

Agent Log

Why and what?

Introduction to the Agent Log application

Scheduled agents in Lotus Notes are vital – but often fail:

Error types:

1. Agent runs, but fails (e.g. object variable not set, division by zero)
2. Agent manager tries to execute the agent, but fails (e.g. access problem, agent needs recompilation)
3. Agent did run at all (e.g. deactivated or activated on an invalid server)

To secure that the agents always runs, we need to implement logging, error trapping and monitoring. If, for any reason, an agent fails we want to notify the responsible database owner, developer or system administrator.

Type 1 errors can easily be handle in the code with standard IF..THEN..ELSE statements or ON ERROR GOTO statements. Type 2 errors can only be found by scanning then log.nsf on every server and type 3 errors can only be trapped by comparing the agents schedule information with the agent last run date.

The Agent Log gives a simple way to handle all 3 error types:

- Contains a standard script library to include in agents, giving a simple and consistent interface to log errors
- Gives a common replicated log database to collect agent logs from all servers
- Can scan the log.nsf on all servers to find type 2 errors
- Includes functionality to find type 3 errors
- From a setup view, it is easy to setup the notification system – allowing reports to be sent to the database owner, agent signer, admin group or a helpdesk database. Error reports can be escalated to the Notes event handler (can be tracked by GSX or Tivoli)

How?

Quick Developer introduction and a database overview

Developer intro

The agent log database should be replicated to all servers.

Follow these steps to implement logging in an agent:

1. Copy the “cls.log.6”, “cls.fw” and “cls.system” script libraries to your database (the libraries can be copied from the Agent Log database)

2. Write these lines in your Lotus script agent:

```
Use "cls.log.6"  
Use "cls.fw"  
Dim config as new DbConfig()  
Dim l As New AgentLog( config.getDatabase( "LOG" ) ) 'create the log object  
'<your code here>  
Call l.logAction( LOG_NORMAL, "Hello World", Nothing ) 'how to write a message to the log (optional)  
Call l.close() 'close the log
```

That's all. Now the agent will create a log document each time is run. If the agents fails (for any reason) a notification mail will be sent. Use the 'l.logAction(...)' statements to write detailed information about agent execution to the log document.

Note: there is a sample agent (called 'Sample agent') in the Agent Log database

Log database overview

The Agent Log database has 3 main sections:

- Logs
- Agents
- Administration

The screenshot shows the 'NCC Agent Log' interface. It features a navigation pane on the left with sections for 'Logs', 'Agents', and 'Administration'. The main area displays a tree view of log entries under 'NDK01'. The entries are organized into folders such as 'Agent Log 6.3', 'NCC Agent Log', and 'Ncc Agent Log 6.3'. Each folder contains sub-entries like 'Agent Error Reporter' and 'Replication Configurator'. Two specific log entries are visible with dates and times: '02-09-2009 14:05' and '02-09-2009 13:52', both with the message 'Division by 2'.

	Date	Message
NDK01		
Agent Log 6.3		
Agent Error Reporter		
Replication Configurator		
dbConfig 5.2a		
NCC Agent Log		
Agent Error Reporter		
Replication Configurator		
Sample Agent / Niro		
02-09-2009 14:05	14:05	Division by 2
02-09-2009 13:52	13:52	Division by 2
Ncc Agent Log 6.3		
Agent Error Reporter		
Replication Configurator		

Log section

In this section you can see all the log documents created by the enabled agents.

Special views

- Errors: show only logs that has reported an error
- Latest by database: shows only the latest log document (one document per agent)

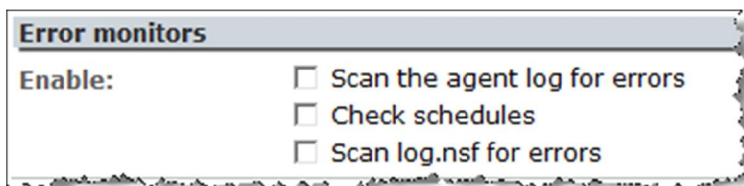
Agent section

In this section you can view all the enabled agents (document will be created first time the agents runs). On the agent document you can

- See the scheduling information for the agent
- See the agent last run time
- Set the logging parameters for the agent (severity levels)
- Disabled the notifications for this agent

Administration section

Only visible if you have the [Admin] role. Click 'Database...' to open the setup profile document – from here you can enable/disable the notification system and specify to whom the reports should be mailed to.



Error monitors	
Enable:	<input type="checkbox"/> Scan the agent log for errors
	<input type="checkbox"/> Check schedules
	<input type="checkbox"/> Scan log.nsf for errors

Developer reference

More developer documentation and samples

The LogAction method

The LogAction method takes 3 parameters:

- **Severity.** Severity can be: LOG_FATAL, LOG_FAILURE, LOG_WARNING_HIGH, LOG_WARNING_LOW, LOG_NORMAL, LOG_VERBOSE or LOG_DEBUG. Severities are used in the Agent Log application to control log levels and notifications.
- **Message string.** This message will be added to the log document.

- **Link.** Include a NotesDatabase, NotesView or NotesDocument object to insert a link in the log document.

About the ParseParams function

You can use the ParseParam function to make string concatenation more readable.

This code sample code:

```
ParseParams( "Document '%1' is created %2 by %3 on %4", doc.subject, now, session.commonUsername, db.server )
```

Gives exactly the same result as:

```
"Document" & doc.subject( 0 ) & "' is created " & now & " by " & session.commonUsername & " on " & db.server
```

With the ParseParams function it is easier to use constant's or to make language translations.

About the global SystemLog

The AgentLog should be enabled in the top level script – but will, potentially, be used in all functions and methods and in all included libraries.

We recommend to use the global SystemLog object when you want to log from sub functions/libraries (better than using global variables or passing the object as a parameter).

Sample code:

```
Sub Initialize
  Set logdb = session.GetDatabase( db.Server, "General\Administration\NccAgentLog.nsf" )
  Dim l As New AgentLog( logdb ) 'create the log object
  Call SystemLog.set( l ) 'make the object global

  Call Foo()

  Call l.close() 'close the log
End Sub

Function Foo()
  Call SystemLog.LogAction( LOG_NORMAL, "Hello World", Nothing ) 'write to the log
End Function
```

Traceable Error messages

Use a construction like this in all subs, methods and functions:

```
Function foo()  
  On Error Goto eh  
  
  '<Your code here>  
  
Exit Function  
eh:  
  Error Err.GetErrorInfo("")  
End Function
```

If an unexpected error occurs you will get a full traceable report on the log document:

```
Name:                Sample Agent / Niro  
Database title:     NCC Agent Log  
Location:           NDK01/NDK/Niro:General\Administration\NccAgentLog.nsf  
Username:           Jakob Majkilde/NDK/GEAP  
  
Division by zero  
FOO, line: 19, class: CUSTOMCLASS  
RUN, line: 7, class: CUSTOMCLASS  
INITIALIZE, line: 21, agent: Sample Agent / Niro
```

As you can see on the log document above, a 'Division by zero' error occurred in line 19 in the 'Foo' function in the 'CustomClass'. This function was called by the Run method (line 7) which was called from the sub Initialize in the 'Sample Agent'

The sample agent can be found in the Agent Log database if you want to view all the code.