by
Julie
Kadashevich

Iris Today Archives

Level: Advanced
Works with: Designer 4.6
Updated: 03/02/98

When you've designed a great agent, what do you do when it won't run? In my previous articles for *Iris Today*, we talked about the factors that affect who can run agents in "**Controlling the agents in your system**," and when the agents run in "**Minimizing delays in the Agent Manager**." This time, we'll look at the specific steps you can take to troubleshoot agents to find out *why* your agents won't run.

This article will first introduce you to the different tools available for troubleshooting agents, including Notes.ini settings, the LotusScript NotesLog class, and server console commands. Then, we'll look at some common problems you may run into when developing agents, and explain the possible causes and solutions for each situation.

Note that this article describes troubleshooting agents in Notes/Domino 4.6. For information on troubleshooting agents in R5 and Notes/Domino 6, see the *LDD Today* article "**Troubleshooting agents in Notes/Domino 5 and 6**."

Some background

Before we begin examining the tools for troubleshooting, we should review a few points about agents and the Agent Manager. Whenever you find that your agent won't run, you should first look at the Agent Log. The Agent Log will show you the last time the agent executed, and whether it completed its execution. For more detailed information, you should examine the server console or Notes Log (under Miscellaneous events) for any messages from the Agent Manager. If you don't have physical access to the actual server console, you might have access to the live console by choosing File - Tools - Server Administration and clicking the Console button. If you don't have access to the server console, you will not be able to issue the commands that we will cover later in this article, but you will be able to see the output generated by the Agent Manager in the Notes Log.

As you may know, you can easily test agents by choosing Actions - Test, and view the results in the Test Run Agent Log. And, if you're using LotusScript, you can use the LotusScript debugger by choosing File - Tools - Debug LotusScript. However, these tests only work for agents running in the foreground. As you may remember from my previous articles about the Agent Manager, agents that run in the background do not behave the same as agents that run in the foreground. For one thing, agent security (that is, agent restrictions and ACL rights) works differently when an agent runs in the foreground, compared to when it runs in the background. Also, the UI classes are not supported in the background. So, we'll also look at how you can use the NotesLog class to debug background agents.

Debugging with Notes.ini settings

When you discover that your agent won't run, the first thing you can do is to modify your Notes.ini file to turn on Agent Manager debugging. To do this, you simply add the following line to your server's Notes.ini file.

Debug_AMgr = *flag*

where *flag* can be one or more of the following: (listed in alphabetical order)

c - to output agent control parameters

e - to output information about Agent Manager events
l - to output agent loading reports
m - to output agent memory warnings
p - to output agent performance statistics
r - to output agent execution reports
s - to output information about Agent Manager scheduling
v - verbose mode, which outputs more messages about agent loading, scheduling, and queues
* - to output all of the above information (same as turning on all the flags)

The output appears in the console log and the Notes Log. I usually run with the setting "Debug_Amgr=*", but it may generate more output that you are interested in. Also, be aware that having all debugging flags turned on has approximately 5% performance cost on the average user response time.

You can also turn on agent execution logging by adding the following line to the Notes.ini file. (You can also do this in the server's Server Configuration document in the Public Address Book.)

Log_AgentManager = *value*

where *value* can be one of the following:

0 - do not show logging
1 - to show partial and complete successes
2 - to show complete successes

The Log_AgentManager setting provides you with a subset of the debugging information that Debug_AMgr generates. This option provides less output, but it has a smaller impact on performance. Some people keep the Log_AgentManager setting turned on even when there are no problems, just to have additional information in the log. If you have both Notes.ini variables specified, Debug_AMgr settings will take precedence.

Background agents (by definition) cannot generate output to UI. All output generated by the background agent (for example, print statements) goes to the server console (and to the Notes Log under Miscellaneous events). The same is true of the error and warning messages generated by the Agent Manager on behalf of the agent. So, it is important to always examine the server console or Notes Log for information from the Agent Manager.

You also can output the server console messages to a text file by adding the DEBUG_OUTFILE setting to your server's Notes.ini file. For example, DEBUG_OUTFILE="C:\mydebug.txt". I find that sometimes it is easier to search this file rather than the entries in the Notes Log or the server console. Of course, the extra I/O is costly in terms of performance, and you should use this technique judiciously.

If you are running locally scheduled agents, the debugging information does not appear on the server console, because the Agent Manager is running locally rather than on the server. In this case, you can output the information to a text file by adding the DEBUG_OUTFILE setting to your client's Notes.ini file. Then the file will be created locally on your workstation.

Debugging using the NotesLog class

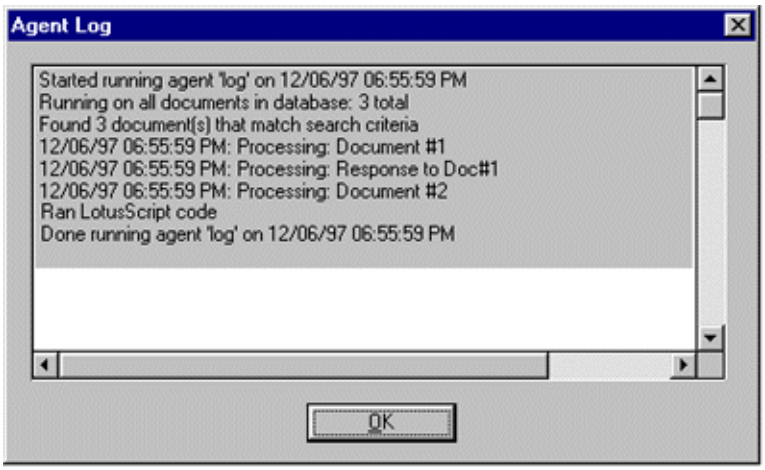
To debug LotusScript and Java agents that run in the background, you can use the NotesLog class to output information to the Agent Log. The Agent Log is useful for error handling and checking variable values, as well as checking the logic of your agent. For example, you can see whether the Search Builder is picking up the documents you expected the

agent to find.

Here is an example of a LotusScript agent that processes all unprocessed documents and then puts the title of each document into the Agent Log. The four lines of code you need to add to your agent to generate an Agent Log are highlighted in bold.

```
Dim agentLog As New NotesLog("Agent log")
    Call agentLog.OpenAgentLog
    Set s = New NotesSession
    Set db = s.CurrentDatabase
    Set collection = db.UnprocessedDocuments
    Set note = collection.GetFirstDocument
    count = collection.Count
    Do While (count > 0)
        Subject = note.Subject
        Call agentLog.LogAction( "Processing: "+Subject(0))
        Set note = collection.GetNextDocument (note)
        count = count - 1
    Loop
    Call agentLog.Close
```

To find the generated log, select your agent, and choose Agent - Log (or right-click and select Log). Here is what the Agent Log generated by my sample agent looks like.



Instead of logging errors to the Agent Log, you might prefer to write them to a text file. Here is an example of an error handler that writes the errors to a file.

```
  On Error DivisionByZero Goto LogError
    Dim errorLog As New NotesLog( "Errors" )
    Dim x As Integer, y As Integer, z As Integer
    Call errorLog.OpenFileLog( "c:\errlog.txt" )
    y = 13
    z = 0
    rem The following line will generate an error for this example
    x = y / z
    Call errorLog.Close
    Exit Sub
LogError:
    Call errorLog.LogError( ErrDivisionByZero, Error$( ErrDivisionByZero )
)
```

Resume Next

Debugging at the server console

The Agent Manager supports the following server commands that are useful for troubleshooting:

- Tell amgr schedule
- Tell amgr status
- Tell amgr debug

Tell amgr schedule

The "Tell amgr schedule" command shows the Agent Manager schedule of all the agents scheduled to run for the current day. This is useful for debugging purposes, because you can see if your agent is waiting in one of the Agent Manager queues.

The Agent Manager has three different queues: a queue for agents that are eligible to run (E), another queue for agents that are scheduled to run (S), and a third for event-triggered agents that are waiting for their event to occur (V). Scheduled agents (the ones that have any of the schedule triggers) are queued in the "Scheduled" queue. When the time they are scheduled to run arrives, they are moved into the "Eligible to run" queue. Event-triggered agents (new mail and document creation/modification agents) are queued in the "Event" queue until an event they are waiting for occurs. When the event occurs, the agents move into the "Scheduled" queue and then into the "Eligible to run" queue.

Here is a sample output from the "Tell amgr schedule" command:

```
E  S  04:03 PM Today      agent1  CENTRAL.NSF
S  S  05:04 PM Today      agent2  CENTRAL.NSF
V  U                 agent3  CENTRAL.NSF
```

The first column of the output contains the Agent Manager queue type. The second column contains the agent trigger type, where *S* means the agent is scheduled, *M* represents a new mail-triggered agent, and *U* represents a new/updated document-triggered agent. These columns are followed by the time the agent is scheduled to run, the name of the agent, and the database name.

In the sample output above, agent1 is a scheduled agent that has been moved to the "Eligible to run" queue, and it will run as soon as the Agent Manager is free to run it. Agent2 is a scheduled agent that is waiting in the "Scheduled" queue for its scheduled time to arrive. Agent3 is an event-triggered agent waiting in the "Event" queue for a document to be created or modified.

Note that the agents take about a minute or so to appear in queues, or to move from one queue to another. For information on the different factors that control scheduling, see the "Minimizing delays in the Agent Manager" article.

Tell amgr status

The "Tell amgr status" command shows a snap shot of the Agent Manager status. You can see the status of the various Agent Manager queues and control parameters. This information is useful for reviewing all the parameters that are currently in effect.

Here's the sample output:

```
12/26/97 10:30:15 AM  AMgr: Status report at '12/26/97 10:30:15 AM'
12/26/97 10:30:15 AM     Agent Manager has been running since
```

'12/22/97 02:18:25 PM'
12/26/97 10:30:15 AM There are currently '1' Agent Executives running
12/26/97 10:30:15 AM There are currently '4' agents in the Scheduled Task Queue
12/26/97 10:30:15 AM There are currently '0' agents in the Eligible Queue
12/26/97 10:30:15 AM There are currently '0' databases containing agents triggered by new mail
12/26/97 10:30:15 AM There are currently '0' agents in the New Mail Event Queue
12/26/97 10:30:15 AM There are currently '3' databases containing agents triggered by document updates
12/26/97 10:30:15 AM There are currently '3' agents in the Document Update Event Queue
12/26/97 10:30:15 AM AMgr: Current control parameters in effect:
12/26/97 10:30:15 AM AMgr: Daily agent cache refresh is performed at '12:00:00 AM'
12/26/97 10:30:15 AM AMgr: Currently in Daytime period
12/26/97 10:30:15 AM AMgr: The maximum number of concurrently executing agents is '1'
12/26/97 10:30:15 AM AMgr: The maximum number of minutes a LotusScript/Java agent is allowed to run is '10'
12/26/97 10:30:15 AM AMgr: The maximum percentage of time agents are allowed to execute is '90'
12/26/97 10:30:15 AM AMgr: Executive '1', total agent runs: 414
12/26/97 10:30:15 AM AMgr: Executive '1', total elapsed run time: 273

Tell amgr debug

You can use the "Tell amgr debug" command to either display the current debug settings for the Agent Manager, or to set new ones. When using this command to set debug values, you can use the same flags as shown earlier for the Notes.ini setting Debug_AMgr. These settings take effect immediately; you do not need to restart the Agent Manager or the server.

Here is a sample output:

>tell amgr debug
02/12/98 02:03:15 PM AMgr: Current debug control setting is 'mecvrspl'

Common problems

Now that you understand the debugging tools that are available, let's look at the common problems you may run into when developing agents. And, more importantly, let's look at the solutions for those problems.

Deploying on different servers

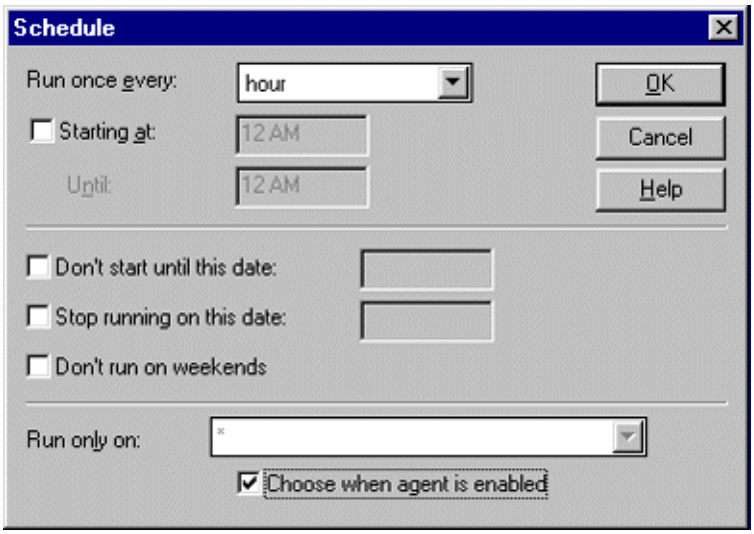
If you develop an agent and then need to deploy it on many different systems, you will find yourself in a situation where:

- The server names on which you need to deploy are different from the name of your server. In addition, you may not know the server names in advance.
- The agent developer's signature is different from the end-user's signature.
- The development server does not have a certificate in common with the production server.

We will look at several approaches to solving the first two issues, which in turn, will make the certification issue irrelevant.

Changing the server name
When you create an agent, the name of the server on which the agent is supposed to run is stored in the agent. By default, the server name is the server on which you're developing the agent. In order for the agent to run on a different server, the server name needs to change. There are three ways to do this. The first approach is to deploy your agent as disabled, click the Schedule button in the Agent Builder, and select the option "Choose when agent is enabled." When the user first enables the agent, a prompt appears from which the user can choose a server name for running the agent.

The second approach (which requires Release 4.6 or later) is to specify that the agent can run on any server. To do this, you can click the Schedule button in the Agent Builder, and enter a wildcard "*" as the server name in the "Run only on" field. Note that if you have replication set up between different servers and the agent modifies the same documents on those servers, you may end up with replication conflicts. The screen shot below illustrates both of these options.



The third approach is to write an agent that programmatically sets the server name of another agent. To do this, you can use the following code fragment:

```
agent.ServerName="ServerName"
Call agent.save()
```

where *ServerName* is the new server name, which you can read from a database, or obtain from the end user. You need to save the agent in order to update the server name. For security reasons, this method is only supported for agents that run in the foreground.

Signing the agent
When you develop an agent, your signature is stored in the agent. You need the user's signature to replace yours in the agent, so the agent can run in the background with the user's rights. There are two ways to do this. Both approaches are based on the fact that enabling an agent re-signs that agent with the signature of the person who enabled it. In addition, both approaches require that you deploy the agent as disabled. The first approach relies on the end user to manually enable the agent by clicking on the enable checkbox.

The second approach is to enable the agents programmatically. This

approach is more appropriate if you need to deploy a large number of agents. In that case, you may prefer to write an agent that enables other agents. The end user will run this agent, and it will enable all the other agents. Here is the code that you need to include in your agent in order to programmatically enable other agents.

```
agent.IsEnabled = True
Call agent.save()
```

You need to save the agent in order to update the IsEnabled property. For security reasons, the IsEnabled method is only supported for agents in the foreground. Note that if you enable the agent programmatically, you cannot also prompt the user to select what server to run on when the agent is enabled. If you try to do both, the agent will never run.

As I mentioned before, enabling and setting the server name methods are only valid in the foreground. If your background agent attempts to either set the server name or enable the agent, the following error message will be generated:

02/04/98 05:14:35 PM AMgr: Agent ('DoEnable' in 'test1.nsf') error message: You must have permission to sign documents for server based agents

My agent runs from the UI, but not as a scheduled agent
Typically, this is the result of one of two things: either the problem is in the security or you used a UI class in the agent. For information on agent security, see the sidebar "**Agent security at a glance**."

When any of the UI classes are used in a background agent, even if it is just a dim statement for NotesUIWorkspace, the agent will not run in the background. As you may know, back-end classes perform operations on Notes databases, while front-end classes manipulate the UI (NotesUIWorkspace, NotesUIDatabase, NotesUIView, NotesUIDocument). Back-end classes can run anywhere (in the background or foreground, server or workstation), whereas front-end classes can only run in the foreground on the workstation. If you run an agent containing a reference to one of the front-end classes in the background, the UI classes will not be found and the agent will not run. You'll then receive the following error:

"Error loading USE or USELSX module: XXX"

This error is generated to the server console and Notes Log, so if you are not watching either of these, you may miss this clue to the cause of your problem.

My agent containing script libraries runs from the UI, but not as a scheduled agent
This problem generally occurs when you use a script library and do not account for case sensitivity of the server file naming. For example, let's say the script library is called "TestLib." If you specify "Use TestLib" in the agent, the agent will work both if invoked from the workstation (for example, when invoked through the menu) as well as from the server.

But, if the agent instead contains "Use testlib," the agent will work from the workstation, but not from the server. The following error will be generated on the server console and the Notes Log:

12/11/97 01:35:58 PM AMgr: Agent ('test' in 'Test.nsf') error message: Error loading USE or USELSX module: testlib

Background agents stop running
If your agents run for a while and then stop running, and start running normally again if you restart the server, the most common cause of the problem is that the load on the Agent Manager is higher that the "Max% busy before delay" parameter allows. The "Max% busy before delay" setting is defined in the Agent Manager section of the Server document. If you have the Agent Manager debugging for scheduling turned on (AMGR_DEBUG=s or AMGR_DEBUG=*), you will see the following warning:

AMgr: Percentage of time agents allowed to run exceeded, delaying additional agent executions

If you see these warnings on a consistent basis, it is an indication that the "Max% busy before delay" parameter is set too low for the load on your system. You should adjust the settings higher and/or increase the number of concurrent agents that your system is executing. For more information on the "Max% busy before delay" setting, see the "**Minimizing delays in the Agent Manager**" article.

My mail agents do not run
Mail agents are defined to run on the home mail server of the agent owner (not the agent invoker). If the agent is replicated from your home mail server to another system, or if you are trying to run an agent that was written by someone else, then your home mail server will not match the server on which you are attempting to run the mail agent, and the agent will not run. You can suppress the check for the home mail server by adding the following Notes.ini setting (which is supported in Release 4.5 and later):

AMgr_DisableMailLookup =*value*

where *value* can be 0, to check for the home mail server; or 1, to not check for the home mail server. The default is 0.

My formula mail agents are not running
If you create a shared agent that contains formulas and you would like it to be triggered by a new mail trigger, in many cases, the agent will not be triggered. This happens because with the 4.5 (or later) clients, formula agents are saved in the V3 format (to be compatible with prior releases), and they are missing some information that new mail-triggered agents need. To work-around this problem, you can do one of the following:

- Force the formula agent to be saved in V4 format. For example, you can click the Options button in the Agent Builder, select any of the options, and click OK. You will then receive a warning that your agent will be saved in V4 format, and will no longer be editable by V3 systems.
- Call your formula agent from a simple agent.
- Call your formula agent from a LotusScript agent.

My scheduled agents do not run in MAIL.BOX
If scheduled or event-triggered agents are created in the MAIL.BOX database, these agents will never be automatically scheduled to be executed. This happens because MAIL.BOX is an internal database -- it is not treated as a regular database, and the Agent Manager does not scan it for scheduled tasks.

I've assigned execution rights to users, but they're still rejected
As you know, you assign execution rights for users in the Agent Restrictions fields in the Agent Manager section of the Server document.

These fields have a maximum limit of 256 characters. Any names entered beyond 256 characters will not be seen by the server, and those users are rejected as not having the proper execution rights. If you exceed this limit, you will receive a warning while editing the field as well as periodically while the Agent Manager is running. You'll receive the following warning on the server console:

01/12/97 14:05:45 AMgr: Agent execution ACL is longer than 256 characters. Use groups.

To solve this problem, you should switch to using group names in the Agent Restrictions fields.

Something is disabling my agent
Does it seem like something goes around at night and mysteriously disables your agents? If you use agents defined in a template, every time the template design is updated, by default, the agent design will be updated as well. When the agent design is updated, all elements that comprise an agent are updated, which includes the enabled state as well as the signature on the agent. You can suppress the agent design update by selecting the option "Do not allow design refresh/replace to modify" in the agent properties.

I want my server-based agent to run immediately
You can write a foreground agent that invokes a server-based agent using a method RunOnSever in the Agent class. Then, the user can invoke the agent through the UI, which in turn, runs the agent on the server. To do this, you can use the following code fragment:

agent.RunOnServer(),

where *agent* is the agent you want to execute on the server. The call is synchronous, so the workstation waits for the agent to complete before returning. This agent will run on the same server as the database in which it resides. One of the benefits of this method is that if you have an agent that performs database operations, you do not need to have a copy of the database software on the client. This method was introduced as hidden in 4.6, and is unhidden in 4.61.

The security for this type of agent works differently for the portion that runs on the workstation and the portion that runs on the server. They follow their respective rules:

- For the calling agent running on the workstation, the ACL rights are determined by the rights of the invoker and agent restrictions are not used.
- For the called agent running on the server, the ACL rights and the agent restrictions are determined based on the signer of that agent.

For more information on agent security, and how it can affect when your agents run, see the sidebar "**Agent security at a glance**."

Conclusion

In this article we've covered some of the techniques for debugging agents, including Notes.ini settings, the NotesLog class and server console commands. In addition, we've looked at some of the common pitfalls people encounter. I hope you find these techniques and solutions helpful when it comes time to troubleshoot your own agents.

ABOUT THE AUTHOR
Julie Kadashevich came to Iris in March of 1997 after working at FTP Software on Java and C++ mobile agent technology. Previously, she worked in the area of

applied Artificial Intelligence at Wang Labs and received five patents in the field. She has Bachelor's and Master's degrees from Boston University, both in computer science. Outside of the Agent Manager, her two main interests are photography and quilting.