



Novell Distributed Print Services NDPS Manager Debug Display

Version 1.1.01APR2001

Presented by WSS - Alan Adams

Copyright © 2001 by Novell, Inc. All rights reserved.
No part of this document may be reproduced or transmitted in any form or by any means,
electronic or mechanical, including photocopying and recording,
for any purpose without the express written permission of Novell.
All product names mentioned are trademarks of their respective companies or distributors.

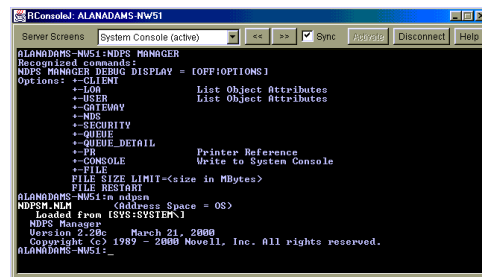
Novell®

NDPS Manager Debug Display. The NDPS Manager NLM (NDPSM.NLM) has built-in debug logging functions which are designed primarily to assist the NDPS developers in troubleshooting problems that occurred during NDPS Manager operations. These debug logging facilities can be invoked by anyone on any NDPSM.NLM version; no special debug build of the NDPSM.NLM is required. The only caveat is that 50% of the information captured might only make sense if you actually are an NDPS developer.

The information exposed by the NDPS Manager Debug Display can be highly useful for isolating and pinpointing specific issues. In exactly what way the NDPS Manager Debug Display information will be useful depends on the nature of the problem. The NDPS Manager Debug Display only exposes information about NDPS Manager operations, which is only one aspect of a complete NDPS printing environment. If the problem is client-side, specific to a particular NDPS gateway or with a particular NDPS Broker service the NDPS Manager Debug Display may only provide indirect clues or simply confirmation that no problems appear to be occurring at the NDPS Manager.

Many of the operation concepts of the NDPS Manager Debug Display are similar to an NDS Trace screen. The NDPS Manager Debug Display parameters allow the enabling and disabling display of individual categories of debug information. When enabled, the NDPS Manager Debug display may show the output of several different but concurrent operations occurring asynchronously. Specific classes of information are differentiated by color to facilitate easier identification.

Activation of the NDPS Manager Debug Display. The command used to control this debug logging is “NDPS MANAGER DEBUG DISPLAY”, issued at the server console. This is an actual command (not a SET parameter) which is registered and valid when NDPSM.NLM is running in memory. If you intentionally enter only “NDPS MANAGER” at the server console while NDPSM.NLM is running, a useful help screen will be displayed with most of the common NDPS Manager Debug Display options.



```
ALANADAMS-NW51:NDPS MANAGER
Received command:
NDPS MANAGER DEBUG DISPLAY = [OFF|OPTIONS]
Options:
+CLIENT
+LOG List Object Attributes
+USER List Object Attributes
+GATEWAY
+NDS
+SECURITY
+GUIDE
+GUIDE_DETAIL
+PR Printer Reference
+CONSOLE Write to System Console
+FILE FILE SIZE LIMIT<size in MBytes>
+FILE RESTART
ALANADAMS-NW51:n ndpsm
NDPSM.NLM Address Space = OS
Loaded from [SYS:SYSTEM]
NDPS Manager
Version 2-20c March 21, 2000
Copyright (c) 1987 - 2000 Novell, Inc. All rights reserved.
ALANADAMS-NW51:
```

Turning on the NDPS Manager Debug Display is simply a matter of issuing the following command:

```
NDPS MANAGER DEBUG DISPLAY = ON
```

This command creates a new console screen named “NDPS Manager Debug Display” on the server. Initially nothing will be displayed on the screen until additional options are also enabled. The NDPS Manager Debug Display screen is where all NDPS Manager Debug Display output will primarily appear. (The information may also be optionally logged to a file, or output to the main server console screen; see the +FILE and +CONSOLE parameters discussed in later sections.)

To turn the NDPS Manager Debug Display screen back off:

```
NDPS MANAGER DEBUG DISPLAY = OFF
```

Note that the NDPS Manager Debug Display screen is also where the “NDPS Manager Debug Display Options Menu” is accessible from. The NDPS Manager Debug Display Options Menu is discussed further in later sections.

Debug NDPSM.NLM builds. For nearly all cases of troubleshooting an NDPS issue, a debug build of the NDPSM.NLM module is not required. Nearly all information available is presented by the shipping NDPSM.NLM module. There is additional information included in the output of the NDPS Manager Debug Display when using actual debug builds of NDPSM.NLM, however. Examples include time spent either acquiring or holding a semaphore in excess of 30 seconds, including the address of who was holding the semaphore and/or the address of the function attempting to acquire it. For the additional output from the actual debug NDPSM.NLM builds, a coredump of the running server when the log was taken would also be required for matching address information being displayed. As such, this approach will typically only be recommended when working directly with an NDPS developer on troubleshooting a problem, and need not be employed unless specifically requested.

Control of the NDPS Manager Debug Display. As additional options on the NDPS Manager Debug Display are enabled or disabled, the NDPS Manager Debug Display will show the output of functions generating the requested information in real time. Depending on the frequency with which the functions occur, either due to how frequently they occur in

normal operation and/or how busy the NDPS Manager is with operations that call those functions, there could potentially be a massive amount of information being sent to the NDPS Manager Debug Display Screen.

To keep information from scrolling off the NDPS Manager Debug Display screen before it can be viewed, pressing the "P" key while viewing the NDPS Manager Debug Display screen will cause the display to pause. The NDPS Manager Debug Display will show "<Screen Output Paused>" when this option is active. Information being captured from the enabled NDPS Manager Debug Display options will be queued up for display while the screen is paused. If more than 2000 messages become queued while the display is paused, the display will automatically un-pause.

While the NDPS Manager Debug Display screen is paused, pressing the spacebar will cause only "the next screen full" of messages to be displayed. Because some messages can be rather long, "the next screen full" may be less than a full screen in order to prevent newly displayed information from scrolling off the screen. Because some operations output multiple messages, the output of a single operation may potentially be broken across two different screens.

To fully resume a paused NDPS Manager Debug Display screen, pressing the "R" key will resume normal output without queuing up messages or displaying only the next screen full. Note that even when the screen is un-paused, the "<Screen Output Paused>" indicator may still be visible until new output causes it to be scrolled from the NDPS Manager Debug Display screen or the screen is cleared.

Pressing the "C" key will clear the current NDPS Manager Debug Display Screen. Note that this will clear the "<Screen Output Paused>" indicator if the output is paused, so it won't be possible to tell whether the cleared screen is also still in a paused state.

To insert a blank line into the current NDPS Manager Debug Display output, simply press the ENTER key. This can be useful to separate distinct operations during testing or troubleshooting.

The other mechanism which helps cope with large amounts of information being received by the NDPS Manager Debug Display is the +FILE option discussed later which causes all debug output to also be logged to a file.

Finally, an additional key active on the NDPS Manager Debug Display screen is the "O" key which activates the NDPS Manager Debug Display Options Menu, also discussed later. Note that while the NDPS Manager Debug Display Options Menu is active (prompting for input to select an option) or a specific menu option is displaying information, all NDPS Manager Debug Display output is paused. Pressing the "R" key to resume output is required after having viewed a specific option menu screen.

NDPS Manager Debug Display parameters. In addition to "ON" and "OFF", the "NDPS Manager Debug Display" command supports many additional parameters for enabling specific classes of information to be displayed such as "+CLIENT" and "+NDS". The exact parameters supported and what information they expose is discussed in later sections.

The NDPS Manager Debug Display parameters which begin with "+" also support invoking them with "-" to disable the specific directive without disabling debug display as a whole. e.g. "+FILE", "-FILE", "+CLIENT", "-CLIENT", etc.

The NDPS Manager Debug Display parameters also support multiple directives being passed on a single command line, for example:

```
NDPS MANAGER DEBUG DISPLAY = +CLIENT +NDS +USER -LOA
```

The NDPSM.NLM will display which options it considers active after each NDPS MANAGER DEBUG DISPLAY command line is processed, so look at the displayed options carefully. If an intended parameter isn't confirmed by NDPSM.NLM, it may simply be misspelled on the command line.

Enabling the NDPS Manager Debug Display before the NDPS Manager loads. Note that although the NDPS MANAGER DEBUG DISPLAY command is not available until NDPSM.NLM is actually loaded in memory, this does not mean you cannot enable the NDPS Manager Debug Display screen prior to NDPSM.NLM actually starting to load the NDPS Manager. To accomplish this, load NDPSM.NLM without specifying an NDPS Manager object as a command line parameter such that a browse dialog is presented for selecting which NDPS Manager object to load as. Before actually browsing to and selecting an NDPS Manager object, switch over to the Novell NetWare server console prompt and issue your NDPS MANAGER DEBUG DISPLAY commands. Now switch back

and select the NDPS Manager object and the debug output will show the activity during initialization. This is usually a good method for getting additional information on fatal NDS initialization errors such as -603; by enabling +NDS prior to selecting which NDPS Manager to load as all the initial NDS operations will be logged.

NDPS MANAGER DEBUG DISPLAY = +NDS

Enables the logging of NDS operations being performed by the NDPS Manager through NDPSM.NLM's own NDS functions (as opposed to indirectly, through CLIB or other potentially DS-dependant calls), regardless of what operation within NDPSM.NLM is requesting the information.

There are four main types of messages which appear on the NDPS Manager Debug Display with the +NDS option enabled: "UniGetAttrVal()", "UniSetAttrVal()", "Read", and "Modify".

"Read" and "Modify" are the basic debug output for any DS operation logged by the NDPS Manager. Normally they are not verbose unless an error has occurred. Under normal circumstances they show what NDS server the operation was performed against, which NDS context handle on that server connection was used, and the amount of time spent performing the operation. (More on identifying NDS server connections and context handles in the section "NDPS Manager Debug Display Options Menu".)

```
Read          server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0034
Modify        server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0037
```

When an error has occurred during a "Read" or "Modify" operation, the NDS error code and object name on which the operation was being performed is also displayed. Note that in the current NDPSM.NLM versions through v2.20g, "Read" operations do not have verbose error output, only "Modify" operations.

```
Read          server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0034
Modify        server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0037
Modify returned error -614 on object HPGATE Test Agent One.Novell
```

"UniGetAttrVal()" and "UniSetAttrVal()" are the NDPS Manager wrapper functions for manipulating NDS object attribute values. Normally these wrapper functions do not produce any additional output. But on error,

"UniGetAttrVal()" and "UniSetAttrVal()" will display the particular attribute which was subject of the modify or read operation being performed. Note that a "Read" or "Modify" operation displaying the server, handle & time information will precede the "UniGetAttrVal()" and "UniSetAttrVal()" output.

```
Read          server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0034
UniGetAttrVal() returned error -119
  Attempting to get "ACL"
  Object HPGATE Test Agent One.Novell
```

The operation we see in this example is a "Read" operation displaying the normal NDS server, NDS context handle and operation time. Immediately in the next line, we see that this "Read" operation was actually part of a "UniGetAttrVal()" attempting to read the "ACL" attribute off the object ".HPGATE Test Agent One.Novell". We only know this only because the "Read" operation returned an NDS error code; if the "Read" operation had been successful, "UniGetAttrVal()" would not have produced any additional output and the "Read" operation would have been the only line visible.

The NDS error code in this case was -119, DSERR_BUFFER_TOO_SMALL. While at first glance the implication of this error code appears serious, this error is actually not uncommon to encounter with NDPS. The NDPS Manager was actually performing an operation more to test the existence of a particular attribute, in this case "ACL". The NDPS Manager used UniGetAttrVal() to read the ACL list, but supplied a zero-length buffer to accept the reply with. Success (or more specifically, lack of an error code) was never a possibility; the NDPS Manager just needed to see which error code would result. If the error had been -603, ERR_NO_SUCH_ATTRIBUTE the NDPS Manager would have known that the attribute didn't exist. Causing a -119, DSERR_BUFFER_TOO_SMALL lets us know the attribute exists without having to allocate or handle room for the actual attribute contents. This is a not uncommon scenario, where NDS errors logged in the NDPS Manager Debug Display are actually 100% expected and are part of normal operation. Seeing -119, -603, -614 and other NDS errors are not necessarily indication of a problem.

```
Read returned error -603 (expected) on object HPGATE Test Agent One.Novell
UniGetAttrVal() returned error -603 (expected)
  Attempting to get "Preferred Resource Services"
  Object HPGATE Test Agent One.Novell
```

When calling a “Read” or “Modify” operation under which an error is expected, the NDPS Manager specifies one error code which is potentially “expected”. The NDPS Manager Debug Display output for +NDS attempts to make expected errors known by displaying “(expected)” if the specified error code was received.

The limitation is that for the purposes of the NDPS Manager Debug Display only one error code can be an “expected” error code, while the reality is that many NDS error codes might actually be expected. As visible in the log example above, the -603, ERR_NO_SUCH_ATTRIBUTE error displayed during a “UniGetAttrVal()” for the “Preferred Resource Service” attribute was displayed as “(expected)” even though it’s potentially no more or less expected than -119, DSERR_BUFFER_TOO_SMALL might have been.

```
Modify          server: ALANADAMS-NW51, handle: 2   Time: 0:00:00.0037
Modify returned error -614 on object HPGATE Test Agent One.Novell
UniSetAttrVal() returned error -614
  Attempting to change "ACL"
  Object HPGATE Test Agent One.Novell
```

Because the “Modify” operation shown above encountered an error, the “Modify” operation itself displayed additional information citing the NDS error code received and NDS object name on which the “Modify” operation was occurring.

Immediately after we see that this “Modify” operation was actually part of a “UniSetAttrVal()” operation, which we know because “UniSetAttrVal()” kicked out additional information when the NDS error code was received.

In this case the NDS error code was -614, ERR_DUPLICATE_VALUE. This too was expected and/or had no ramifications; the NDPS Manager was simply ensuring that a particular entry existed in the “ACL” attribute. Rather than spend multiple operations testing for the attribute’s existence and then testing whether the needed value already exists for that attribute, the NDPS Manager simply attempts to add the needed value. This way only a single operation is expended and the NDPS Manager looks at the NDS error code returned to judge what happened, existed and/or was already present as a value.

When reviewing output of an NDPS Manager Debug Display with the +NDS option enabled, the type of object and attribute being

manipulated with have to be taken into account in addition to the error code received. For example:

```
Read returned error -603 on object ALANADAMS-NW51_SYS.Novell
UniGetAttrVal() returned error -603
  Attempting to get "Host Server"
  Object ALANADAMS-NW51_SYS.Novell
```

This represents a real problem even though it’s again a -603, ERR_NO_SUCH_ATTRIBUTE (which was an “expected” error in other cases). Without a “Host Server” attribute on what appears to be an NDS volume object, there is no way that NDPS Manager is going to resolve a connection to that volume. Issues such as queue servicing problems or access to the NDPS Manager database or a particular NDPS Printer Agent’s spooling area would be observed.

Note that the screen output of the +NDS parameter is typically purple in color, with mixed-case text display.

NDPS MANAGER DEBUG DISPLAY = +GATEWAY

The +GATEWAY parameter exposes the calls being made through the gateway interface APIs of the NDPS Manager. In addition to actual NDPS gateway interface APIs, +GATEWAY will show nearly all “NLM interface” operations, which in addition to a gateway could be a job scheduler NLM, accounting NLM, etc. Since primarily the NDPS Manager Debug Display would be used to view operations being requested by NDPS gateways, these operations will be collectively referred to as simply “gateway operations”.

For normal operation, the gateway operations exposed by the +GATEWAY parameter will display the PA ID and the time at which the operation occurred (as opposed to time spent performing the operation). Depending upon the operation requested, the operation may have a single entry (simply that it was done/requested), or there may be an operation pair showing it was requested and then responded to.

The following log sample shows typical output from a single gateway starting for an NDPS Printer Agent:

```

BIND_TO_PA                               Time: 2-11-01  7:09:51 pm
BIND_TO_PA                               Time: 2-11-01  7:09:51 pm
SET_ATTRIBUTE_SET                         PA 0
LIST_OBJ_ATTR                             PA 0
FREE_LOA_SET_RESULT                       PA 0
REQ_JOB_FROM_JPM                          PA 0

```

The BIND_TO_PA operation lets the loading NDPS gateway NLM bind to the specific NDPS Printer Agent for which it was loaded. Once bound the SET_ATTRIBUTE_SET operation represents the gateway updating the NDPS Manager database with status information about the gateway and/or printer, and the LIST_OBJ_ATTR and FREE_LOA_SET_RESULT operations are the gateway querying information from the NDPS Manager Database. The amount of information updated or queried will vary from gateway to gateway, depending upon basic implementation details and/or the amount of printer information and features the gateway is capable of tracking.

The REQ_JOB_FROM_JPM is the gateway evidently being in a ready enough state that it's ready to take an available print job as soon as one is available. A gateway will typically not make this request if the gateway failed some part of initialization. Depending upon the specific gateway this request may not be made if the printer isn't connected and ready to accept a job; some gateways may make the request and then wait on the physical printer to be ready once they have a job.

The gateway simply sits in the idle state after having called REQ_JOB_FROM_JPM waiting for a job handle to be returned to the gateway. The gateway may perform other operations in the background while it's waiting such as updating printer information received from the physical printer. So there may be other operations logged in the NDPS Manager Debug Display for this gateway while it's sitting in this state, but in terms of printing the gateway is waiting on it's REQ_JOB_FROM_JPM call to return a job handle.

```

JOB_OPENED                               PA 0 Job D550FC8C Time: 2-11-01  7:11:51 pm
REQ_DOC_DATA                             PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
REQ_DOC_DATA                             PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
REQ_DOC_DATA                             PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
SUPPLY_DOC_DATA                          PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
SUPPLY_DOC_DATA                          PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
SUPPLY_DOC_DATA                          PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
REQ_DOC_DATA                             PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
SUPPLY_DOC_DATA                          PA 0 Job D550FC8C Time: 2-11-01  7:11:53 pm
SET_ATTRIBUTE_SET                         PA 0
SET_ATTRIBUTE_SET                         PA 0
REPORT_EVENT                              PA 0
SET_ATTRIBUTE_SET                         PA 0
ATTR_FLAG_ADD                             PA 0
ATTR_FLAG_ADD                             PA 0
REQ_JOB_FROM_JPM                          PA 0

```

The JOB_OPENED entry is the response to the REQ_JOB_FROM_JPM request made earlier. Note that a specific job handle is displayed by which specific jobs can be identified from one other within the NDPS Manager Debug Display.

Once the gateway has the document handle, it will use REQ_DOC_DATA to fill it's own data buffers with print job data which it will in turn transmit to the physical printer. The gateway does not typically just take the entire job; it takes just enough to fill buffers and then comes back for more once those buffers have been transmitted. In this example, we see an HPGATE.NLM-based gateway requesting three buffers full of document data, which are provided in the SUPPLY_DOC_DATA responses from the NDPS Manager. One more REQ_DOC_DATA / SUPPLY_DOC_DATA occurs, which may represent the final part of the job since no more requests are seen.

The gateway performed more SET_ATTRIBUTE_SET operations and a REPORT_EVENT; although the +GATEWAY output doesn't expose which attributes or events are being referenced these were likely to be the gateway updating status to report that we're currently printing.

Finally, we see the gateway issue another REQ_JOB_FROM_JPM request, which indicates the gateway is ready to process another document as soon as one is available and begin the same process over again.

```

SET_ATTRIBUTE_SET      PA  0          Time: 2-11-01  7:12:49 pm
REPORT_EVENT           PA  0          Time: 2-11-01  7:12:49 pm
SET_ATTRIBUTE_SET      PA  0          Time: 2-11-01  7:12:49 pm
ATTR_FLAG_ADD          PA  0          Time: 2-11-01  7:12:49 pm
ATTR_FLAG_ADD          PA  0          Time: 2-11-01  7:12:49 pm
SET_ATTRIBUTE_SET      PA  0          Time: 2-11-01  7:13:07 pm
SET_ATTRIBUTE_SET      PA  0          Time: 2-11-01  7:13:07 pm
CLOSE_DOC              PA  0   Job D550FC8C Time: 2-11-01  7:13:07 pm
CLOSE_JOB (reason: 0)  PA  0   Job D550FC8C Time: 2-11-01  7:13:07 pm
SET_ATTRIBUTE_SET      PA  0          Time: 2-11-01  7:28:09 pm
ATTR_FLAG_ADD          PA  0          Time: 2-11-01  7:28:09 pm
ATTR_FLAG_ADD          PA  0          Time: 2-11-01  7:28:09 pm

```

The job hasn't actually printed at the printer yet though, something we know indirectly by what is shown in the above log section.

When the gateway is done with a document handle received from a JOB_OPENED operation, it will issue a CLOSE_JOB operation on the handle. Although the REQ_JOB_FROM_JPM request was made again at 7:11:59PM, we didn't see a CLOSE_JOB operation until 7:13:07PM. This is because this particular gateway actually knew how to check whether a job sent to the printer has physically printed, and held onto the job handle until it knew for sure what happened at the physical printer.

The intended effect of this is making the job still be listed on by the NDPS Printer Agent as "printing" until the job is actually laying in the output tray. This prevents user notification (if requested) from occurring until physical paper exists, as opposed to having been sent to the printer's local memory buffer. Conversely, the gateway may also find the printer reports an error, and the gateway will return a non-successful status on close so that the job is returned to the list of jobs waiting to be serviced.

The "CLOSE_JOB (reason: 0)" indicates that the gateway is done with the particular job handle and closed it. The "reason" code shows the code returned by the gateway as the reason for closing the job:

0x00, JOB_CLOSE_REASON_COMPLETE: The job was completed successfully and normally. The job will be deleted from the spooling area unless the spooling configuration on the NDPS Printer Agent specifies job retention.

0x01, JOB_CLOSE_REASON_CANCEL: The user or operator has requested that the job be canceled. The job will be deleted from the spooling area since the job was explicitly canceled.

0x02, JOB_CLOSE_REASON_ABORT: The gateway encountered some operating condition that forced the aborting of the current printing attempt for the job. The job will be left in the spooling area as an available document for the gateway to attempt printing again.

0x03, JOB_CLOSE_REASON_JOB_PAUSE: The gateway was requested to pause output. The job will be left in the spooling area as an available document for the gateway to begin printing again.

0x04, JOB_CLOSE_REASON_NO_DATA_FILE: The gateway was unable to open the document file for the specified job. Basically this means the NDPS Manager database had record of a job & document file but the document file no longer exists in the NDPS Printer Agent spooling area. This could be because the NDPS Manager was restored to a new server (which restores only the database, not the pending or retained print jobs). The job record will be removed from the NDPS Manager database.

0x05, JOB_CLOSE_REASON_ABORT_AND_HOLD: The job is being aborted (as opposed to canceled) and will remain in the NDPS Printer Agent's spooling area but in a "Held" state (which prevents it from printing again automatically but is still available for printing once the "Held" state is removed). Currently only occurs in response to an accounting NLM reporting that the job could/should not be printed.

Not all gateways are capable of doing anything besides sending the job to the physical printer and then closing the job as serviced, because either the gateway and/or the remote printer device don't support more granular status checking or event reporting.

Gateways may also generate scores of SET_ATTRIBUTE_SET and LIST_OBJ_ATTR operations for no apparent reason when simply performing a periodic status check or printer information update.

Note that the screen output of the +GATEWAY parameter is typically gray in color, with all upper-case text for function names.

NDPS MANAGER DEBUG DISPLAY = +CLIENT

The +CLIENT parameter exposes the calls being made through the NDPS client interface APIs. Note that "client" does not exclusively mean client workstations; it means anything which uses the client APIs. The

client APIs can be called by NDPSM.NLM itself, a gateway, or other NLMs running on the server.

The primary operations which will be seen through the client interface are workstations NWDP_BindPA, NWDP_Unbind, NWDP_ListObjectAttributes, NWDP_Print and NWDP_TransferData operations. This constitutes probably 95%-99% of the client workstation's interaction with the NDPS Manager.

These operations reported by +CLIENT display the calendar times of the incoming request ("Start"), and for the response display the amount of time spent performing the operation ("End"). In addition, the NDPS client process name on which the operation is being handled is displayed. These are the names of actual processes running on the NetWare server which were created to handle an incoming client connection. A connected client gets its own "NDPSM_____xxxxx" process on the server, on which it can perform as many workstations NWDP_BindPA, NWDP_Unbind or other operations as it wants until the connection is closed. Note that a single client workstation may, and typically does, have several client connections open to a server concurrently.

The following example shows a typical workstation printing pattern:

```

NWDP_BindPA           Start      2-13-01  9:58:13 am  NDPSM           23
NWDP_BindPA           End, Time used: 0:00:00.0007  NDPSM           23
NWDP_ListObjectAttributes  Start      2-13-01  9:58:13 am  NDPSM           23
NWDP_ListObjectAttributes  End, Time used: 0:00:00.0011  NDPSM           23
NWDP_Unbind           Start      2-13-01  9:58:13 am  NDPSM           23
NWDP_Unbind           End, Time used: 0:00:00.0003  NDPSM           23
NWDP_BindPA           Start      2-13-01  9:58:13 am  NDPSM           24
NWDP_BindPA           End, Time used: 0:00:00.0006  NDPSM           24
NWDP_ListObjectAttributes  Start      2-13-01  9:58:13 am  NDPSM           24
NWDP_ListObjectAttributes  End, Time used: 0:00:00.0008  NDPSM           24
NWDP_Print            Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_Print            End, Time used: 0:00:00.0247  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0013  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0014  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0012  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0014  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0011  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0011  NDPSM           24
NWDP_TransferData     Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_TransferData     End, Time used: 0:00:00.0004  NDPSM           24
NWDP_Print            Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_Print            End, Time used: 0:00:00.0087  NDPSM           24

```

```

NWDP_Unbind           Start      2-13-01  9:58:14 am  NDPSM           24
NWDP_Unbind           End, Time used: 0:00:00.0003  NDPSM           24
NWDP_BindPA           Start      2-13-01  9:58:14 am  NDPSM           25
NWDP_BindPA           End, Time used: 0:00:00.0006  NDPSM           25
NWDP_ListObjectAttributes  Start      2-13-01  9:58:14 am  NDPSM           25
NWDP_ListObjectAttributes  End, Time used: 0:00:00.0010  NDPSM           25
NWDP_Unbind           Start      2-13-01  9:58:14 am  NDPSM           25
NWDP_Unbind           End, Time used: 0:00:00.0003  NDPSM           25

```

The NDPS Manager Debug display shows a NWDP_BindPA Start/End on the "NDPSM 23" client connection process. Note that the "Start" and "End" refer to the NWDP_BindPA operation itself; not that the client was bound and then unbound.

The client then performed an NWDP_ListObjectAttributes operation on that connection. This, just like the LIST_OBJ_ATTR operation over the gateway interface, is the client querying information from the NDPS Manager database. Typically the client is simply querying printer agent status information and/or list of active and ready jobs. Knowing what specifically the NWDP_ListObjectAttributes was querying can be revealed using the +LOA parameter discussed elsewhere in this document.

Note that the client issues an NWDP_Unbind operation once the NWDP_ListObjectAttributes is finished, which indicates that's all the client intended to do with this connection and no longer needs to be bound. Technically the client can NWDP_BindPA again using the existing client connection, but this is rarely seen from Windows-based client workstations except during specific management functions.

The client created another connection (which shows up as "NDPSM 24" for the client process name) on which they also NWDP_BindPA and then NWDP_ListObjectAttributes checking the printer status. On this same connection however, the client also issues an NWDP_Print operation. This is the operation the client uses to create a new print job on the NDPS Printer Agent. The subsequent NWDP_TransferData operations are the client submitting the actual print job data. The client updates the job as being ready for servicing after the NWDP_TransferData operations are complete using a final NWDP_Print operation. With this done, the client issues an NWDP_Unbind for the client connection.

After successfully submitting the job, a third client connection process ("NDPSM 25") is shown using NWDP_BindPA and

NWDP_ListObjectAttributes to check the printer agent status information and list of active and ready jobs.

Note that the screen output of the +CLIENT parameter is typically gray in color, with mixed-case text for function names.

NDPS MANAGER DEBUG DISPLAY = +USER

The +USER parameter works in conjunction with the +CLIENT display to expose the user credential under which the operation is being performed. For workstation client connections this will typically display the NDS user object or other NDS object under which authority the operation is being called. The +USER parameter has no effect when +CLIENT is not also enabled.

```
NWDP_ListObjectAttributes      Start      2-13-01 10:42:03 am  NDPSM      215
NWDP_ListObjectAttributes      End, Time used: 0:00:00.0011  NDPSM      215

NWDP_ListObjectAttributes      Start      2-13-01 10:42:03 am  NDPSM      216
NWDP_ListObjectAttributes      User: .AlAdams.Novell      NDPSM      216
NWDP_ListObjectAttributes      End, Time used: 0:00:00.0011  NDPSM      216
```

These two operation examples show the same operation being performed with only +CLIENT enabled, and then again with +CLIENT and +USER enabled.

Note that the screen output of the +USER parameter is typically cyan in color, with mixed-case text for function names.

NDPS MANAGER DEBUG DISPLAY = +SECURITY

The +USER parameter works in conjunction with the +CLIENT display to expose what credentials & security level is being enforced during a client bind operation. The +SECURITY option itself is only available in NDPSM.NLM 2.10 and later.

```
NWDP_BindPA                    Start      2-13-01 10:51:42 am  NDPSM      272
NWDP_BindPA                    End, Time used: 0:00:00.0009  NDPSM      272

NWDP_BindPA                    Start      2-13-01 10:51:42 am  NDPSM      273
Bind Type: 1, Credential Type: NWDP_CREDENTIALS_NDPS_1, Security Level 2
Bind to: .HPGATE Test Agent One.Novell
File Server: ALANADAMS-NW51, Connection number: 23
User: .AlAdams.Novell
Client Rights: ROLE_PSM_MANAGER ROLE_PA_USER ROLE_PA_OPERATOR
ROLE_PA_ACCOUNTANT
NWDP_BindPA                    End, Time used: 0:00:00.0009  NDPSM      273
```

These two operation examples show the same bind operation being performed with only +CLIENT enabled, and then again with +CLIENT and +SECURITY enabled.

The “Bind Type” refers to the operation being NWDP_BindPA as opposed to any other type of bind. “Credential Type” declares which of the NDPS internal credential types were provided by the client; this is usually always the same and not important.

The “Security Level” shows the security level set and enforced for the NDPS Printer Agent the client is binding to. “Security Level 3” is the one which equates to “High” security level and causes huge amounts of NDS traffic for every client operation. Medium will report as “2” and low reports as “1”.

The “Bind to” line shows what the client had requested to bind to; this could be an NDPS Printer object name, or an NDPS Printer Agent name.

The “File Server”, “Connection number” and “User” are actually what the client provided as part of his credential set for the bind indicating which NetWare server name and connection number they’re on. This information is used as part of the Medium and High security processing.

The “Client Rights” display the roles that the specified user has been assigned or is equivalent to.

Note that the screen output of the +SECURITY parameter is typically yellow in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +PR

Displays information regarding NDPS Manager functions maintaining NDPS Printer object to NDPS Printer Agent relationships ("PO to PA" relationships).

```
UpdatePrinterObjectLists 2-13-01 11:19:39 am
Last resync occurred 54 seconds ago. Early return
```

UpdatePrinterObjectLists() is the function which refreshes the information from NDS for the NDPS Manager. The UpdatePrinterObjectLists() function has a timer threshold during normal use which prevent refreshing from NDS too often (which will be expensive, potentially slow, and in many cases unnecessary). During specific events such as creation of a new NDPS Printer Agent, UpdatePrinterObjectLists() will be called to perform a refresh immediately (regardless of the timer threshold) just to ensure no conflicts exist for what the NDPS Manager is about to do.

The "Early return" flag shown in the above example indicates that the threshold for refreshing NDS information has not yet been exceeded, so the UpdatePrinterObjectLists() function did not actually go to NDS to refresh the information. It also indicates that the NDPS Manager had not called for an immediate refresh regardless of the timer threshold. If the NDPS Manager had actually commenced with an NDS refresh, only the "UpdatePrinterObjectLists" line would have been displayed.

Note there is no additional output when a UpdatePrinterObjectLists() refresh is actually occurring, at least not as a result of having +PR set. The +PR output of the NDPS Manager Debug Display would simply indicate the UpdatePrinterObjectLists() event started and does not show "Early return". The +NDS output would show any errors actually being encountered when accessing the information in NDS.

```
Operation: NWDP_SET_PRINTER_OBJ_ADD
Printer Agent: HPGATE Test Printer Two
DS Printer: HPGATE Test Printer Two.Novell
```

The output shown above is an example of the output during an UpdatePrinterObjectReference() function which was adding ("Operation: NWDP_SET_PRINTER_OBJ_ADD") a NDPS Printer NDS object ("DS Printer: HPGATE Test Printer Two.Novell") to an NDPS Printer Agent ("Printer Agent: HPGATE Test Printer Two"). This occurs when creating a new controlled access NDPS Printer Agent and the

NDPS Printer NDS object is associated to the NDPS Printer agent for the first time. This would also occur when converting an existing public access NDPS Printer Agent to controlled access by associating a new NDPS Printer object to an existing NDPS Printer Agent.

```
BackgroundAssocPoToPa PA: HPGATE Test Printer Two, Mask: FFFFFFFF, Count: 1
BKG_MASK_PO_TO_PA_ID_MAPPING ccode: 0
BKG_MASK_PRINTER_NET_ADDR ccode: 0
BKG_MASK_PSM_OBJECT_NAME ccode: 0
BKG_MASK_PRINTER_AGENT_NAME ccode: 0
UpdatePrinterObjectLists 2-13-01 11:20:35 am
BackgroundAssocPoToPa completed PA: HPGATE Test Printer Two
```

BackgroundAssociatePoToPa() is used to make the NDPS Manager perform effectively even when NDS synchronization delays are occurring. This function effectively allows the NDPS Manager to return immediate "success" for associating an NDPS Printer object to an NDPS Printer Agent, but in the background this process just keeps trying until the cited NDPS Printer object actually appears and is successfully associated using the NDS replica NDPS is talking to.

The +PR output for BackgroundAssociatePoToPa() displays a count of how many times BackgroundAssociatePoToPa() has tried to associate, and an operation value indicating how far through the association process we have successfully progressed.

Note that the screen output of the +PR parameter is typically yellow in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +DESCRIPTOR

The +DESCRIPTOR output reveals warnings and failures related to descriptors used during NDPS Manager database access. In general the conditions flagged by the +DESCRIPTOR output would be considered secondary symptoms of higher level problems that caused NDPS Manager or NDPS Gateway code to get stuck while it happened to be holding a descriptor. These warnings and failures would result from other code wanting or using the descriptor.

Most typically output from +DESCRIPTOR is rare and not seen. It's existence during CPU hog or high utilization may just be a secondary symptom of the real problem. Enabling the output of +DESCRIPTOR is usually recommended simply "just in case" the information proves useful.

```

DBDeleteRecord Failed; Object User Count > 1. Thread: NDPSM 34245
Object Descriptor: 0xDFE78201
Object Key: 0xC045FE02 0x00000001 0x00000000 0x00000001
DBDeleteRecord 0xD48832EA
Calling Function 0xD1E98421
Opening Function: 0xCF02833A
Opening thread: NDPSM 8723

```

```

CheckOutDescriptor Failed. Thread: NDPSM 34245
checkinWait: 0x00000010 checkinWaitClear: 0x00001000
Object Descriptor: 0xDFE78201
Object Key: 0xC045FE02 0x00000001 0x00000000 0x00000001
CheckOutDescriptor: 0xCD3472EE Lock: Shared
Calling Function 0xD1E98421
Locking Function: 0xCF02833A Lock: Exclusive
Locking thread: NDPSM 8723

```

```

CheckinDescriptor Warning. Thread: NDPSM 34245
Locked Time: 239947
Object Descriptor: 0xDFE78201 Lock: Shared
Locking Function: 0xCF02833A
Symbol: NWDPPGetObjectRef+54AE

```

Descriptor 0xDFE78201 was checked in but was not checked out

All of the above examples are of output that could be seen with +DESCRIPTOR enabled. Note that the values in the above examples are completely fictitious and not from an actual NDPS Manager Debug Display output.

Note that the screen output of the +DESCRIPTOR parameter is typically red in color (except for warnings which are yellow), with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +LOA

Displays more detail about List Object Attribute (LOA) calls being made to the NDPS Manager database. LOAs are how most information is retrieved from the NDPS Manager database and is a very common and useful call to log.

```

NWDP_ListObjectAttributes Start 2-13-01 10:36:24 am NDPSM 183
NWDP_ListObjectAttributes End, Time used: 0:00:00.0816 NDPSM 183

NWDP_ListObjectAttributes Start 2-13-01 10:36:24 am NDPSM 184
Object Class: Printer
Object Id: HPGATE Test Agent One
Requested attribute: Printer State
Requested attribute: Jobs Scheduled Count
Requested attribute: Active Jobs Count
NWDP_ListObjectAttributes End, Time used: 0:00:00.0816 NDPSM 184

```

These two operation examples show the same bind operation being performed with only +CLIENT enabled, and then again with +CLIENT and +LOA enabled.

Note that the screen output of the +LOA parameter is typically cyan in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +CHECK_SERVICES

Exposes the operations being performed as part of the CheckConnections() function. The CheckConnections() function is a loop running on it's own thread which is used verify the connections which the NDPS Manager maintains to NDPS Broker services (ENS, RMS and SRS) on behalf of the NDPS Printer Agents running on the NDPS Manager, and to initiate obtaining a new connection to a NDPS Broker service when no connection exists or the current connection had problems.

Once every second CheckConnections() looks to see whether a global flag has been set indicating that the NDPS Event Notification Service (ENS) broker connections (reconnectNotificationGbl) or NDPS Resource Management Service (RMS) broker connections (reconnectResourceGbl) need to be checked.

These global flags can get set because a new NDPS Printer Agent has been created and needs NDPS Broker services assigned, or because some NDPS Printer Agent has encountered a problem communicating with its assigned NDPS Broker service, destroyed the existing connection, and needs a functioning NDPS Broker service assigned.

So as soon as any NDPS Printer Agent or function raises either the reconnectNotificationGbl or reconnectResourceGbl flag, the CheckConnections() loop should notice within about a second unless CheckConnection() is already active. Note there is no output from +CHECK_SERVICES during this per-second check if both reconnectNotificationGbl and reconnectResourceGbl are found to be FALSE (0, i.e. no reconnection needed).

When CheckConnection() finds either of these global reconnect flags set, CheckServices() becomes active and walks the list of NDPS Printer Agents running and verifies their NDPS Broker service connection. If the NDPS Printer Agent is controlled access, CheckConnections() will

ensure the NDPS Printer Agent is talking to its assigned preferred NDPS Broker for the service in question and will attempt to connect to the preferred NDPS Broker if the NDPS Printer Agent isn't already connected to it.

For non-controlled access NDPS Printer Agents or for controlled access for which the preferred NDPS Broker cannot currently be reached, CheckConnections() ensures that an NDPS Broker is connected and that we can ping the service (at an NDPS level, not simply network transport protocol). CheckConnections() will initiate finding a new NDPS Broker service for the NDPS Printer Agent and/or clearing the current connection and raising the appropriate reconnect flag depending on which of those checks aren't true.

```
CheckConnections() Active, 3-06-01 9:52:33 am
reconnectNotificationGbl: 1, reconnectResourceGbl: 1
GetLocalBrokeredService() returned 0 (0)
GetBrokeredService() serviceType 2
GetBrokeredService() selected broker .ALANADAMS-NW51-BROKER.NOVELL
CheckNService(), ccode 0, paHandle 0, 0:00:00.2919
GetLocalBrokeredService() returned 0 (0)
GetBrokeredService() serviceType 3
GetBrokeredService() selected broker .ALANADAMS-NW51-BROKER.NOVELL
srsRefGbl is NULL
SRS reconnect code: 0:00:00.1398
CheckConnections() suspended, 3-06-01 9:52:33 am
```

In the example above, the +CHECK_SERVICES output shows CheckConnections() became active and had found both reconnectNotificationGbl and reconnectResourceGbl set to TRUE (1). Either one of these flags on its own could have been set and CheckConnection() would have still become active. In this case both flags were set because a new NDPS Printer Agent, the only NDPS Printer Agent on the NDPS Manager in fact, had been created.

Because reconnectNotificationGbl is set, for each NDPS Printer Agent the CheckConnections() loop calls CheckNService() to check the NDPS Printer Agent's NDPS Event Notification Service (ENS) broker connection. In the +CHECK_SERVICES example above the CheckNService() return code for a specific NDPS Printer Agent ("paHandle 0") is shown as successful (0). There is only one NDPS Printer Agent on the NDPS Manager, so checking this one NDPS Printer Agent represents the entire list. The time displayed on the CheckNService() line ("0:00:00.2919") is how long we spent in CheckNService() for the specified NDPS Printer Agent ("paHandle 0").

Because reconnectResourceGbl is set, for each NDPS Printer Agent the CheckConnections() loop calls CheckRMService() to check the NDPS Printer Agent's NDPS Resource Management Service (RMS) broker connection. Note that the +CHECK_SERVICES output does not explicitly show the CheckRMService() NDPS Printer Agent handle, return code or time information. This isn't intentional and may change in future builds of NDPSM.NLM.

Both CheckNService() and CheckRMService() first determine whether there is a preferred NDPS Broker set for the NDPS Broker service type (ENS or RMS) being checked. If there is a preferred NDPS Broker specified, CheckNService() and CheckRMService() will ensure the NDPS Printer Agent is already connected to it or will attempt to connect to the preferred NDPS Broker. If there isn't a preferred NDPS Broker service (i.e. not a controlled access printer) and the NDPS Printer Agent already has a NDPS Broker connection, nothing is changed.

When no NDPS Broker service is found connected for the NDPS Printer Agent and/or the preferred NDPS Broker service cannot be reached, both CheckNService() and CheckRMService() use GetBrokeredService() to obtain a connection to an NDPS Broker of the specified type (ENS or RMS).

GetBrokeredService() first checks for an NDPS Broker service of the specified type running on the local server, then checks for an NDPS Broker service of the specified type to which the NDPS Manager is already maintaining a connection for some other NDPS Printer Agent on the NDPS Manager, and finally will try and obtain an NDPS Broker service of the specified type by querying the NDPS Service Registry Service (SRS) broker to which the NDPS Manager is connected.

GetBrokeredService() stops at any one of these steps as soon as it has successfully acquired an NDPS Broker service of the specified type. The above example showed "GetLocalBrokeredService() returned 0 (0)" which indicates success in finding an NDPS Broker an NDPS Broker service of the specified type running on the local server. If there had not been an NDPS Broker service of the specified type running on the local server, and the NDPS Manager did not already have a connection to an NDPS Broker service of the specified type, the +CHECK_SERVICES output during this CheckNService() would have looked more like the following example:

```
GetLocalBrokeredService() returned 1 (1)
GetInUseBrokeredService() returned 1 (1)
GetBrokeredServiceFromReg() returned 0 (0)
```

This +CHECK_SERVICES output represents that the check for a local NDPS Broker service of the specified type and the check for the NDPS Manager already being connected to an NDPS Broker service of the specified type both returned non-zero (non-success) values which is why we went all the way to the NDPS Service Registry Service (SRS) to get the NDPS Broker connection (via GetBrokeredServiceFromReg()).

This is one place to be watchful within the NDPS Manager Debug Display output when NDPS Printer Agents aren't getting a the preferred NDPS Broker service connection they have been assigned, or are failing to get an NDPS Broker service connection at all. Knowing how far GetBrokeredService() had to go in attempting to get a connection, whether excessive time was spent attempting and timing out on any specific check, etc. will provide evidence to what kind of problem the NDPS Manager is encountering when attempting to obtain an NDPS Broker service connection for the NDPS Printer Agent.

Looking at the original example again, note that the +CHECK_SERVICES output for GetBrokeredService() occurs immediately prior to the CheckNService() output indicating which NDPS Printer Agent the check was conducted for. Again, there is no CheckRMService() output to indicate which NDPS Printer Agent is being processed, although as shown in the original example above the GetBrokeredService() output during the CheckRMService() was still shown.

Having covered what CheckConnections() will do when the reconnectNotificationGbl or reconnectResourceGbl flags are found to be set, let's return for a moment to the per-second loop which is checking these flags and waiting to run CheckConnections().

Every 120 seconds, regardless of whether the reconnectNotificationGbl or reconnectResourceGbl flags are set, the CheckConnections() function becomes active anyway. In this case, because reconnection has not been indicated as necessary by either reconnectNotificationGbl or reconnectResourceGbl, CheckConnections() does not immediately walk the list of NDPS Printer agents.

Instead CheckConnections() walks the list of current NDPS Broker connections which the NDPS Manager is maintaining, and ensures each

of them can be successfully pinged (at an NDPS level, not simply network transport protocol).

```
CheckConnections() Active,      3-10-01  7:11:24 pm
reconnectNotificationGbl: 0, reconnectResourceGbl: 0
Pinged nsClientTableGbl[1], ccode 0,      0:00:00.1245
Pinged rmsClientTableGbl[1], ccode 0,      0:00:00.0012
Pinged rmsClientTableGbl[2], ccode 0,      0:00:00.1210
NWDPsSrsGetConnectionStatus, ccode 0,      0:00:00.0013
CheckConnections() suspended, 3-10-01  7:11:24 pm
```

In the above example it's clear the CheckConnections() function became active even though reconnectNotificationGbl and reconnectResourceGbl are both zero (no reconnection necessary).

CheckConnections() then pinged each NDPS Event Notification Service (ENS) and NDPS Resource Management Service (RMS) connection which the NDPS Manager was maintaining. The NDPS Manager only maintains one connection to any given NDPS Broker for any given NDPS Broker service type (ENS, RMS or SRS). Seeing multiple entries being pinged for any one service type (in the above example, the fact that we see CheckConnection() pinging both rmsClientTableGbl[1] and rmsClientTableGbl[2]) means that NDPS Printer Agents on the NDPS Manager probably have preferred NDPS Brokers set which have caused the NDPS Manager to have to maintain connections to various NDPS Broker services.

If a non-responsive NDPS Broker connection is discovered, functions are called to clear the NDPS Broker service connection from any NDPS Printer Agent using it and raise the appropriate reconnect flags. The +CHECK_SERVICES output will show anon-zero value (e.g. "ccode 1") being returned for the ping attempt on the table entry.

If the reconnect flags don't end up getting set during the ping of the entire table of NDPS Broker service connections, the CheckConnections() processing goes on to check whether any NDPS Printer Agents are currently without an NDPS Broker service connection (either ENS or RMS) or are not connected to their preferred NDPS Broker service (for controlled access printers) and will raise the appropriate reconnect flags. There is not any +CHECK_SERVICES output specific to this operation except that the CheckConnections() function would immediately go active again with one of the reconnect flags (reconnectNotificationGbl or reconnectResourceGbl) raised even though they were both zero when the ping of the entire NDPS Broker service connection table had started.

Finally, the focus has been on NDPS Event Notification Service (ENS) and NDPS Resource Management Service (RMS) connections because references to these connections are maintained per-NDPS Printer Agent and thus require more processing.

The NDPS Manager maintains a single NDPS Service Registry Service (SRS) connection that is used for all NDPS Printer Agents on the NDPS Manager. There is not a table of SRS broker service connections because there is only one of them, ever.

```
NWDPSSrsGetConnectionStatus, ccode 0, 0:00:00.0013
```

When the single NDPS Service Registry Service (SRS) connection entry is valid, CheckConnections() will simply attempt to ping the connected NDPS Broker service to ensure it's responding. If the ping fails, the +CHECK_SERVICES output will show "NWDPSSrsGetConnectionStatus" with a non-zero "ccode" value.

```
srsRefGbl is NULL  
SRS reconnect code: 0:00:00.1398
```

If the single NDPS Service Registry Service (SRS) connection has been deemed invalid or has never been established, +CHECK_SERVICES will show that "srsRefGbl" (which is the single entry maintained as opposed to a table for ENS and RMS) is "NULL". The "SRS reconnect code" output is actually only showing the time spent in attempting to get an SRS; whether the NDPS Manager was successful or not is not revealed. Whether success was achieved can be inferred however depending on whether srsRefGbl is still NULL the next time CheckConnections() runs.

Note that the screen output of the +CHECK_SERVICES parameter is typically yellow in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +QUEUE

Exposes the remote server communication involved in checking and servicing jobs from native Novell NetWare print queues, if any of the NDPS Printer Agents are configured to service legacy queues.

Primarily the +QUEUE output reveals connection-oriented activity, i.e. did the NDPS Manager successfully create an NCP-level connection for servicing the queue, did the NDPS Manager lose an NCP-level

connection to a remote server we had a legacy queue on, was the NDPS Manager successful in re-establishing the connection, etc.

```
Reconnect Process active, 3-10-01 7:21:54 pm  
file server ALANADAMS-NW51, CONNECT_PARTIAL, retry count 2  
Reconnect Process suspended, 3-10-01 7:21:54 pm
```

The process identified as "Reconnect Process" by the +QUEUE output is a loop which runs immediately once the NDPS Manager is up and then runs every 60 seconds, and is responsible for making the initial connections and re-connections for native queue NCP server connections.

The "CONNECT_PARTIAL" state that can be reported for a connection means that the NCP connection has been re-established but the queues that need the specified connection are not yet being serviced. The queues will be moved back to being in service the next time the "Reconnect Process" is active.

```
PSMLoginToFileServer() failed for server ALANADAMS-NW51, ccode 5  
GetQueueInfo() failed, Error 248, Queue .Native Queue One.Novell  
  
PSMAttachQueueServerToQueue() failed  
Queue: .Native Queue One.Novell  
  
NWQAttachServer() failed, Error code: 5  
  
PSMLoginToFileServer() returned: 5  
fileserver: ALANADAMS-NW51  
connectionId: 1  
  
Error: 255, getting the 'HOST SERVER ATTRIBUTE'  
queue: .Native Queue One.Novell
```

These are examples of some error messages which can be seen from +QUEUE being enabled. Note the specific error values shown are fictitious and not from an actual NDPS Manager Debug Display.

Typically +QUEUE is used with +TIME in order to determine whether excessive delay is occurring in the operations that are invoking NCP-level connections or DS operations.

Note that the screen output of the +QUEUE parameter is typically green in color (except for outright errors, which are red), with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +QUEUE_DETAIL

Provides additional debug output around the internal functions used while transferring queue jobs from the queue and then creating the native NDPS job for data retrieved from the queue. Another way to think of this is that +QUEUE enables output which may show why the NDPS Manager was unable to successfully connect to a queue, while +QUEUE_DETAIL shows information about the NDPS Manager queue activity when successfully connected.

Typically +QUEUE_DETAIL is not useful for most issues except where jobs submitted to NDPS via native Novell NetWare queues are being serviced (removed from the queue) but are failing to appear or print through assigned NDPS Printer Agents. Note the +QUEUE_DETAIL parameter is only available in NDPSM.NLM 2.20c and later.

```
NWQBeginJobService returned 0
1347 Bytes Read
DataXfer complete, 1347 Bytes
CreateObjectsAndXferData returned 0
NWQEndJobService returned 0
```

Primarily what is exposed by +QUEUE_DETAIL is the actual data transfer and return codes for the calls used in turning a single legacy queue job into a single NDPS print job on the assigned NDPS Printer Agent. Watching that the return codes are zero (successful) and the amount of data transferred from the legacy queue matches what was expected were the intention behind the information +QUEUE_DETAIL exposes.

Note that neither parameter (+QUEUE nor +QUEUE_DETAIL) exposes the queue polling operation that is occurring every six seconds.

Note that the screen output of the +QUEUE parameter is typically green in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +TIME

The +TIME parameter causes specific functions that have been coded to display additional time information to expose how long a particular function or operation took to complete.

In some cases the +TIME parameter causes functions which are also involved in other NDPS Manager Debug Display parameters to simply

display additional time information, but there are other functions which will only appear within the NDPS Manager Debug Display by virtue of +TIME having been enabled.

```
Time: Gateway Unbind                NDPSM_Main      0:00:00.2490
Time: SetPrinterStatusInDS          NDPSM_Main      0:00:00.0149
Time: _ShutdownPANoSemaphore        NDPSM_Main      0:00:00.2948
Time: SetPrinterStatusInDS()        NDPSM_Main      0:00:00.0174
Time: DoPrinterSpecificStuff()      NDPSM_Main      0:00:00.1373
Time: SetDSInfoInHandle()           NDPSM_Main      0:00:00.1479
Time: ClearEventFlagsAndJobProfiles() NDPSM_Main      0:00:00.0229
Time: InitPANoShutdownStateChange(),HP Banner NDPSM_Main      0:00:00.2180
Time: LoadPhPdsAndAccounting()      NDPSM_Main      0:00:00.0015
```

Having +TIME enabled when shutting down and restarting a single NDPS Printer Agent generated the above output. Note that functions such as "LoadPhPdsAndAccounting()" (the function which causes the NDPS Gateway to load for the NDPS Printer Agent) is not exposed as having occurred through any other NDPS Manager Debug Display parameter.

As such, in addition to being useful for exposing functions in which excessive amount of time was spent when troubleshooting a performance or hanging-type issue, turning on +TIME in general can be useful for simply exposing the trail of operations which was executed, and not specifically for the time information revealed about those operations.

Note that the screen output of the +TIME parameter is typically cyan in color, with mixed-case text.

NDPS MANAGER DEBUG DISPLAY = +FILE

Logs any event and API output generated on the NDPS Manager Debug Display to the file SYS:SYSTEM\NDPSM.LOG. Note this file is reset to 0 bytes each time +FILE is enabled, so be sure to save the existing NDPSM.LOG if needed before enabling +FILE again.

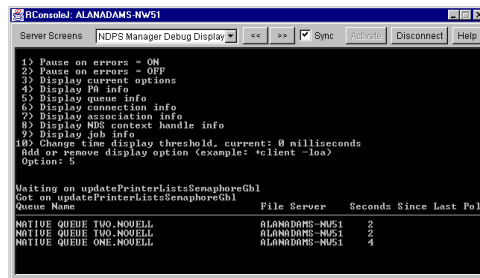
NDPS MANAGER DEBUG DISPLAY = FILE SIZE LIMIT=50

Controls how large the NDPSM.LOG is allowed to grow. By default the log is only 4MB, but can be increased via this parameter to the specified number of megabytes. (The example shown sets the log to 50MB maximum.) When the log reaches the specified limit, it is simply deleted and the NDPSM.LOG starts over at 0 bytes again. As such a

Manager Debug Display screen. For example, if the NDPS Manager Debug Display had shown an LOA operation being performed by gateway on "PA 4", entering "4" on the "Display PA info" would provide the details on that NDPS Printer Agent including the name.

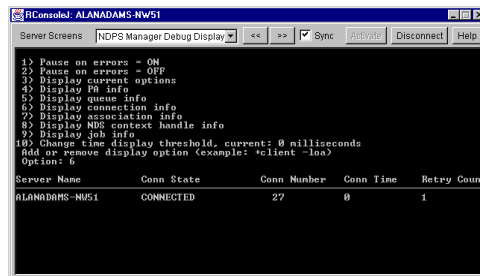
Most of the other information shown refers to structures and addresses not useful in troubleshooting but useful for a developer interactively debugging an issue.

5) Display queue info: This option will list all of the NetWare native queues which are associated to NDPS Printer Agents, which NetWare server the NDPS Manager has a connection to for servicing the queue, and how many seconds have elapsed since the queue was last polled checking for waiting jobs. Note that the queue polling loop works on a six-second interval, so usually the "Seconds since last poll" is less than that value. Also note that if there are more queues associated than will fit on a single screen, the data will simply scroll and cannot be viewed or captured to a log in its entirety.



6) Display connection info: Shows the NCP server connections which the NDPS Manager is currently maintaining. NCP connections are usually only related to queue access or the checking of security on medium and high security NDPS Printer Agents.

The "Server Name" identifies the server to whom we have created a connection, and the "Conn Number" column identifies our NCP connection number on that server.



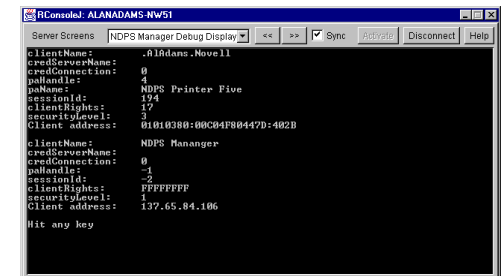
The "Conn State" is the field the NDPS Manager uses to track the state of the connection. "CONNECTED" identifies a non-temporary connection in a good state. "CONNECT_LOST" indicates a connection in need of logging in; possibly a new connection just created and not logged in yet, or a connection

that was intentionally flagged as in need of being reconnected due to hard errors or too many retries. "CONNECT_PARTIAL" indicates that the connection has been logged in but queues have not been successfully reconnected (yet). "CONNECT_TEMPORARY" is a connection the NDPS Manager will create to other servers (usually for the purposes of security checking) which if not used for 60 seconds should be disconnected automatically. Note that "CONNECT_TEMPORARY" also implies the connection is currently in a good state.

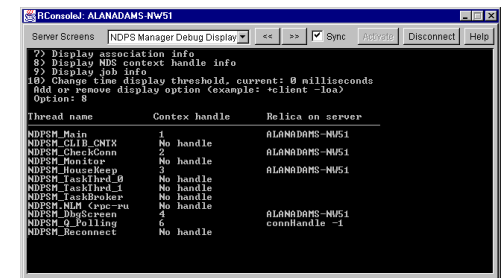
The "Conn Time" is only used on temporary connections; on any connection other than a temporary connection it will display "0". On a temporary connection it holds the number of seconds for which the connection has been living.

"Retry Count" gets incremented when queue operations cannot be completed and are left for the "Reconnect Process" to attempt again. Currently after ten retries the connection should be moved into a "CONNECT_LOST" state for full reconnection and login processing to be initiated on the next "Reconnect Process" loop.

7) Display association info: NDPS association structures maintain information about an authenticated entity. The NDPS Manager maintains a table of these structures and this option will display all of the entries in the table two at a time, pausing after each screen. The information on this table at any given time is changing because client connections are frequently being created or destroyed.

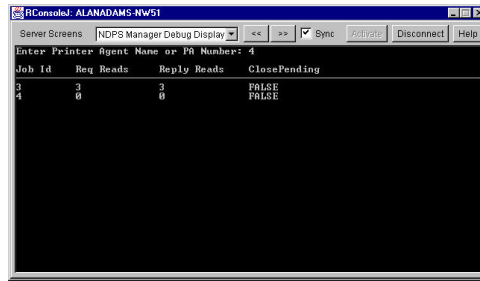


8) Display NDS context handle info: Various NDPS Manager threads will use NDS and maintain an NDS context handle. This screen quickly lists which threads currently have NDS contexts handles and for the threads which do, which NDS replica server that



handle is presently accessing data from. Information from this list can be useful during testing and troubleshooting to determine in larger environments which NDS replica(s) are being hit by the NDPS Manager.

9) Display job info: By specifying the NDPS Printer Agent name or ID number, this option will display the list of currently active jobs for an NDPS Printer Agent. The “Req Reads” and “Reply Reads” is in reference to data requested and supplied for the document through the gateway interface (JPM_OP_REQ_DOC_DATA & JPM_OP_SUPPLY_DOC_DATA). In other words, the data going out as the job is being serviced by the gateway, not the data coming in from the client connection or job submission. “Close Pending” is true when the gateway has requested a close that has not been processed to completion by the NDPS Manager yet.



The screenshot shows a window titled "RConsole: ALANADAMS-NW51" with a menu bar "Server Screens" and "NDPS Manager Debug Display". Below the menu bar is a prompt "Enter Printer Agent Name or PR Number: 4". The main area displays a table with the following data:

Job ID	Req Reads	Reply Reads	ClosePending
3	3	3	FALSE
4	0	0	FALSE

10) Change time display threshold: The +TIME parameter on the NDPS Manager Debug Display can be configured to only display the “Time:” information in the NDPS Manager Debug Display screen when the amount of time a function took exceeds a pre-defined number of milliseconds. The reason that all “Time:” information is displayed by default when +TIME is enabled is because this value defaults to “0”. By setting this parameter to something greater than zero, any functions that take less than the number of milliseconds specified will not display “Time:” information when +TIME is enabled. To revert back to displaying “Time:” information for all functions, specify a value of “0”.

Add or remove display option: Finally, simply typing an NDPS Manager Debug Display parameter at the menu prompt, e.g. “+CLIENT” or “-NDS”, and the parameter will be interpreted as though the parameter had been specified on the NDPS MANAGER DEBUG DISPLAY command line.