# Running Java agents in Domino

*by Russ Lipton*

*[Editor's note: This article resides in "Iris Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]*

*Software agents play an increasingly useful role within Domino applications. You can think of agents as event-based or triggered programs that are composed from pre-defined simple actions, @commands or LotusScript, and then scheduled for execution either at the client or server level. This article describes how support for Java agents is now integrated within Domino.*

## What is a Java agent?

Simply stated, a Java agent is a self-contained Java executable program of any size, located and placed under the control of a specific Domino server. Since Domino does not currently provide native Java development tools, you can develop the agent with any standard Java tool set. Then, within the same Notes user interface that you use to select and schedule other agents, you can specify and assemble the Java classes required to execute the Java agent.

Java agents, as described here, are developed explicitly for execution within Domino. They run with a Notes-supplied Java runtime, and within a Notes-supplied context. These agents normally do not have a user interface, but rely on the Notes user interface in the same way that other Notes agents do. They can access Notes databases directly, using Java Notes classes. Take a fresh look at Bob Balaban's article on "The Future of Notes and Java," published in *Iris Today* earlier this year, to better understand the relationship between Java and Notes. The *Java Programmer's Guide* shipped with Domino 4.6 is also a useful source of information on developing Java in Notes.

By contrast, Java applets, which can also be hosted within Domino, are generally written to be served up by any standard Web server. While Notes-supported Java applets do run under the Java run-time, they also run under browser-supplied runtimes. That is, a Java applet relies on Netscape Navigator or Internet Explorer to handle all client rendering and display. The context for locating, specifying and running an applet is defined in part by the browser and in part by the codebase parameter that is specified as part of an applet tag. Consequently, a Java applet hosted under Domino behaves exactly as it would when hosted by any Web server.

Finally, while Java agents gain direct access to Notes databases -- assuming security rights have been granted -- Java applets, like any Web-standard component, rely on URLs for access to documents. Of course, Java agents can also use URLs, which may be more convenient or efficient in a given situation.

## Similarities between Java agents and LotusScript agents

The most significant difference between Java agents and LotusScript agents is that the latter are coded directly within the Notes integrated development environment, while the former, as already stated, are designed within a Java-specific environment (for instance, Symantec Cafe or Microsoft J++). Otherwise, Java agents and LotusScript agents are very similar. Both types of agents are executable programs that can carry out actions of arbitrary complexity, including the calling of other agents, before passing control back to the user at the interface level. They also:

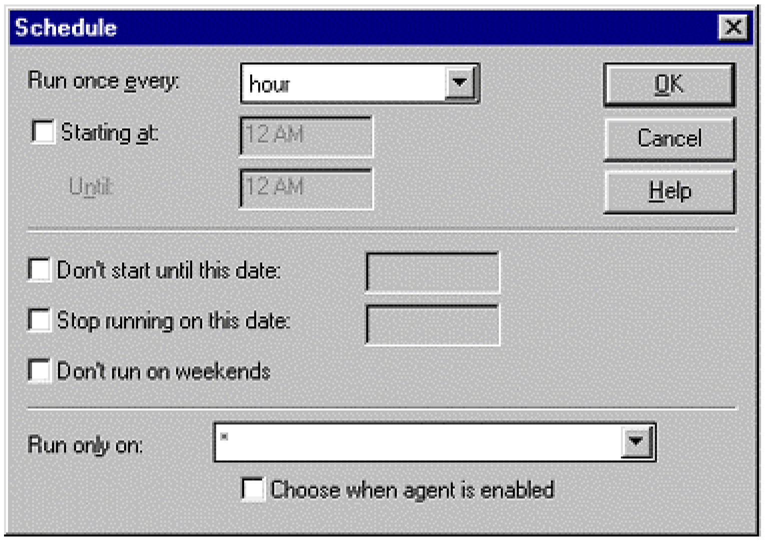
- Employ the same Notes user interface paradigm for their creation and management. (For more information, see the recipe section below.)
- Pass parameters via fields within documents. (Note that Java applets, by contrast, pass parameters within the header that launches the applet itself.)
- Enable users to specify public sharing of agents.

## Scheduling agents

You can schedule the new Java agents in exactly the same way that you would schedule any Notes agent. With Release 4.6, you can now schedule agents to run on multiple servers.

Previously, a specific, named server executed a specific, named Notes agent. The Agent Manager service scheduled agents for execution when the agent's "machine name" matched the name of the current server. This had the virtue of greatly reducing potential replication conflicts between agents hosted on multiple servers.

However, because some developers expressed a need for hosting single agents on multiple servers, you can now specify "*" as the server name when you schedule an agent.



Run once every: hour
☐ Starting at: 12 AM
Until: 12 AM
☐ Don't start until this date:
☐ Stop running on this date:
☐ Don't run on weekends
Run only on: *
☐ Choose when agent is enabled

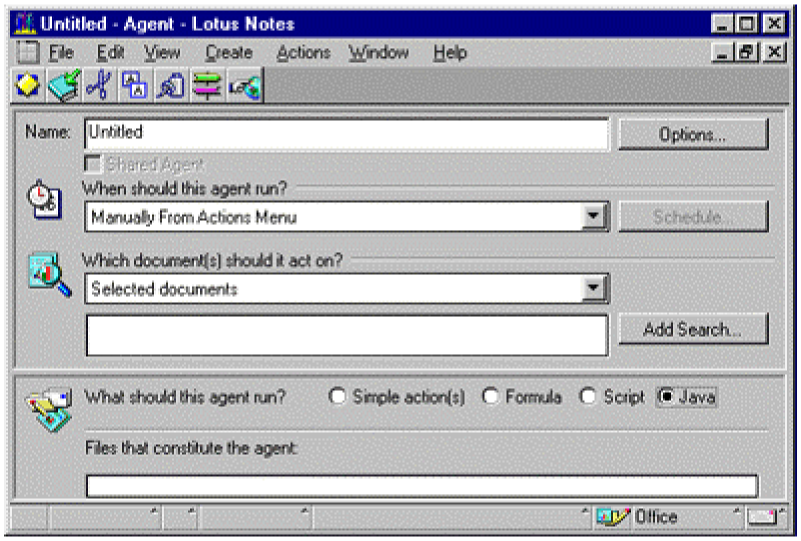
This eliminates the need to enable background agents on every server and the server name check when the $MachineName item contains this symbol. If you take advantage of this, consider the type of document modification activity assigned to such agents on each individual server to anticipate possible replication effects.

## Recipe: Implementing Java agents within Notes

Ideally, your Java agents should already be developed, compiled, and successfully tested before you import them into Notes. Once this is complete, the integration process itself is very simple. It consists of selecting Java as the agent type, specifying the base class of the agent, and importing the class files for that agent. Notes also makes it convenient to update agents with new files and/or to reorganize or delete obsolete class files.

### First, select Java as the agent type

The programmer's pane has been slightly reorganized for this release. The Agent Window's "programming pane" now allows you to supply Java classes (in addition to LotusScript, simple actions and formulas) as the "code" for execution when the agent is run. When you select Java, you are presented with a dialog requesting the needed files for the agent.



Name: Untitled

☐ Shared Agent

When should this agent run? Manually From Actions Menu

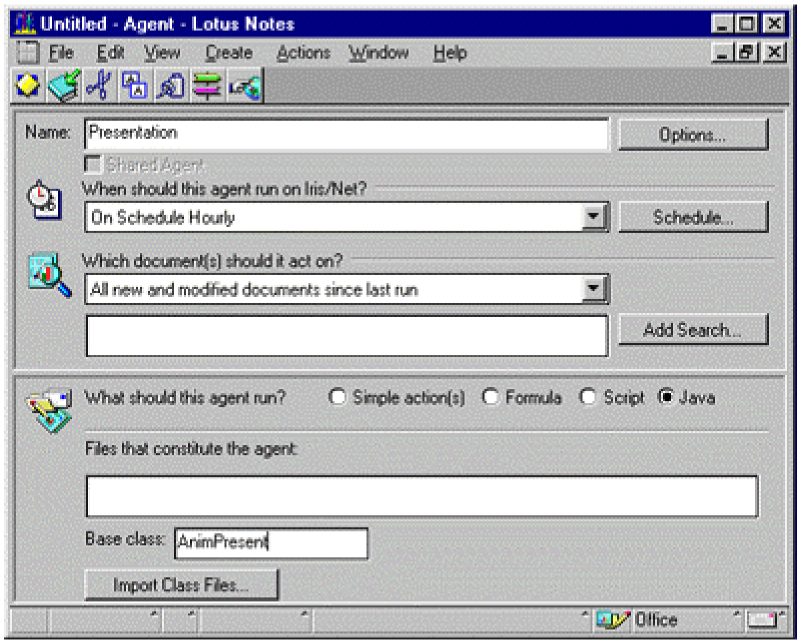
Which document(s) should it act on? Selected documents

What should this agent run? ○ Simple action(s) ○ Formula ○ Script [✓ Java]

Files that constitute the agent:

## Second, specify the base class

When you have selected the Java agent radio button, the next step is to specify the base class for the agent. In Java, the base class is the one that will execute when the agent is launched. Here, we are defining a Java agent ("Presentation") that will manipulate documents in an online Web presentation. We have entered the name of the base class ("AnimPresent.Class") in its field. Notes will assume the .class extent for the class named.



Name: Presentation

☐ Shared Agent

When should this agent run on Iris/Net? On Schedule Hourly

Which document(s) should it act on? All new and modified documents since last run

What should this agent run? ○ Simple action(s) ○ Formula ○ Script [✓ Java]

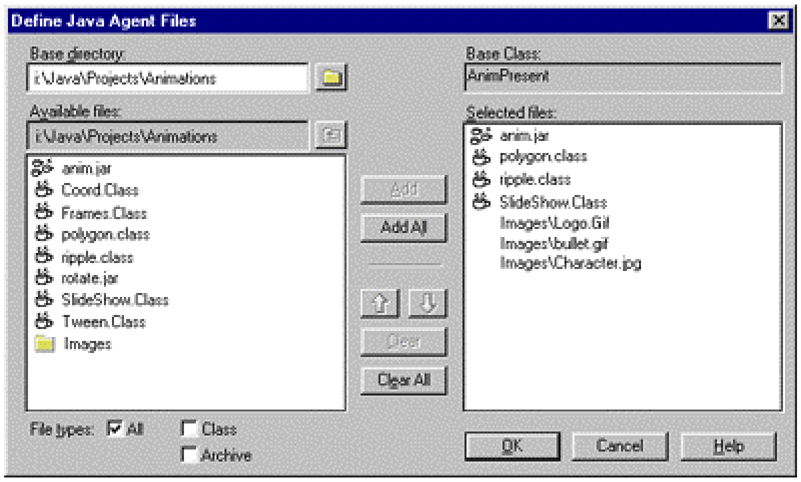
Files that constitute the agent:

Base class: AnimPresent

If the base class field is empty, it is populated with the name of the first class in the list (the first file with a .class extent).

**Third, define the agent files needed.**

Clicking "Import Class Files" displays a dialog box almost identical to the one also used to locate Java applet files:



Base directory: i:\Java\Projects\Animations

Base Class: AnimPresent

Available files: i:\Java\Projects\Animations

- anim.jar
- Coord.Class
- Frames.Class
- polygon.class
- ripple.class
- rotate.jar
- SlideShow.Class
- Tween.Class
- Images

Selected files:

- anim.jar
- polygon.class
- ripple.class
- SlideShow.Class
- Images\Logo.Gif
- Images\bullet.gif
- Images\Character.jpg

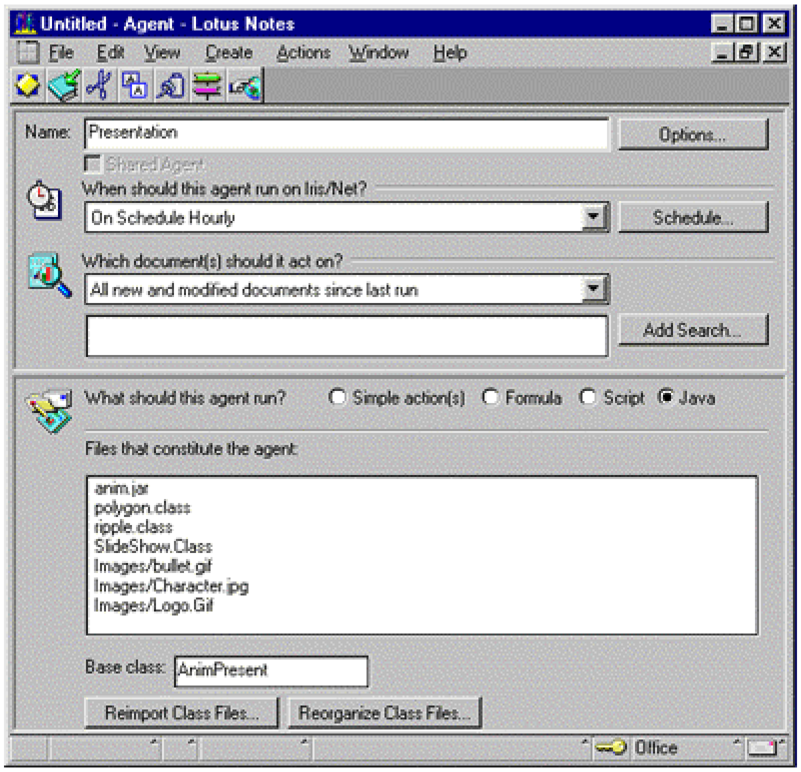
File types: [✓ All] Class Archive

Follow these steps to locate the needed files:

1. Specify the base directory relative to which all of the needed Java classes will be referenced (here, "i:\Java\Projects\Animations"). You may select either Class-type, Archive-type or "All". Here, the panel shows a collection of archive (.jar) and class (.class) files as well as a folder containing graphic images for the applet.
2. Select the individual class (.class), archive (.jar) or source (.java) files that constitute the agent in the left panel and click Add to add them to the panel on the right. Clicking "Add All" moves every file on the left to the right. Their paths will be relative to the base directory. You can re-order files in the selected list with the arrows, or remove them with either the Clear or Clear All buttons.
3. When all needed files for that agent have been selected, click OK.

   Notes now imports all the needed files into the current database. The displayed files are not directly editable, but they can be re-imported (see below). As mentioned before, you can attach the Java source files (.java) as well for housekeeping purposes, but Notes does not do anything directly with these files.

**Four, re-import files for Java agents.**
When a Java agent is first created, the "Import" dialog is automatically displayed. However, once you have named and saved a Java agent, the screen below displays the files that have been imported, and offers you the opportunity to "re-import" or "reorganize" an existing agent.


Untitled - Agent - Lotus Notes
File Edit View Create Actions Window Help
Options...
Schedule...
Add Search...
Reimport Class Files...
Reorganize Class Files...
Office

Name: Presentation

Shared Agent

When should this agent run on Iris/Net? On Schedule Hourly

Which document(s) should it act on? All new and modified documents since last run

What should this agent run? Simple action(s) Formula Script [✓ Java]

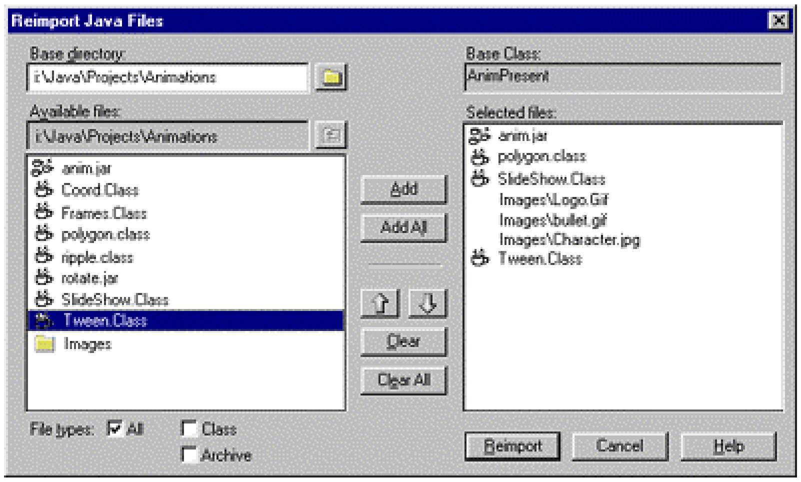
Files that constitute the agent:

anim.jar
polygon.class
ripple.class
SlideShow.Class
Images/bullet.gif
Images/Character.jpg
Images/Logo.Gif

Base class: AnimPresent

The process for modifying Java agents is very similar to the one for modifying Java applets:

1. Click "Reimport Class Files" to bring up the file dialog. You may selectively replace any or all files shown in the list with more current copies of the same classes. Or, you may add new classes. Here, we have removed the "ripple.class" and added the "tween.class."



Base directory: i:\Java\Projects\Animations

Base Class: AnimPresent

Available files: i:\Java\Projects\Animations

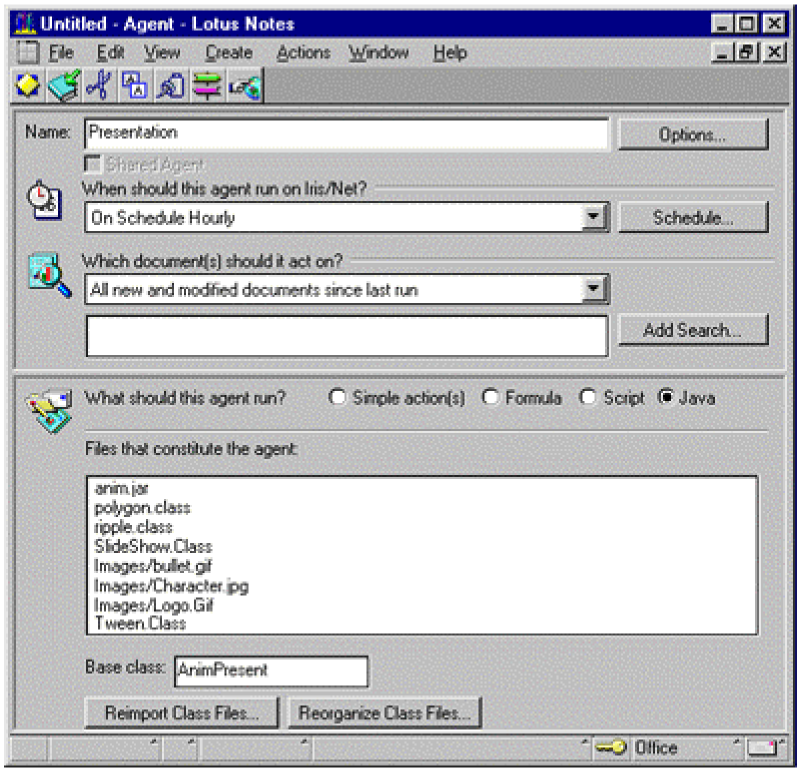
anim.jar
Coord.Class
Frames.Class
polygon.class
ripple.class
rotate.jar
SlideShow.Class
Tween.Class
Images

Selected files:

anim.jar
polygon.class
SlideShow.Class
Images\Logo.Gif
Images\bullet.gif
Images\Character.jpg
Tween.Class

File types: [✓ All] Class Archive

When you click OK, the new agent files list is displayed in the design pane:


Untitled - Agent - Lotus Notes
File Edit View Create Actions Window Help
Options...
Shared Agent
Schedule...
Add Search...
Reimport Class Files...
Reorganize Class Files...
Office

Name: Presentation

When should this agent run on Iris/Net? On Schedule Hourly

Which document(s) should it act on? All new and modified documents since last run

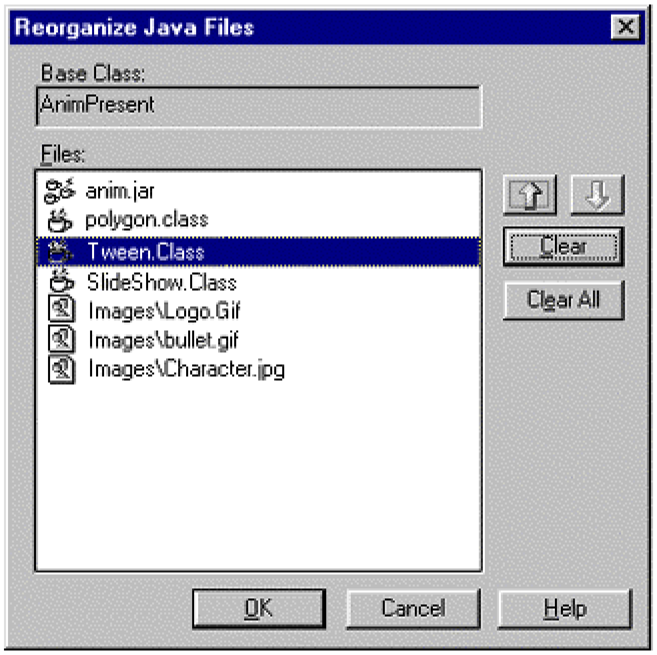
What should this agent run? ○ Simple action(s) ○ Formula ○ Script [✓ ● Java]

Files that constitute the agent:

```
anim.jar
polygon.class
ripple.class
SlideShow.Class
Images/bullet.gif
Images/Character.jpg
Images/Logo.Gif
Tween.Class
```

Base class: AnimPresent

2. Click "Reorganize Class Files" to delete and/or reorder files in the list. The order is important, since the Java Virtual Machine searches for class files in the order they are specified. If a class file invokes another file at run-time that has not already been located, an error will result. Here, we have used the "up arrow" icon to promote the "tween.class" between the "polygon.class" and the "SlideShow.class."

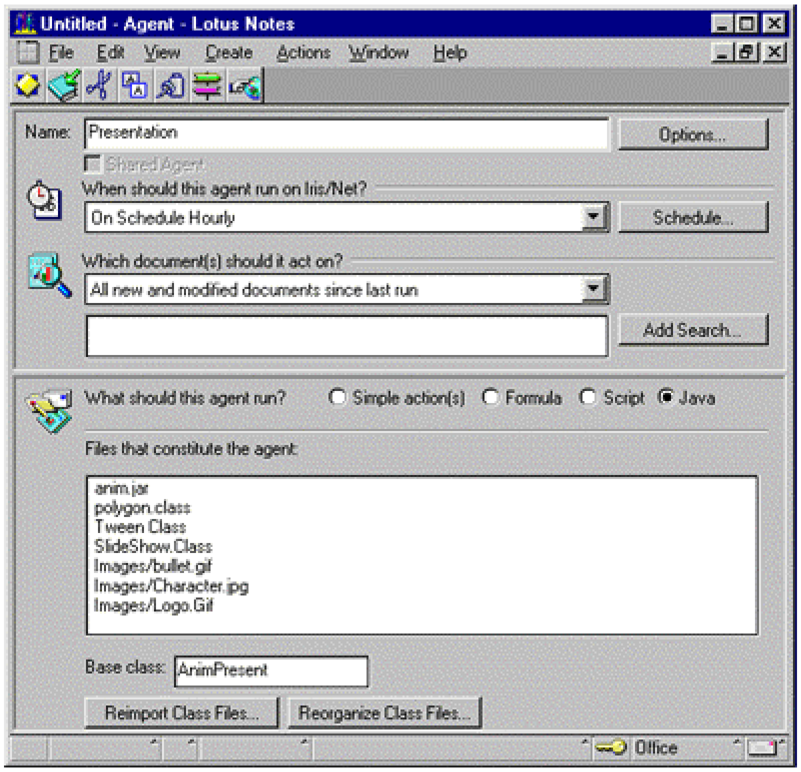


Base Class: AnimPresent

Files:

```
anim.jar
polygon.class
Tween.Class
SlideShow.Class
Images\Logo.Gif
Images\bullet.gif
Images\Character.jpg
```

Again, when you click OK, the reorganized agent files list will be displayed in the design pane:


Untitled - Agent - Lotus Notes
File Edit View Create Actions Window Help
Options...
Schedule...
Add Search...
Reimport Class Files...
Reorganize Class Files...
Office

Name: Presentation

Shared Agent

When should this agent run on Iris/Net?

On Schedule Hourly

Which document(s) should it act on?

All new and modified documents since last run

What should this agent run? ( ) Simple action(s) ( ) Formula ( ) Script [✓ Java]

Files that constitute the agent:

```
anim.jar
polygon.class
Tween.Class
SlideShow.Class
Images/bullet.gif
Images/Character.jpg
Images/Logo.Gif
```

Base class: AnimPresent

## Testing and executing agents

Because a Java agent is "just" a regular Java application created in a Java development environment, you should test all Java agents *before* importing them into Notes. Once you've imported the agents, test their run-time behavior by scheduling and executing them as you would any Notes agent. Notes executes a Java-standard application (or "agent") normally.

If you experience problems within Notes, you should write and test the Java agent again within a Java development environment before re-importing the files into Notes. Although Notes does not currently provide direct debugging of Java agents, you might want to use *system.out.println* lines in your Java code, so helpful debugging information can be output to either the Notes server console or the Java console on the client during execution. Error information is also posted to the agent's log file.

Assuming that you tested a given agent thoroughly before importing it, make sure that you imported all of the needed files and placed them into the correct order. To check for this, simply re-import the agent just as you would if you were updating it.