tell dots diag org.openntf.javaddin.sample

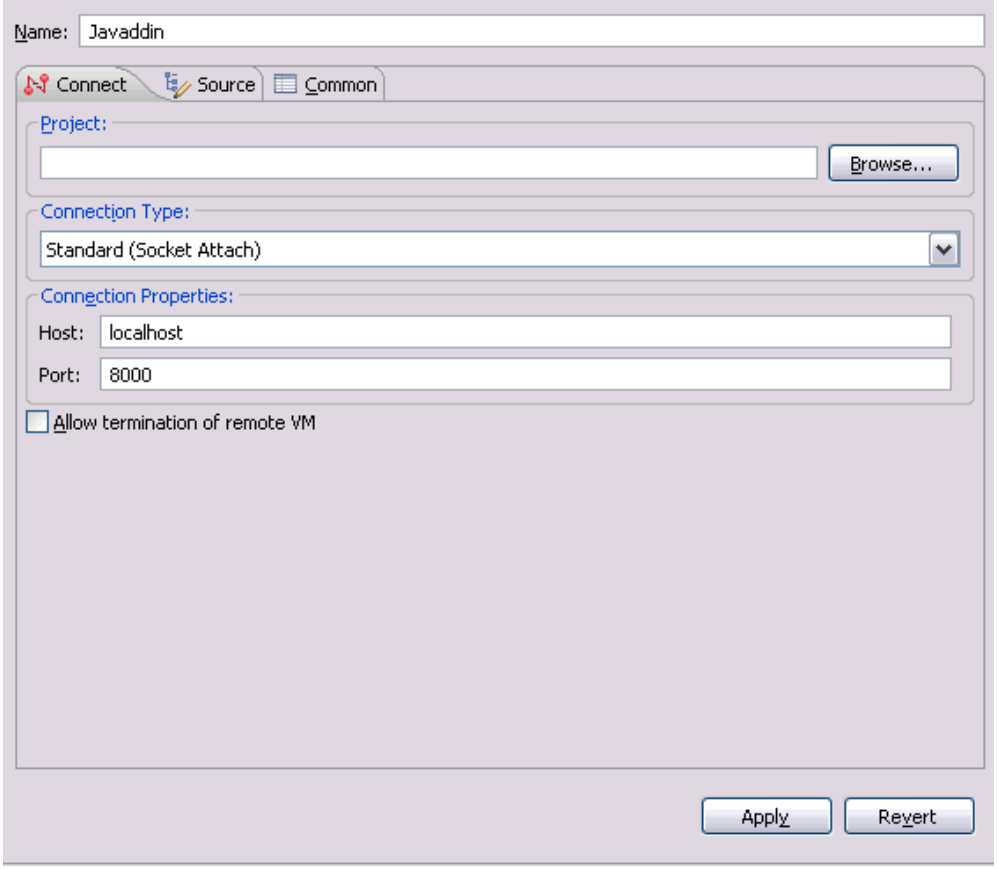
Debugging your server task plugin

Start the jvm in debug mode by using the following parameters:
load dots -debugaddress=<port> -debugsuspend=<y/n>

-debugaddress: connection port for the java debugger (usually 8000)
-debugsuspend: if y, then the jvm is start in suspend mode and resumes only when a debugger attaches to it. This is useful to debug starting code. Default is n

New in 1.3: you can use notes.ini variable to automatically set debugaddress and debugsuspend as follow:
JAVADDIN_DEBUGADDRESS=8001
JAVADDIN_DEBUGSUSPEND=y

In eclipse dev environment, create a debug configuration (use run\debug configurations... menu) for javaddin using the same port used to start the jvm in debug mode, i.e. if port is 8000:

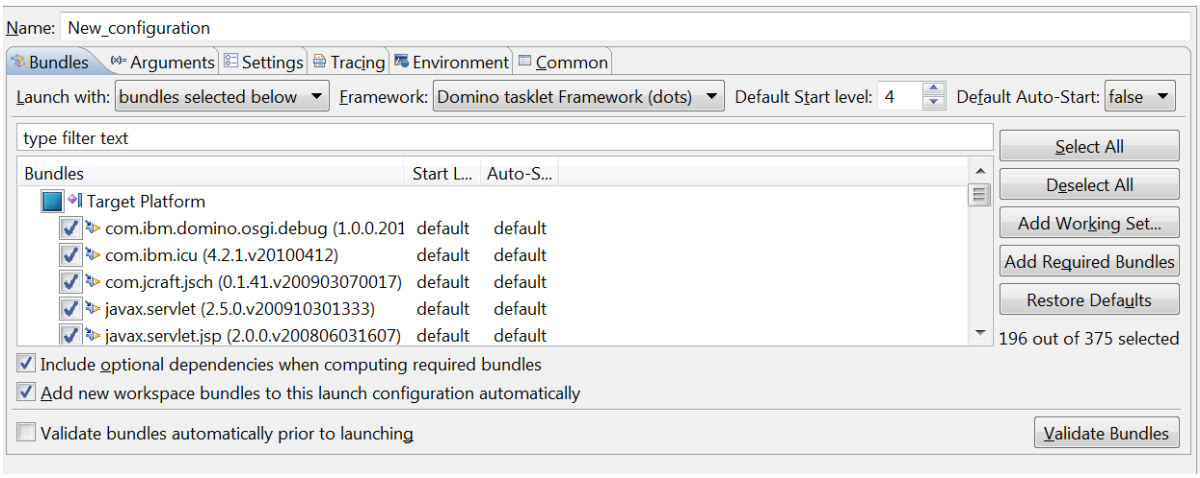


Once the debugger is attached, you can set a breakpoint in your server task and run it manually on the server using: tell dots run <taskid>.

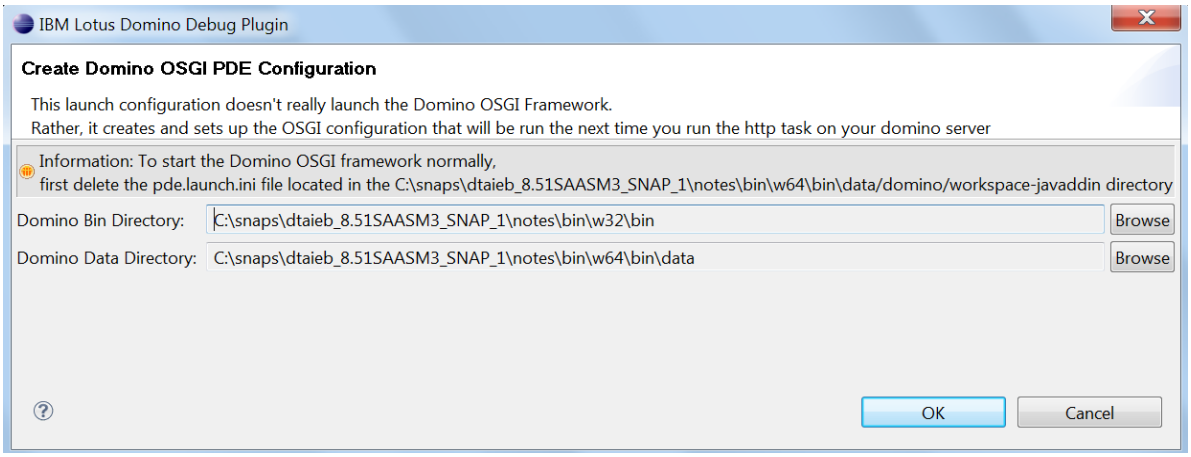
Using the IBM Lotus Domino Debug plugin

This tool, available as a tool project on openntf.org, removes the need for exporting and deploying the custom plugins by configuring the Javaddin runtime to load the plugin classes directly from the Eclipse workspace. To install the tool on Eclipse IDE (3.4, 3.5 or 3.6), please consult the documentation available in the openntf tool project.

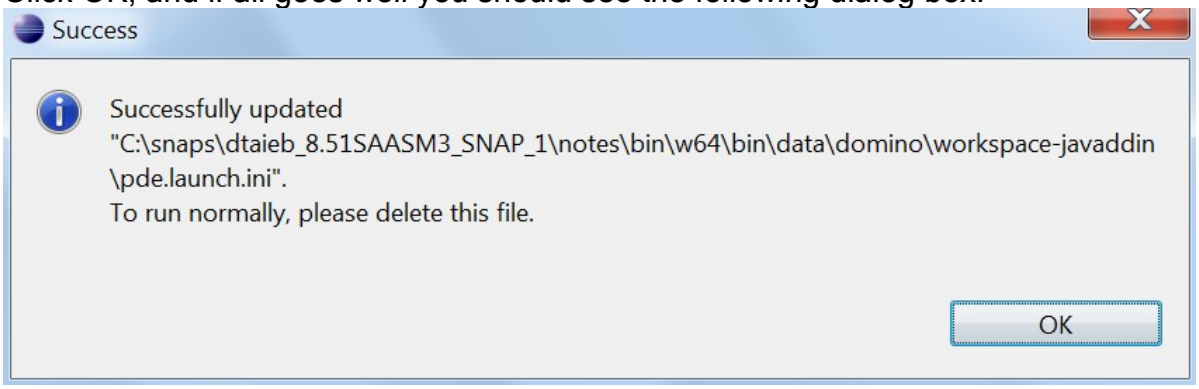
1. Create an OSGi debug configuration for Domino: use menu run\debug configuration, right click on OSGi Configuration entry and select new
2. Use the following options:
   -Framework combo box: select “Domino Tasklet Framework (dots)”
   -Default Auto-Start: select false
3. From the Workspace plugins select the plugins you want to enable. It is recommended to de-select all the plugins in the target platform, the tool will automatically add them from the Domino Install



4. Click debug, you should see the following dialog box that lets you configure where the Domino Server is installed.



5. Click OK, and if all goes well you should see the following dialog box.



6. Next, you need to verify that the plugins are correctly loaded in the Dots framework runtime. In the domino server, restart the dots task:
   >restart task dots
   >tell dots ss : You should see a list of all the plugins loaded in the runtime
7. Assuming that the plugin com.acme.myplugin is a plugin you wanted to load from your workspace, use the following command to verify that it is the case
>tell dots diag com.acme.myplugin

Caveats:

1. It is very likely that it will not work on IBM i platforms as the JRE is provided by the OS and the code currently assumes that it is located under {dominobin} directory

Possible future enhancements:

1. ~~Allow multiple instances of javaddin (done in 1.4 using OSGi profiles)~~
2. Allow plugin to provide credentials for the domino session used to run the task
3. Ability to programmatically change the schedule of a task
4. ~~Possible OSGi Profile Enhancements:~~
    1. ~~Support NSF Based features/plugins in site element.~~
    2. ~~Specify selected features in the site element~~
    3. ~~Add security to site elements. This will become particularly important when NSF Based plugins are supported~~
5. Add Remote OSGi service as a type of remote commands controller