nirva

Nirva Systems technical newsletter – February 2010

New features 2

Command stack 2
Thread session 4
Write to output buffer 6
64 bits 8
Windows 7 8
Processor affinity 8
Large files 8
Debug tool 8
HP Dialogue service 9

Modifications 11

Virtual printer in Nirva distribution 11
New compiler 11
Session list 12
Security 12

Bug corrections 12

Nirva Application Platform 12
PDF Service 13

New features

Command stack

When testing or debugging procedures, it is often useful to be able to view the list of commands being executed in the current procedure. Developers can already visualize this list in the Nirva console by using the ‘debug’ and ‘verbose’ modes of Nirva.

However, when Nirva runs in service mode, or when it is remotely accessed, it might be helpful to visualize in the administration interface the commands and procedures being executed in a particular session. An enhancement of the SESSION:LIST command is now available to achieve this.

SESSION:LIST command

The SESSION:LIST command can be used to obtain the current command and procedure stack, As in Java or C++ programming, each command or procedure in input is placed on the stack and is taken out of it at the end of their execution. As a consequence, only commands and procedures awaiting execution (or being executed) are displayed in the stack,

Please be aware that this enhancement is only available if Nirva has started in ‘Debug’ mode (please refer to the Configuration/System/Parameters/Debug chapter of the Nirva documentation).

The following example illustrates the new functionality. The PERL procedure shown below uses the SESSION:LIST command and displays the results as populated in the STACK column of the SESSIONS table generated by the command.

main.pl:

```
# Get sessions information, including current
NV::Command("NV_CMD=|SESSION:LIST| WITH_ACTUAL=|YES|");
# Get sessions number
NV::Command("NV_CMD=|OBJECT:TABLE_GET_NUM_ROWS| NAME=|SESSIONS|");
my $num_sessions = $NV::RESULT;


# For each session get number of commands in stack trace (corresponding to lines number
in STACK column)
# Then display stack in console
for (my $i=1; $i <= $num_sessions; $i++){
    NV::Command("NV_CMD=|OBJECT:TABLE_GET_CELL_NUM_LINES| NAME=|SESSIONS| ROW=|$i| COLNAME=|STACK|");
   my $num_session_procs = $NV::RESULT;
   my @session_stack = ();
   for (my $j=1; $j <= $num_session_procs; $j++){
      NV::Command("NV_CMD=|OBJECT:TABLE_GET_CELL_LINE| NAME=|SESSIONS| ROW=|$i| COLNAME=|STACK|
LINE_INDEX=|$j|");
   push(@session_stack,$NV::RESULT);
   }
   if($#session_stack > 0){
      NV::Command("NV_CMD=|OBJECT:TABLE_GET_CELL_LINE| NAME=|SESSIONS| ROW=|$i|
```

```
COLNAME=|IDENTIFIER|");
       my $id = $NV::RESULT;
       print "Identifier : $id\n";
       foreach my $action (@session_stack){
          print "\t$action\n";
       }
   }
}
```

URL to launch the test (requires the SYSTEM_SESSION_INFO permission):

```
http://localhost:1081/nv_app_NEWSLETTER/NVS?Command&NV_PROC=perl:NL1/Article3/main&NV_CL
OSE_SESSION=NO&NV_USER=nvadmin&NV_PASSWORD=nirva
```

Result (after deletion of the command display):

```
########################
Identifier : 43E806E696
        Command: SYSTEM:MISC:NOP(B)
        Procedure: perl:perl:NL1/Article3/main
        Command: SYSTEM:SESSION:LIST(P)
########################
```

One can see that only the procedures being executed are present in the stack. In this example, this is the test procedure and, as expected, the SESSION:LIST command being executed when executing the command. If other sessions exist and actively process a command or procedure during the test, the result would also display the other active sessions:

```
########################
Identifier : 5D0AB46D3E
        Command: SYSTEM:MISC:NOP(V)
        Procedure: session_open
        Command: SYSTEM:MISC:NOP(P)
        Procedure: perl:perl:session_open
        Command: SYSTEM:MISC:NOP(P)
        Procedure: perl:perl:Portals/Main/session_open
        Command: SYSTEM:MISC:NOP(P)
        Procedure: perl:perl:Settings/load_application_settings
        Command: SYSTEM:MISC:NOP(P)
        Procedure: perl:perl:Modules/Database/search
        Command: SYSTEM:MISC:NOP(P)
        Procedure: perl:perl:Modules/Database/load_query_file
        Command: SYSTEM:OBJECT:CREATE(P)
########################
########################
Identifier : 691F53A28A
        Command: SYSTEM:MISC:NOP(B)
        Procedure: perl:perl:NL1/Article3/main
        Command: SYSTEM:SESSION:LIST(P)
########################
```

The top part of the results shows procedures and commands currently in the stack of the '5D0AB46D3E' session. This particular example displays the creation of a session in an application.

Administration interface

This enhancement also allows the administration interface to display the current 'stack' for a particular session. This function is accessed by clicking on the chosen session number in the 'Sessions' tab. To display the stack related to a particular session, the session must be active and processing as executed commands and procedures are removed from the stack once completed.

Conclusion

With the SESSION:LIST command enhancement it is now possible to identify a problem occurring in a long procedure. It this procedure takes longer than expected, a list of its stack displays what is currently executing and allows to identify potential problems.

This functionality can also be accessed remotely or with Nirva started in service mode.

More information

The WHAT parameter of the SESSION:LIST command can be used to filter the results: For a particular application, a service or a session ID.

The new 'nvd' tool (see Debug tool in this document) will allow the listing of all command lines associated with a session or even with a future session (e.g. the next Web session). These enhancements follow a deliberate direction Nirva Systems is taking to enhance application development and maintenance.

Thread session

In order to trigger an asynchronous task in an execution flow, one previously had to trigger a predefined task. Data sharing between caller and called was deemed cumbersome.

The new THREAD:CREATE command can be used to alleviate these difficulties by simply mentioning which procedure to execute, with its optional parameters such user ID, output procedure or the error procedure.

The thread that is created will remain independent from the calling thread: the caller can terminate before the called without the latter being affected.

The following example illustrates the case where a Perl procedure (test_thread.pl) needs to trigger a time consuming and heavy process (e.g. sending a 500 MB to an external FTP server). The application must return control to the user as soon as possible to allow further processing while the 500 MB will be transferred at the available network speed.

The long process is represented by the Perl thread.pl procedure. The slow part is simulated by a series of calls to MISC:SLEEP of 5 seconds each

test_thread.pl:

```
NV::GetSessionId();
my $session_id = $NV::RESULT;
```

```
print "[ROOT] Parent process, session id is $session_id\n";
print "[ROOT] Create a thread session to do an asynchronous task...\n";
NV::Command("NV_CMD=|THREAD:CREATE| PROC=|perl:NL1/Article1/thread|
CLOSE=|perl:NL1/Article1/thread_close| "); (1)
my $thread_session_id = $NV::RESULT;  (2)
print "[ROOT] A new thread ($thread_session_id) has been created\n";
print "[ROOT] End of the parent process...\n";
```

test_post_proc.pl:

```
print "[ROOT_POST] Post proc of test\n";
```

test_session_close.pl:

```
print "[ROOT_CLOSE] Session_close\n";
```

thread.pl:

```
NV::GetParameter("NV_CALLING_SESSION"); (3)
my $calling_session_id = $NV::RESULT;

NV::GetSessionId();
my $current_session_id = $NV::RESULT;

print "\t[THREAD] Thread has been started by session $calling_session_id, its session id
is $current_session_id\n";

for( my $j = 0; $j < 5; $j++){
  #wait 5s to simulate extensive process...
  NV::Command("NV_CMD=|MISC:SLEEP| TIME=|5000|"); (4)
  print "\t[THREAD] Send a big file!\n";
}
print "\t[THREAD] $current_session_id Ended\n";
```

Command URL:

```
http://localhost:1081/nv_app_NEWSLETTER/NVS?Command&NV_PROC=perl:NL1/Article1/test_threa
d&NV_POST_PROC=perl:NL1/Article1/test_post_proc&NV_SESSION_CLOSE=perl:NL1/Article1/test_
session_close
```

Result (after deletion of commands display):

```
########################################
[ROOT] Parent process, session id is E9A9E713E3
[ROOT] Create a thread session to do asynchronous tasks...
[ROOT] A new thread (6C5C9F1586) has been created
[ROOT] End of the parent process...
        [THREAD] Thread has been started by session E9A9E713E3,
        its session id is 6C5C9F1586
[ROOT_POST] Post proc of test
[ROOT_CLOSE] Session_close
        [THREAD] Send a big file!
        [THREAD] Send a big file!
        [THREAD] Send a big file!
        [THREAD] Send a big file!
        [THREAD] Send a big file!
        [THREAD] 6C5C9F1586 Ended
        [THREAD_CLOSE] Closing the thread
########################################
```

Log entries prefixed by [ROOT] are generated by the calling procedure whilst log entries prefixed by [THREAD] are generated by the called asynchronous session. Messages clearly show that the asynchronous task is still working despite the calling procedure being terminated, even closed (session close).

Note: sessions are created and not children of their parents: the calling session loses control of the sessions that is created. These sessions are independent from one another. They are ‘aware’ of each other but do not share information about their state.

Conclusion

Asynchronous execution is successfully handled. The life span of the called session is limited to the execution of the procedure it triggers. The calling session is not waiting for called session to close before closing itself. As a result, please note that threads created this way should not manipulate or access objects controlled by the calling session as they may be modified or deleted during execution.

More information

The LANGUAGE parameter can be used to change the session language.

The PROCF parameter can be used to define a procedure that can be executed if an error occurs in the created session.

Write to output buffer

The command COMMAND :SET_OUTPUT_BUFFER can be used to return a character string that can be retrieved with the now traditional $NV::RESULT in PERL or GetResult() with the Java connector. The immediate advantage is to be able to return a value without having to create an object in the session.

With Nirva always wiping out buffer content before each command, it is necessary to call the command as the last step in the procedures or services.

main.pl:

```
#This is the main perl proc file
#Will call another proc and check the output buffer to get the result
my $result = NV::Command("NV_PROC=|perl:NL1/Article2/proc_hello| NAME=|John|");
my $greeting = $NV::RESULT;
print "The greeting is: $greeting\n";
print "The result of the procedure is: $result\n";

NV::Command("NV_Proc=|Java:TestOutputBuffer:set|");
my $message = $NV::RESULT;
print "Message: $message\n";

NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|STRING| NAME=|GREETING| VALUE=|$greeting|
REPLACE=|YES| NV_CONTAINER=|nvdef|");
NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|STRING| NAME=|MESSAGE| VALUE=|$message|
REPLACE=|YES| NV_CONTAINER=|nvdef|");
```

test_post_proc.pl:

```
print "[ROOT_POST] Post proc of test\n";
print "[PROC_HELLO] Start\n";

# Get the name
NV::GetParameter("NAME");
my $name = $NV::RESULT;

# Write the greeting in the output buffer
NV::Command("NV_CMD=|COMMAND:SET_OUTPUT_BUFFER| VALUE=|Hello, $name Smith|");
print "[PROC_HELLO] End\n";
```

Command URL:

```
http://localhost:1081/nv_app_NEWSLETTER/NVS?Command&NV_PROC=perl:NL1/Article2/main
```

Result (after deletion if commands display):

```
########################################
[PROC_HELLO] Start
[PROC_HELLO] End

The greeting is: Hello, John Smith
The result of the procedure is: 1
Message: hello world from a java procedure!
########################################
```

Conclusion

Rather than creating a temporary object to return a value, it is now possible to get an additional buffer. This must be used with caution as a Nirva command inserted after the set_output_buffer eliminates it. Using the command in the service init/exit and service session init/exit sections will not work.

More information

This command can be used in the NV_POST_PROC procedures.

64 bits

The 64 bit version of the nvc client library is now delivered with Nirva. This allows some connectors and particularly the Java client connector running in 64 bits (a 64 bits client JVM can now be used).

The 64 bits of the virtual printer connector is also available.

Windows 7

Nirva has been successfully tested on Windows 7.

The virtual printer connector also installs now on Windows 7.

Processor affinity

The Nirva production license is based on the number of processor cores. Until now it was not possible to make Nirva running on a machine having more cores than the number defined in the license.

Now, Nirva takes care of the processor core affinity and doesn't enter in demonstration mode if this matches the number of processors of the license.

Under Windows and Linux, the affinity mask is automatically set by program following the number of cores of the license.

Under Solaris and HP-UX, the affinity must be set externally by using OS tools (processors sets).

This functionality is not available on AIX because there is no affinity control on this OS (we can limit only to a single core).

Large files

Nirva is now able to manage file objects greater that 2GB. This concerns Nirva itself and the storage service.

Debug tool

Nirva provides a new debug tool named nvd. This tool can connect an existing session or wait for a session of a certain type to be opened and receives some debug information from the session. Nvd can also open a new session and execute a command or a procedure. There are several debug levels.

This tool runs only on the local server. It works also when Nirva is in service mode.

Examples

Create a session, run a procedure and exit the session:

```
C:\>nvd -r perl:ptest -l 3
Waiting for a session to be attached...
Session 37EBFF35D3 has been attached

Current command stack:
Command: SYSTEM:MISC:NOP(V)

15:16:38 start SYSTEM:MISC:NOP(V)
Debug: parameters:
        LEVEL = 3
        NV_POST_PROC =
        NV_PROC = session_open
        PID = 5100
        TYPE =
15:16:38 --- Start Native procedure: session_open
15:16:38 --- End native procedure: session_open
15:16:38 end   SYSTEM:MISC:NOP(V)
15:16:38 start SYSTEM:MISC:NOP(C)
15:16:38 end   SYSTEM:MISC:NOP(C)
Debug: output buffer: 168
15:16:38 start SYSTEM:SESSION:CLOSE(C)
15:16:38 end   SYSTEM:SESSION:CLOSE(C)
Debug: output buffer: 168
15:16:38 --- Start Native procedure: session_close
15:16:38 --- End native procedure: session_close

Console has been closed by nirva
```

Other examples:

```
Connect the next web session with level 3 (open the nirva config to see it):
nvd -s next:web -l 3


Connect the next threaded session for application myapp with level 2:
nvd -s next:thread -l 2 -a myapp


Connect a specific session with level 1:
nvd -s CB47B271A7
```

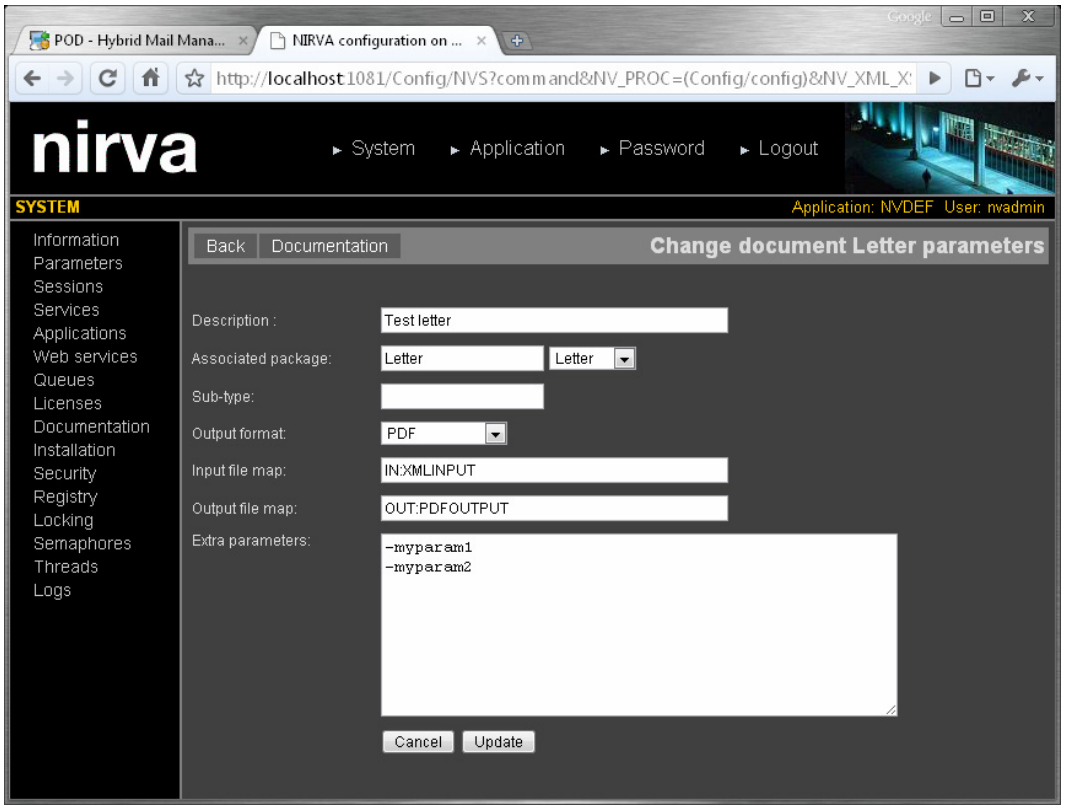
HP Dialogue service

The following functionality has been added to the Dialogue service:

- Static parameters
- Dynamic parameters
- Control file

Static parameters

One can define static Dialogue command line parameters associated to a given document. This is done from the user interface:



or by program using the DIALOGUE:DOCUMENT:SET_PARAM command.

This is useful for defining variables at document level.

Dynamic parameters

It was already possible to define extra command line parameter in a dynamic way by using the EXTRA_PARAM parameter in the DOCUMENT:COMPOSE or DOCUMENT:COMPOSE_DIRECT commands. Now this parameter may also point to a string list object containing the command line parameters to add.

Example:

```
NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|FILE| NAME=|IN|
FILENAME=|C:\\Nirva\\Applications\\NVDEF\\Procs\\Dialogue\\test.xml| PERSIST=|-1|
REPLACE=|YES|");

NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|FILE| NAME=|OUT|
FILENAME=|C:\\Nirva\\Applications\\NVDEF\\Procs\\Dialogue\\result.pdf| PERSIST=|-1|
REPLACE=|YES|");

NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|STRINGLIST| NAME=|PARAM| VALUE=|-param1;-
param2| REPLACE=|YES| SEPARATOR=|;|");
```

```
NV::Command("NV_CMD=|DIALOGUE:DOCUMENT:COMPOSE| WORKSPACE=|DNV| DOCUMENT=|Letter|
ERROR_NOSIZE=|YES| EXTRA_PARAM=|PARAM| CTRL_FILE=|YES|");
```

Control file

The Dialogue service can now use the Dialogue control file feature. This option allows sending command line parameters from a file instead of sending them from the command line. In order to activate the control file feature, just add the CTRL_FILE=|YES| parameter to the DOCUMENT:COMPOSE or DOCUMENT:COMPOSE_DIRECT commands.

Example:

```
NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|FILE| NAME=|IN|
FILENAME=|C:\\Nirva\\Applications\\NVDEF\\Procs\\Dialogue\\test.xml| PERSIST=|-1|
REPLACE=|YES|");

NV::Command("NV_CMD=|OBJECT:CREATE| TYPE=|FILE| NAME=|OUT|
FILENAME=|C:\\Nirva\\Applications\\NVDEF\\Procs\\Dialogue\\result.pdf| PERSIST=|-1|
REPLACE=|YES|");

NV::Command("NV_CMD=|DIALOGUE:DOCUMENT:COMPOSE| WORKSPACE=|DNV| DOCUMENT=|Letter|
ERROR_NOSIZE=|YES| CTRL_FILE=|YES|");
```

Modifications

Virtual printer in Nirva distribution

The virtual printer (VP) is a Nirva connector that retrieves files issued from desktop applications and sends them on a Nirva server to be processed. This can be useful for hybrid mail (post on demand) or archiving Nirva based solutions.

The virtual printer is seen as a printer on the user's workstation.

The virtual printer connector is located on the SDK/Connectors/virtual_printer subdirectory of Nirva installation directory

Win32 and x64 versions are available.

Please contact us for further information and documentation about this connector.

New compiler

The Nirva components have been recompiled under Windows with Visual C++ 2008 instead of Visual C++ 2003. This was necessary for some new Nirva features (ex. large file management).

The Nirva services will progressively migrate to this compiler when there will be some modifications to them. This is the case today with STORAGE, DIALOGUE and PDF services. Services compiled with Visual 2003 can be installed on Nirva compiled under Visual 2008. The opposite is also true.

The use of this new compiler may require the installation of a Microsoft redistribution patch for some old platforms (ex Windows 2003 Server). This patch is available at the following site:

http://www.microsoft.com/downloads/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en

The Perl part of Nirva stays compiled with Visual 2003 because the Perl glob function seems not to work when compiled with Visual 2008. Functionally this does not change anything and your perl procedure will continue to work in the same way.

Session list

The WITH_ACTUAL parameter in the SESSION:LIST command has been replaced by WITH_CURRENT. WITH_ACTUAL is still available for compatibility reason but is deprecated.

Security

A security issue has been corrected. The ".." string is not any more authorized in the NV_PROC or NV_XML_XSL parameters or other similar parameters requesting procedure or XSL file names.

Bug corrections

Nirva Application Platform

Description	Version
The TABLE_JOIN command may fail and potentially crash Nirva when the foreign key is a primary key and is not found.	3.0.011
Some error information was missing in the standard HTML error message returned by Nirva to a Web browser.	3.0.011
Application listener tables were loaded after the application init procedure making the listener list not visible from application init procedures. This may create problems when the init procedure tries to automatically create the missing listeners.	3.0.010
Defining a listener with an open procedure may crash Nirva. This bug occurs only on 3.0.008 version.	3.0.009

Description	Version
Possible compatibility issues with the recompiled perl on windows version 3.0.008 so we came back to a Perl compiled with an older compiler (only the Perl part is concerned, nirva is still compiled with the Visual C++ 2008 compiler). Especially the Perl glob functionality was not working.	3.0.009
Under Windows, hardware exception was not handled correctly (occurs in 3.0.008 only). This generates a Nirva crash when there is an hardware exception (for example when trying to access a bad memory address)	3.0.009
Dotnet connector: the constructor with the connect string was creating a compilation error.	3.0.008
The namespace for output messages was not the same than the one defined in the WSDL when using a namespace prefix.	3.0.008
Possible deadlock when requesting a command to be executed by another session from a procedure a service. This does not occur with named sessions but only when the NV_SESSION_ID parameter is given from a procedure or service command.	3.0.008
The alias path was forced in lowercase making some directories unreachable under Unix.	3.0.007
The OBJECT:TABLE_MODIFY_COLUMN command was not returning error in case of error.	3.0.007
The OBJECT:TABLE_SEARCH MAXDOCS command parameter was not working with a query equal to “*”.	3.0.007

PDF Service

Description	Version
The GET_TEXT command may produce some bad characters when these characters are not included in the character map associated to the font in the PDF flow.	1.40