NetIQ® Privileged User Manager Command Control Perl Scripting Guide

June 2013

Legal Notice

Table of Contents

Privileged User Manager Command Control Perl Scripting

Command Control Perl scripting provides a powerful method of extending the commands the way in which the Command Control Manager authorizes commands.

Command Control Rules are matched in order against the user, the command and the host that submits the command. Perl scripts can be used to extend the command control rules by:

- Interrogating external security databases
- Calling other Perl functions

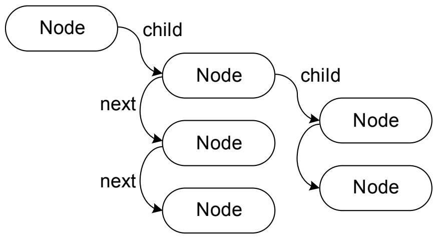
1 Metadata Structure

When you run a command in the Command Control Agent, data about the user environment and the command is collated. This data is known as the metadata. The metadata is sent to the Command Control Manager which makes the authorization based on the information provided. The Command Control Manager then updates the metadata, authenticates, and sends it back to the Command Control Agent. The Command Control Agent then contacts the Remote Execution service on the remote host to execute the command. This metadata can be audited, and can be queried/modified by the external Perl scripts linked to the Command Control Rules.

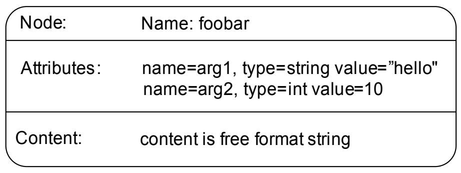
The metadata is encoded into a linked 'node' object. Each node has a name which can have siblings, children and attributes of its own. There is no limit on the number of siblings, children, or nodes with the same name. The structure is designed to complement the XML specification so that communications with external applications can be made using an extensible, industry standard protocol.

An example of the linked node structure:

Node structure



Node information


Node:
Name: foobar
Attributes:
name=arg1, type=string value="hello"
name=arg2, type=int value=10
Content:
content is free format string

Nodes have methods to facilitate setting, changing and retrieving the data from the nodes.

1.1 Creating Nodes

All Privileged User Manager modules have a global object called a 'context', which contains state information about that module, and provides services to the module. External Perl scripts are passed to this object as $ctx, and it is the context that creates nodes.

The following method creates a node with the name 'mynode':

```
 my $node=$ctx->create_node(‘mynode’);
```

To add a newly created node to an existing node use:

```
 my $new_node=$ctx->create_node(‘mynewnode’);
$parent_node->add($new_node);
```

Nodes are generally created as an offspring of an existing node directly:

```
 my $new_node=$node->add_node(‘mynewnode’);
```

1.2 Naming Nodes

Although nodes can be created without names, it is preferred to have a name attached to a node.

To retrieve the current node name use:

```
 my $n = $node->name();
```

To set the name:

```
$node->name(‘mynewname’);
```

1.3 Accessing and Changing Nodes

Each node may have a sibling (effectively the next node in the list) and an offspring (which in itself may have siblings and offspring).

There are a number of methods available to retrieve nodes from the list. Generally, a node with a specific name is required. To search for a specified node use:

```
my $n = $nodes->child('foo');
```

This will find the first offspring with the name 'foo'. However you can retrieve all nodes with a specific name:

```
my @nlist = $nodes->children('foo');

foreach my $node (@nlist) {
   print $node->name();
}
```

To retrieve all child nodes use:

```
my @nlist = $nodes->children();
```

To delete a child node use:

```
my $n = $meta->child('foo');
$meta->del($n);
```

1.4 Accessing and Changing Node Attributes

Each node has a number of attributes associated with the node which have a simple name, type and value.

To retrieve a named attribute use:

```
my $n = $meta->child('foo');
my $a = $n->arg('bar');
```

To add a new attribute to a node use:

```
my $n = $meta->child('foo');
$n->add_arg('bar','foo');
```

However, to modify an existing attribute (or to add one if missing), use:

```
my $n = $meta->child('foo');
$n->arg('bar','foo');
```

1.5 Node Content

Each node can have a 'content'. This allows for a large amount of text to be associated with the node (It is recommended to use the contents instead of an attribute for large amounts of text).

To retrieve a node’s content use:

```
my $c = $node->content();
```

To set the node’s content use:

```
$node->content('This is the node content text');
```

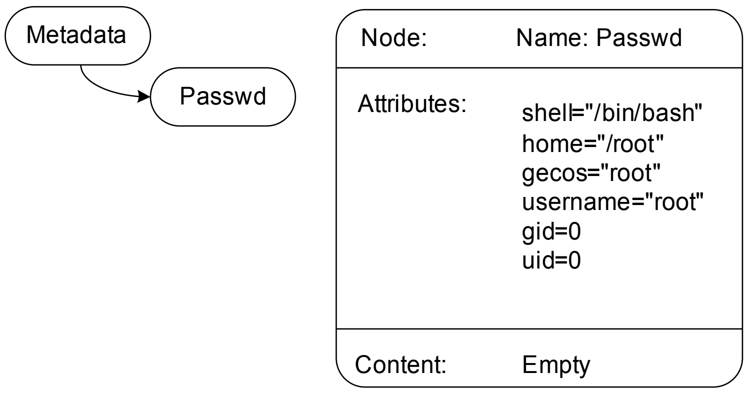
2 Command Control Metadata

Following sections provide description of the existing Metadata node that is passed to the External Scripts. The Metadata node is accessed using the global object, $meta.

2.1 Metadata Structure

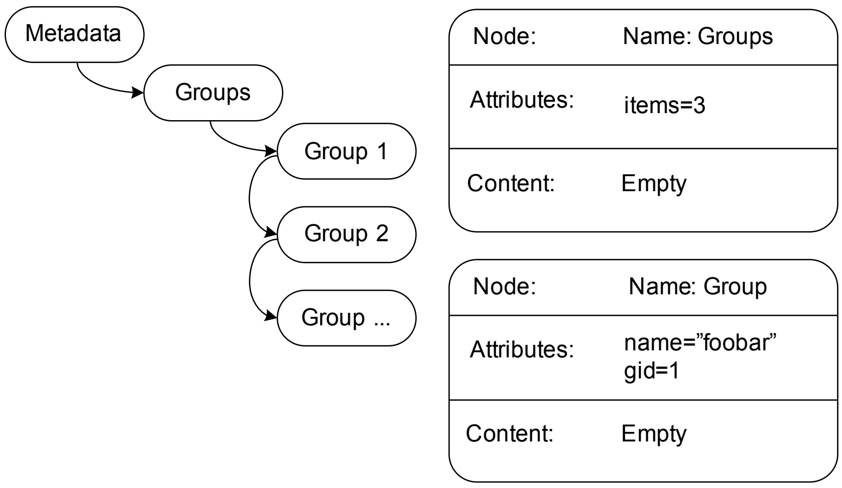
2.1.1 Passwd Node

The passwd node is the Submit Users entry from the Submit Host’s security database. This is for reference only and is not used for authorization purposes by the Command Control Manager.



2.1.2 Groups Node

The Groups node details all the groups that the Submit User is a member on the Submit Host. These groups can be used within Command Contorl User Groups to control access to rules.



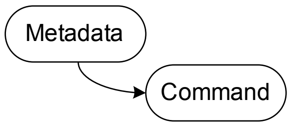
Although, Privileged User Manager can natively check whether a user is in a particular group on the submit host, the following example shows that you can execute the same with Perl.

Example:
To check if a user is in a particular group on the submit host:

```
my @grps=$meta->child('Group')->children('Groups');
foreach my $check (@grps) {
        if($check->arg('name') eq 'staff') {

                return(1);
        }
}

return(0);
```

2.1.3 Command Node

The Command node describes the command that the Command Control Agent is running for authorization process. Once the Command Control Manager has matched the command and executed the External Perl script, this attribute can be overwritten, and will be substituted when the remote host executes the command.



Node:	Name: Command
Attributes:	cmd="ls -F /"
Content:	Empty

Example:
To remove UNIX meta-characters from the command (such as back-quote, less-than, greater-than, ampersand, and so on.)

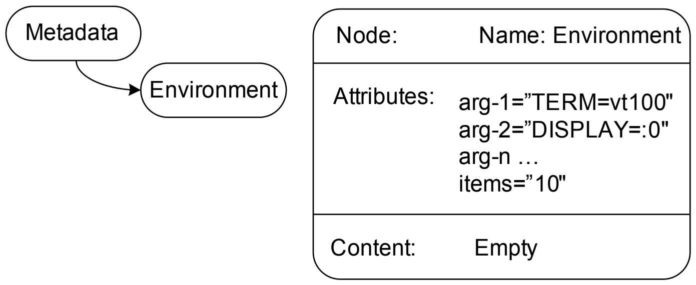
```
my $cmd=$meta->child('Command')->arg('cmd');

$cmd =~ s/\140.*$//g;
$cmd =~ s/([^\\])[\>\<\|].*$/$1/g;
$cmd =~ s/([\;\&]).*$/$1/g;
$meta->child('Command')->arg('cmd',$cmd);
return 1;
```

Example:
To specify which files a user can vi without the individual commands for the rule:

```
my @FileList = ('/tmp/file1', '/tmp/file2');
my $command = $meta->child('Command')->arg('cmd');
my @key = split(/\s/, $command);
my $fname=$key[1];

if((grep(/$fname/, @FileList)) == 0) {
     unauthorize($meta);
} else {
    $command =~ s/^vi /^usvi/g;
    $meta->child('Command')->arg('cmd',$command);
}

return 0
```

2.1.4 Environment Node

The Environment node details the full environment that is passed from the Command Control Agent. All the environment arguments listed will be replicated on the remote host when the command is executed. When adding or removing environment attributes, make sure that the script updates the 'items' attribute with the correct number of environment attributes.



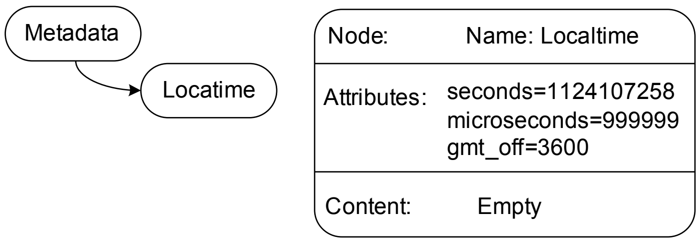
Example:
To delete all unwanted environment variables:

```
my @keeplist = ('TERM', 'DISPLAY');
my $newlist = $ctx->create_node('Environment');
my $enode = $meta->children('Environment');
my $i=0;

foreach my $arg ($enode->arg_values()) {
  my @key = split(/=/, $arg);

  if((grep(/$key[0]/, @keeplist)) != 0) {
        $newlist->arg('arg-' . $i,$arg);
        $i=$i+1;
  }
}

$newlist->arg_int('items',$i);

$meta->del($enode);
$meta->add($newlist);
return 0;
```

2.1.5 Localtime Node

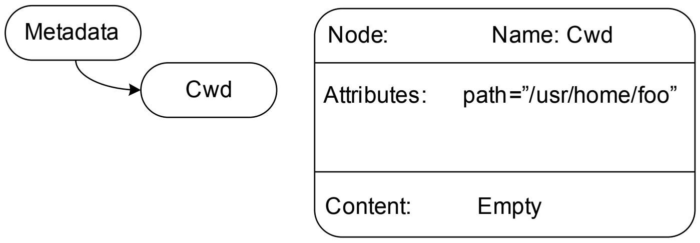
The Localtime node details the local time (including Timezone and Daylight Saving Time) on the submit host. This is the time that is used in the Command Control Manager rule to check access time.

Note, that once the Command Control Manager starts executing the External Perl script, this node has already been matched, and is no longer used. It also has an attribute that details the machines offset from GMT in seconds.



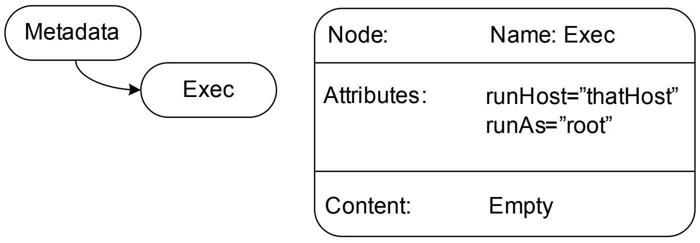
2.1.6 Current Working Directory Node

The Current Working Directory node has one attribute with the Command Control Agent's current working directory when the command was issued. This working directory is used on the remote host to execute the command (if it exists).



2.1.7 Exec Node

The Exec node details the Execute Host and the User that the Command Control Agent requested. These attributes are the ones that are used to match the rule. runHost is then used by the Command Control Agent to resolve the remote host IP address. runAs is used to search the remote user on the remote host to run the command. Changing these attributes will directly change the host or user where the command is running.

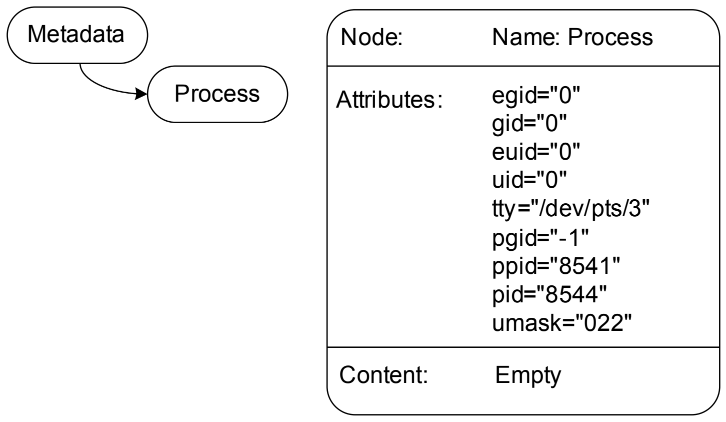


Example:
To change the runHost and runAs directly, and then continue to try to match against the list of rules:

```
$meta->child('Exec')->arg('runAs','root');
$meta->child('Exec')->arg('runHost','foobar');
return 1;
```

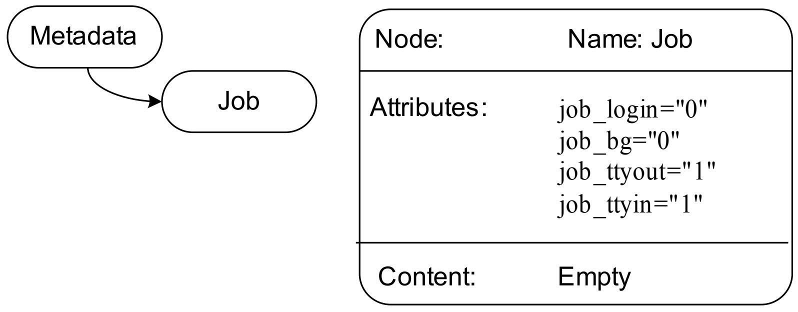
2.1.8 Process Node

The Process node contains the details of the current pcksh/cpcksh privileges, and is used for reference and not for authorization process.



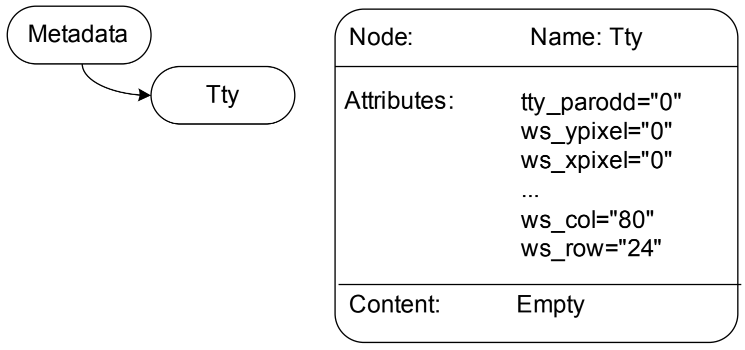
2.1.9 Job Node

The Job node details how the remote command should run. A login process will create a remote terminal, and will execute /etc/profile, etc, in the same way a login shell would. A background process will not allocate a remote terminal, and will not wait for the process to complete. The ttyin/ttyout specifies whether the process has a controlling terminal.



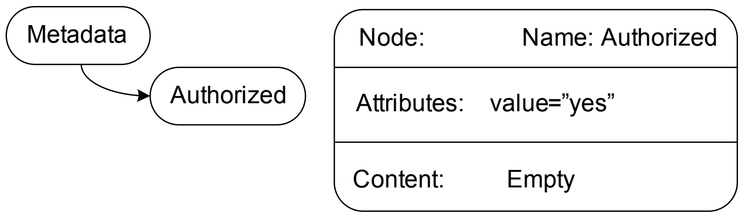
2.1.10 Tty Node

The Tty node holds all the information about the terminal setup for the submit users login shell. For example, the keys that map to character-delete, line-kill and interrupt are all detailed in this node. This node is not involved in the authorization of a command, but is used at the remote host to emulate the user’s terminal.



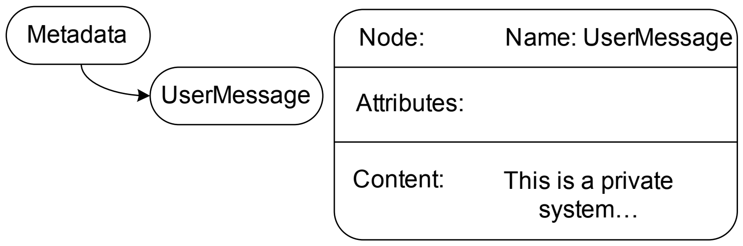
2.1.11 Authorized Node

This node exists to say whether a command is authorized or not. Two functions have been created to authorize or to unauthorize the command (commands are already authorized when they enter the External script because they have already matched the rule).



2.1.12 User Message Node

The User Message node is the text string that is displayed when the submit user tries to run a command. It is displayed even if the command is not authorized. The node is used to change the message if the command was not authorized. Note that the message is stored in the content of the node.

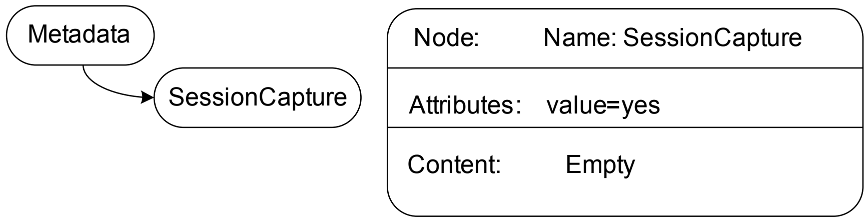


Example:
To change the User Message node use:

```
my $um = $meta->child('UserMessage');
if(! $um) {
     $um=$meta->add_node('UserMessage');
}
$um->content('My new user message');
```

2.1.13 Session Capture Node

The Session Capture node states whether the command input/output should be captured.



Example:
To turn session capture off for a specific user:

```
if($meta->child('Exec')->arg('runAs') eq 'foobar'){
      $meta->child('SessionCapture')->arg('value','no');
}
```

```
    return 0;
```

2.2 External Script Return Codes

External scripts can be defined to be either conditional or action scripts.

A conditional script will be associated with the matching clause on a rule and can either return 1 for match or 0 for no match.

Action scripts are used to perform manipulation of the metadata or to perform actions once a rule has matched. When an action script returns, it can return one of six values which inform the Command Control Manager whether it should continue processing rules, and whether the rule should be marked as applicable.

Return code 1
Next: Continue processing subsequent scripts and rules. (Most simple External scripts should return 1.)

Return code 0
Stop: Stop processing all scripts and rules and return result to client.

Return code -1
Next Sibling: Stop processing this role and sub roles, and continue with the next sibling rule.

Return code -2
Return to Parent: Stop processing this role and sub roles, and continue processing the parent rule.

Return code -3
Stop if authorised: Stop processing all scripts and rules if the current status is authorised.

Return code -4
Stop if unauthorised: Stop processing all scripts and rules if the current status is unauthorised.

2.3 Module Context and Remote Information

External scripts have access to the Command Control Manager's Context object, which allows communication with remote hosts to interrogate it before authorizing a command.

Communication between Privileged User Manager modules is executed using a request/response protocol. The context creates a request object:

```
my $req = $ctx->create_request($ctx->lookup_service('hostname'));
```

Various attributes are then set on the request:

```
        $req->module('sysinfo');
        $req->method('dirList');
```

Then it is sent, and a response retrieved:

```
    my $resp = $ctx->remote_request($req);
```

The response will have the data requested, or will have an error number and error text attached.

3 Metadata Table of Contents

Section	Variable	Description
Passwd	`shell='/bin/bash'`	The /etc/password entry of the submit user
	`home='/root'`	
	`gecos='root'`	
	`passwd='x'`	
	`username='root'`	
	`gid='0'`	
	`uid='0'`	
Groups	`items='3'`	A list of the groups that the submit user is a member of
	`name='wheel' gid='10'`	Group name and GID
	`name='bin gid='1'`	
	`name='root' gid='0'`	
Command	`cmd='ls -l'`	The full command
Host	`name='shuttle'`	The submit host
Environment	`items='7'`	The complete user environment variable list
	`arg-6='LOGNAME=root'`	
	`arg-5='TERM=xterm'`	
	`arg-4='HOME=/root'`	
	`arg-3='USER=root'`	
	`arg-2='SHELL=/bin/bash'`	
	`arg-1 ='PATH=/sbin:/bin:/usr/sbin:/usr/bin'`	
	`arg-0='PS1=# '`	
Localtime	`gmt_offset='0'`	The localtime that the process is running in (including the current offset from GMT)
	`microseconds='374088'`	
	`seconds='1134738518'`	(seconds from 1/1/1970)
Cwd	`path='/tmp'`	The current working directory of the process
Options	`print_motd='1'`	Print the MOTD if it is a logon process
	`print_last_login='1'`	Print the 'last logged on' message if a logon process
	`loglevel='0'`	N/A
	`groupID='+J64hG/Jlpsu'`	A unique session for Session Capture
Exec	`runHost='shuttle'`	The Execute host

Section	Variable	Description
	`runAs='root'`	The Execute user
Logon	`pid='16465'`	The process ID of the users logon shell (man who)
	`from='asia'`	Where the user logged on from
	`when='1134738215'`	Time when user logged on (seconds from 1/1/1970)
Process	`umask='18'`	Process umask
	`egid='0'`	Process Effective UID
	`gid='0'`	Process Group ID
	`euid='0'`	Process Effective GID
	`uid='0'`	Process UID
	`tty='/dev/pts/5'`	Tty of the process
	`pgid='16629'`	Process Group
	`ppid='16628'`	Parent Process ID
	`pid='16629'`	Process ID
Job	`job_login='0'`	Perform proper user logon at remote end
	`job_bg='0'`	Perform job in background
	`job_ttyout='1'`	Terminal attached to output
	`job_ttyin='1'`	Terminal attached to input
Authorized	`value='yes'`	Was the job authorized
SessionCapture	`value='yes'`	Is the session captured
UserMessage	`'message'`	Optional user message displayed before running command
Tty	`tty_parodd='0'`	User terminal setup (see man 'stty')
	`tty_parenb='0'`	
	`tty_cs8='1'`	
	`tty_cs7='1'`	
	`tty_onlret='0'`	
	`tty_onocr='0'`	
	`tty_ocrnl='0'`	
	`tty_onlcr='1'`	
	`tty_olcuc='0'`	
	`tty_opost='1'`	
	`tty_pendin='0'`	
	`tty_choke='1'`	
	`tty_echoctl='1'`	
	`tty_iexten='1'`	
	`tty_tostop='0'`	
	`tty_noflsh='0'`	
	`tty_echonl='0'`	
	`tty_echok='1'`	
	`tty_echoe='1'`	

Section	Variable	Description
	tty_echo='1'	
	tty_xcase='0'	
	tty_icanon='1'	
	tty_isig='1'	
	tty_imaxbel='0'	
	tty_ixoff='0'	
	tty_ixany='0'	
	tty_ixon='1'	
	tty_iuclc='0'	
	tty_icrnl='1'	
	tty_igncr='0'	
	tty_inlcr='0'	
	tty_istrip='0'	
	tty_inpck='0'	
	tty_parmrk='0'	
	tty_ignpar='0'	
	tty_vdiscard='15'	
	tty_vlnext='22'	
	tty_vwerase='23'	
	tty_vreprint='18'	
	tty_vsusp='26'	
	tty_vstop='19'	
	tty_vstart='17'	
	tty_veol2='0'	
	tty_veol='0'	
	tty_veof='4'	
	tty_vkill='21'	
	tty_verase='8'	
	tty_vquit='28'	
	tty_vintr='3'	
	tty_vtime='0'	
	tty_vmin='0'	
	tty_ibaud='9600'	
	tty_obaud='9600'	
	ws_ypixel='0'	
	ws_xpixel='0'	
	ws_col='80'	Terminal number of columns
	ws_row='24'	Terminal number of rows

For debugging a Rule an easy way to display the contents of the Metadata is to write a simple script to display the metadata in the log. Each time a request is processed the script will write the metadata into the unifid.log file.

Example:

```
    $ctx->log_info($ctx->node2xml($meta));
```