Team Engineering Services

Developing for the CA UIM platform

SDK Deep Dive Partners – 02-Jun-2016

Marcio.rodrigues@ca.com

ca technologies

Agenda

- **Introduction – Engineering Services Team**
- **Module 1** What we Can and Can’t do…
- **Module 2** Overview of API’s and resources
- **Module 3** Developing custom probes

Engineering Services Team

Engineering Services Team

Field Solutions – Recent development

- DCIM
- Generic_cluster
- Service_Discovery
- Alarm_enrichment
- MCS

What we Can and Can’t do…

What we Can and Can’t do…

- We can create messages on the message bus
  - QoS, alarm & custom messages
- We can create custom probes
- We can distribute probes
- We can update probe config files (add new profiles)
- We can query hub / probes for data
- List of hubs and robots (callbacks)
- Anything Infrastructure Manager can do
- We can enrich the messages
  - Alarm_enrichment

What we Can and Can’t do…

Open New Possibilities

- We can create Accounts
- We can create Contacts
- We can get QoS data (polling)
- We can get alarm data (polling)
- We can get sla’s
- We can get slo’s

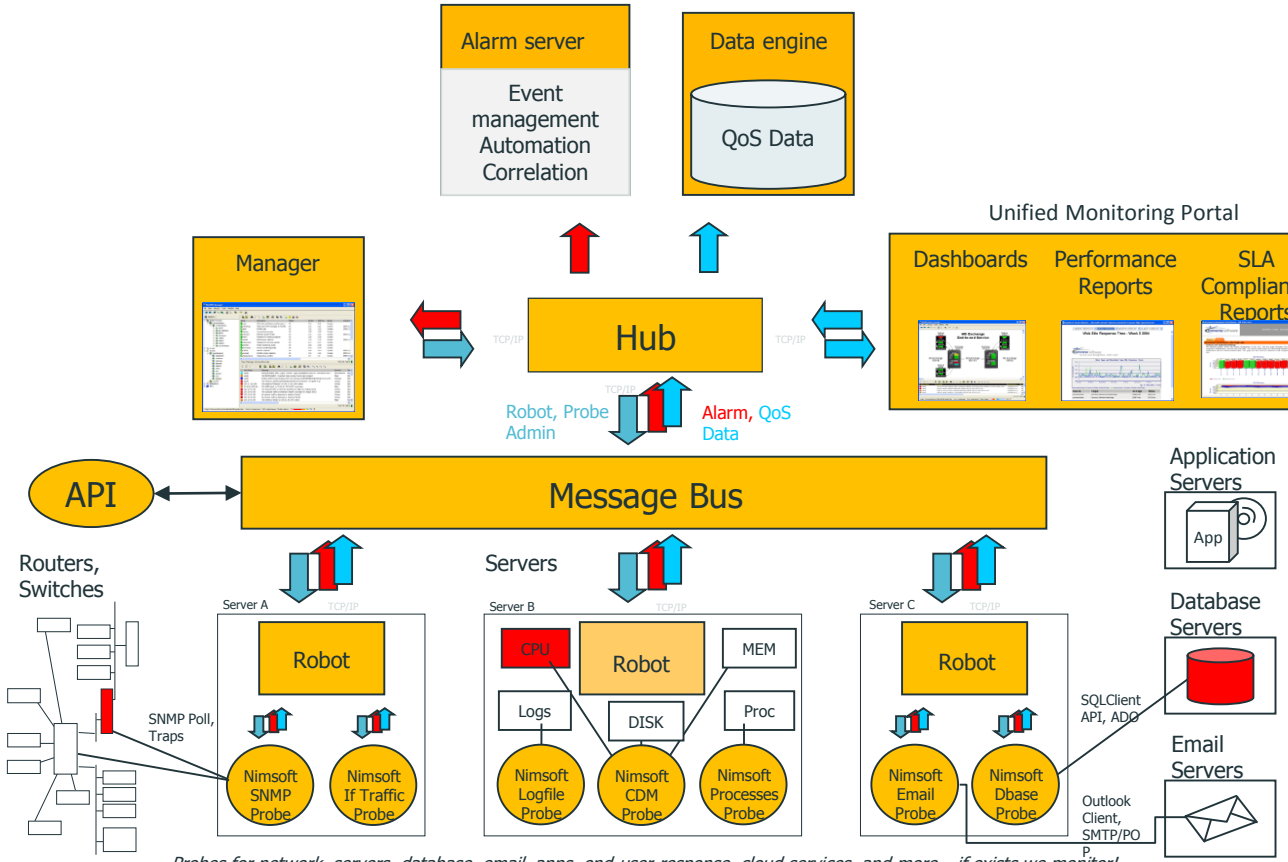
What we Can and Can’t do…

- We can’t do anything GUI related
  - We can’t access portlets
  - We can’t create portlets
  - We can’t create list views

Custom Probes Examples

- Generic Cluster
  A probe that can failover probe profiles from one robot to another.
- balanceRobots
  “Balances” the load on hubs by moving robots from busy hubs to other hubs.
- Alarm_enrichment
  Enrich CA UIM alarms with additional information read from external data source (in alarm out alarm2)
- SSR / MCS
  Monitoring configuration Service - Profiles

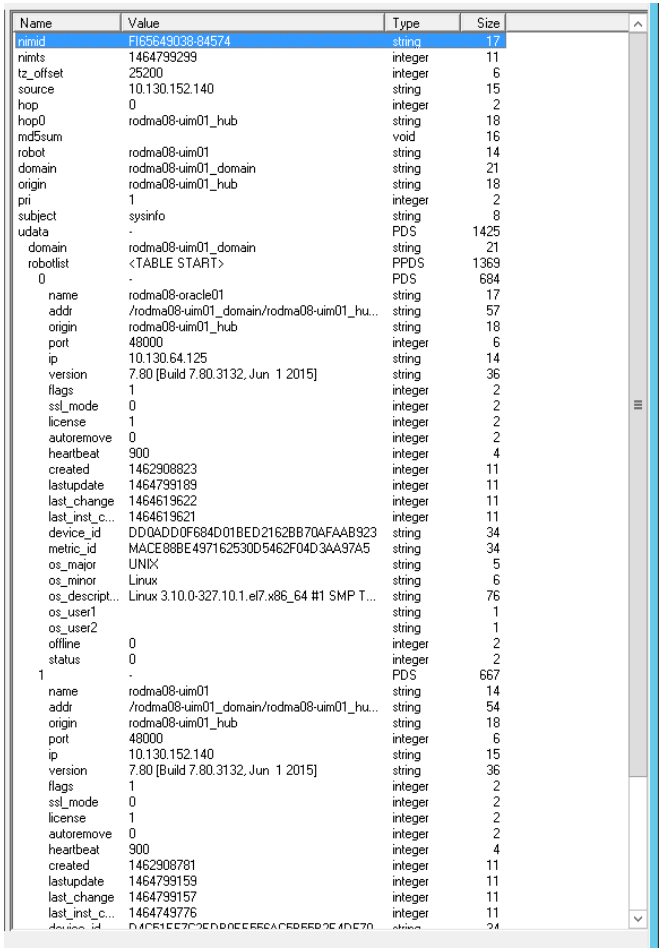
CA UIM architecture



Probes for network, servers, database, email, apps, end-user response, cloud services, and more… if exists we monitor!

ca technologies

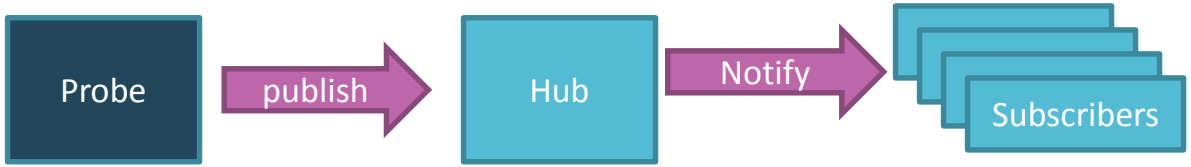
PDS – Portable Data Stream



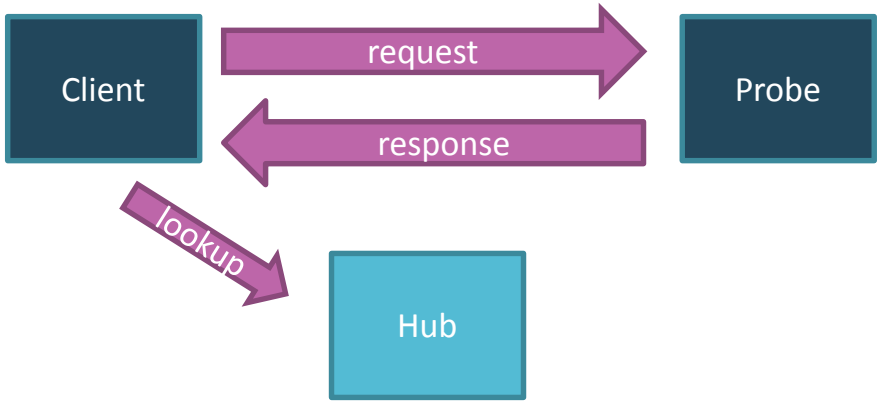
Name	Value	Type	Size
nimid	FI65649038-84574	string	17
nimts	1464799299	integer	11
tz_offset	25200	integer	6
source	10.130.152.140	string	15
hop	0	integer	2
hop0	rodma08-uim01_hub	string	18
md5sum		void	16
robot	rodma08-uim01	string	14
domain	rodma08-uim01_domain	string	21
origin	rodma08-uim01_hub	string	18
pri	1	integer	2
subject	sysinfo	string	8
udata	-	PDS	1425
domain	rodma08-uim01_domain	string	21
robotlist	<TABLE START>	PPDS	1369
0	-	PDS	684
name	rodma08-oracle01	string	17
addr	/rodma08-uim01_domain/rodma08-uim01_hu...	string	57
origin	rodma08-uim01_hub	string	18
port	48000	integer	6
ip	10.130.64.125	string	14
version	7.80 [Build 7.80.3132, Jun 1 2015]	string	36
flags	1	integer	2
ssl_mode	0	integer	2
license	1	integer	2
autoremove	0	integer	2
heartbeat	900	integer	4
created	1462908823	integer	11
lastupdate	1464799189	integer	11
last_change	1464619622	integer	11
last_inst_c...	1464619621	integer	11
device_id	DD0ADD0F684D01BED2162BB70AFAAB923	string	34
metric_id	MACE88BE497162530D5462F04D3AA97A5	string	34
os_major	UNIX	string	5
os_minor	Linux	string	6
os_descript...	Linux 3.10.0-327.10.1.el7.x86_64 #1 SMP T...	string	76
os_user1		string	1
os_user2		string	1
offline	0	integer	2
status	0	integer	2
1	-	PDS	667
name	rodma08-uim01	string	14
addr	/rodma08-uim01_domain/rodma08-uim01_hu...	string	54
origin	rodma08-uim01_hub	string	18
port	48000	integer	6
ip	10.130.152.140	string	15
version	7.80 [Build 7.80.3132, Jun 1 2015]	string	36
flags	1	integer	2
ssl_mode	0	integer	2
license	1	integer	2
autoremove	0	integer	2
heartbeat	900	integer	4
created	1462908781	integer	11
lastupdate	1464799159	integer	11
last_change	1464799157	integer	11
last_inst_c...	1464749776	integer	11
device_id	D4C515E7C2EDB0E5E5AC5B5EB2E4DE70	string	34

Types of communications

- Pub/sub



- Request/Response



Pub/Sub Messaging

- A message is “published” on the message bus
- No destination address is supplied
- The message has a “subject” (alarm, QOS_MESSAGE, custom)
- Processing is asynchronous
- Programs can subscribe to messages with one or more subjects
  - E.g. NAS subscribes to “alarm” subject (alarm2)
- Message is sent through the local spooler to the hub
- The hub can “queue” messages in an “attach” queue
- Pub/sub is used for alarms, qos metrics, emails, etc.

Pub / Sub Utilities

Default install dir - C:\Program Files (x86)\Nimsoft\bin

- Testalarm
  - Windows client utility to send an alarm or user-defined message
- Nimalarm
  - Command-line utility to send an alarm
- Nimqos
  - Command-line utility to send a QoS metric
- Dr. Nimbus
  - Multi-purpose Windows client utility
  - Sniffs the bus and shows messages
  - Attaches to queues

Request/response messaging

- A “request” message is sent to a probe
  - Keep in mind that everything in CA UIM is a probe, including the robot, hub, nas, etc.
- Probe is addressed using the CA UIM address space
  - /Domain/Hub/Robot/Probe
  - Socket can also be used, e.g. hostname:48000
- Message consists of Address, Callback and optional payload in a PDS
- The probe sends back a response PDS
- Processing is synchronous
- Used by probe gui’s
- Access control is enforced by API’s

Request/response utilities

- Pu
  - Command-line “Probe utility” to invoke callbacks
- Probe utility
  - CTRL-P
  - Invoke callbacks

Summary

- Everything is a message (a PDS)
- Two types of communications:
  - Publish/subscribe
  - Request/response
- All CA UIM programs run as probes

Exercises

- Use Dr. Nimbus to look at messages on the bus
- Look at header and udata for a QOS_MESSAGE
- Invoke callbacks using Probe Utility
  - Read the loglevel of the probe via a callback
  - Get_info, get_hub … get…

Overview of API’s and resources

CA UIM Current APIs

- .NET API
- C SDK
- Java SDK
- nas Extensions to Lua
- Perl SDK
- Probe Software Developer Kit Guide
- RESTful Web Service

API’s and resources

- Codewizard
- Documentation for all API’s: click “help” in codewizard
  - Opens docs/sdk.chm
- Additional documentation for specific API’s:
  https://docops.ca.com/ca-unified-infrastructure-management/8-4/en/development-tools

API calls

- Send an alarm
- Send QoS metric
- Use configuration file
- Use logging
- Subscribe to messages / attach to queue
- Register callback
- Invoke callback

Exercise

- Deploy the SDK
- Explore the Documentation examples
- https://docops.ca.com/ca-unified-infrastructure-management/8-4/en/development-tools
- ALL code shared https://github.com/marciokugler/cadeepdive2016

Exercise

- Install code_wizard
- Check documentation, try to run an example
- Review the training course files

Send an alarm

- Perl

```
use Nimbus::API;
nimAlarm(NIML_MAJOR,“Message");
```

- Java

```
import com.nimsoft.nimbus.NimAlarm;
NimAlarm alarm = new NimAlarm(NimAlarm.NIML_WARNING, “Message");
alarm.send();
```

- LUA

```
nimbus.alarm(NIML_MAJOR,“Message","probe.checkpoint.id","1.1.1")
```

- .Net

```
Alarm alarm = new Alarm("Test alarm from .NET", SeverityLevel.Information);
PDS ret = session.SendMessage(alarm);
```

Exercise - Exercise01.pl

- Run the script
- Change the Message

Perl API Documentation

https://docops.ca.com/ca-unified-infrastructure-management/8-4/en/files/289572835/PERL--Perl+SDK_12_14.pdf

Subscribe to Messages – Exercise02.pl

- See Exercise02.pl File
- Discussion

Use configuration file input

- Perl

```
$config       = Nimbus::CFG->new("$prgname.cfg");
$loglevel   = $config->{setup}->{loglevel} || 0;
```

```
<setup>
        logfile = sqlserver_monitor.log
        timeout_hb = 30 sec
        qos_group = QOS_SQLSERVER
        loglevel = 0
        QoS_V2_compatibility = no
        logsize = 100
</setup>
```

Use Logging

- Set the loglevel
  - Usually defined in the configuration file
- Invoke NimLog.log(level,text) method
- Perl:

```
nimLogSet($logfile,$prgname,$loglevel,0);
nimLog(0,"---------------- Starting  (pid: $$) -----------------");
```

- Java:

```
NimLog logger = NimLog.getLogger(this.getClass());
NimLog.setLogLevel(NimLog.WARN);
logger.log(NimLog.WARN,"A waring message");
logger.log(NimLog.FATAL,"This one will always be written");
```

Register callbacks

- Must be a daemon probe (not timed)
- Define the session
- Register the callback

```
$sess = Nimbus::Session->new("$prgname");
$sess->setInfo($version,"Nimsoft Software AS");

if ($sess->server (NIMPORT_ANY,\&timeout,\&restart)==0) {
   $sess->addCallback ("hello", "arg1,arg2_str,arg3_num%d");
}else {
   nimLog(0,"unable to create server session");
   exit(1);
}
nimLog(0,"Going to dispatch the probe");
$sess->dispatch();
```

Exercise - Exercise03.pl

- Explore documentation for the language of your choice
  - Find out how to create and populate a PDS
  - Dump a PDS example Exercise03.pl
  - Try a different PDS message

Exercise

- Use codewizard and Textpad to create and run a probe
- You MUST add:
  - use lib "c:\\program files (x86)\\nimsoft\\perllib\\";
  - Modify Message of Exercise01.pl change severity
  - Use documentation

Types of probes

- Timed probe
  - Probe starts, does work, ends
  - Controller starts probe every n minutes
  - Controller kills probe if not finished in n minutes
- Daemon probe
  - Probe is always running
  - Interval processing is handled in code
  - Controller restarts probe if it ends unexpectedly

Structure of a daemon probe (Perl)

```
sub doWork {
# My code
}
sub restart {
}
sub timeout {
   doWork();
}
sub ctrlc {
   nimLog(0,"Got a control-C so am restarting");
   exit;
}
# MAIN ENTRY
$sess = Nimbus::Session->new("$prgname");
$sess->setInfo($version,"Nimsoft Software AS");

if ($sess->server (NIMPORT_ANY,\&timeout,\&restart)==0) {
   $sess->addCallback ("hello", "arg1,arg2_str,arg3_num%d");
}else {
   nimLog(0,"unable to create server session");
   exit(1);
}
$sess->dispatch();
exit;
```

Generating alarm messages

- Suppressionid causes NAS to auto-suppress previous alarms with same suppressionid & source.
- Constructor

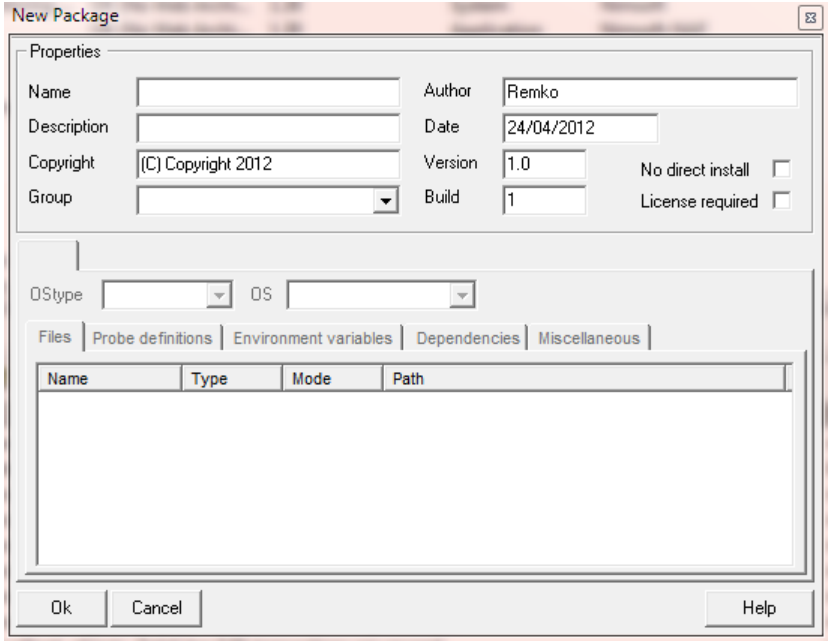
```
NimAlarm(severity,
         message,
         subsystem,
         suppressionid,
         source);
```

Lab exercise – Exercise01.pl

- Send alarms with suppression key
- Verify suppression
- Add logging
- The Level constants:
  - NIML_CLEAR (0)
  - NIML_INFO (1)
  - NIML_WARNING (2)
  - NIML_MINOR (3)
  - NIML_MAJOR (4)
  - NIML_CRITICAL (5)

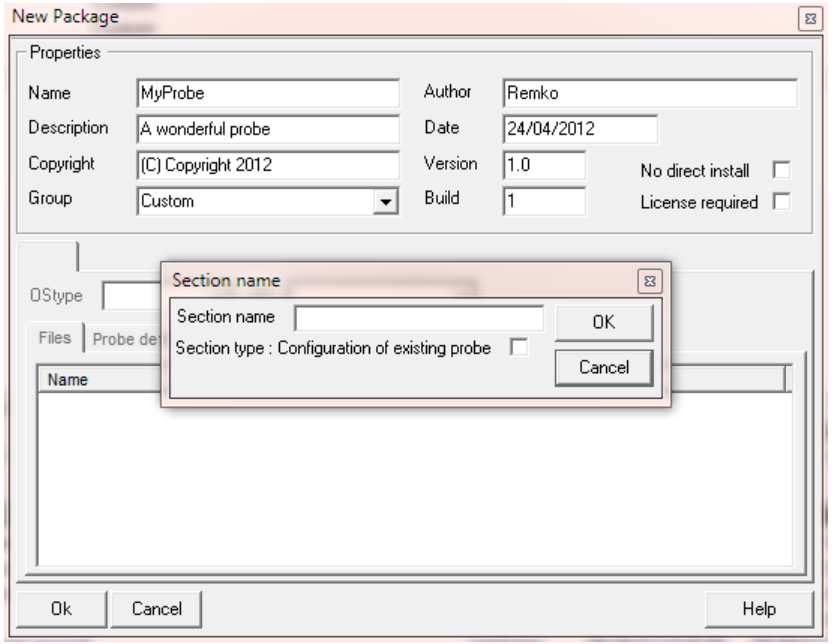
Packaging a probe for distribution

- Use infrastructure manager – Archive
- Right-click and select new, fill in the fields



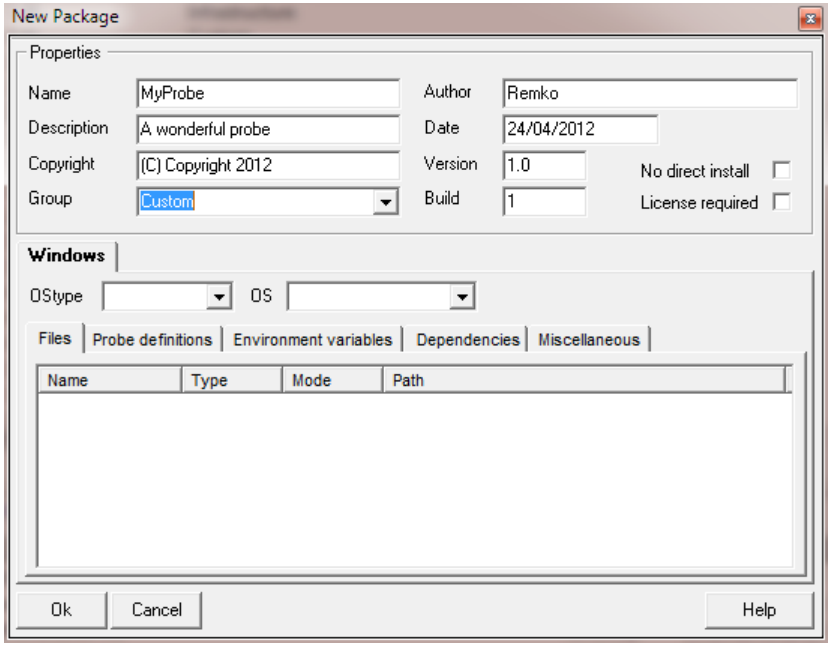
Packaging a probe

- Right click on the empty tab, and select Add Section



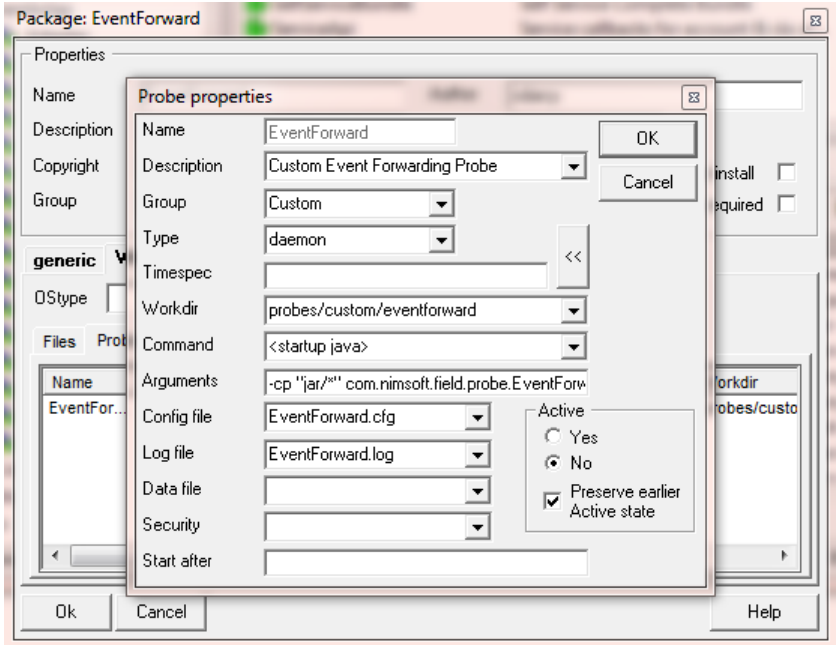
Packaging a probe

- Section for OS
  - Take care with 32 vs 64 bits!
- Add files
  - Jar files
  - Drag or import
- Add probe definition



Add probe definition

- Daemon or timed?
- Command: your executable or <startup java>
- Configuration file
- Log file
- Security



What's a NMS client?

- Any program that invokes NMS services, usually from outside the NMS ecosystem
- Examples:
  - Probe GUI's
  - Tools to automate probe configuration
  - Portal to show alarm status
- A client needs to authenticate with NMS
  - Only NMS users, not account contacts, can do this
  - The client needs access to a robot (can be remote)

Clients invoke callbacks

- Define a session object
  - Pass address of probe and callback name
  - Optionally pass PDS with parameters
  - Invoke nimNamedRequest
  - Output is returned in PDS format

```
#!perl

use lib "c:\\program files (x86)\\nimsoft\\perllib\\";
use Nimbus::API;
use Nimbus::Session;
        my($sid) = nimLogin("administrator", "nimbus");
        my $pds = pdsCreate();
        psd
        r($iet,$pds_out) = nimNamedRequest("controller", "gethub", $pds , 90);
        if ($iret == 0) {
           my $domain = pdsGet_PCH($pds_out,"domain");
           print("Domain: $domain\n");
        }
```

Exercise 4 – Exercise04.pl

- Write a probe to run a different callback

Clients invoke callbacks

- Instantiate a NimRequest object
  - Pass address of probe and callback name
  - Optionally pass PDS with parameters
  - Invoke NimRequest.send()
  - Output is returned in PDS format

```
NimRequest request = new NimRequest("controller","gethub");
     PDS pdsout = request.send();

     String hubaddress =
        "/" + pdsout.getString("hubdomain") +
        "/" + pdsout.getString("hubname") +
        "/" + pdsout.getString("hubrobotname") +
        "/hub";
```

Lab exercise – Exercise02.pl

- Write a probe that implements a callback that receives a PDS and dumps the PDS to stdout
  - Hint: use Perl: pdsDump, Exercise02
- Write a client that invokes the callback
- Start the probe on the cmd prompt.

Invoking callbacks

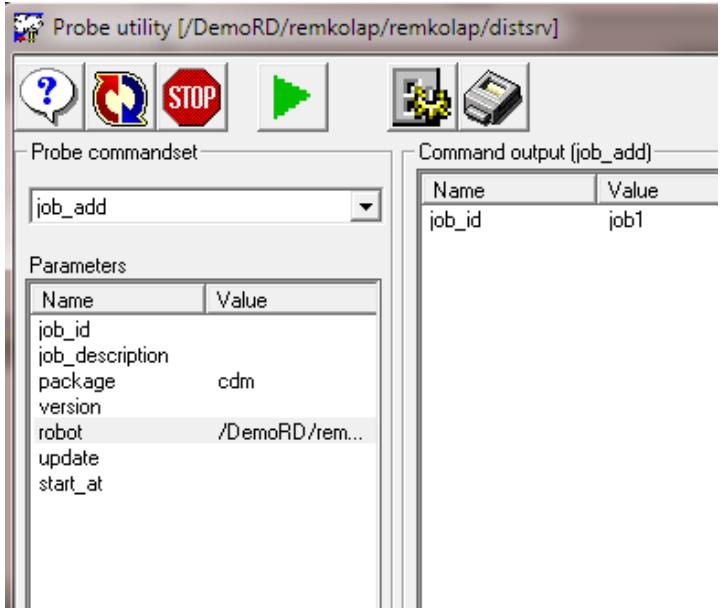
- You'll want to invoke callbacks on
  - Controller – to configure probes
  - Distsrv – to install probes
  - Hub – for directory services
  - Nas – to get alarms
  - Your own probes

Some useful callbacks - controller

- Controller
  - Probe_config_get
    - Reads a configuration file
  - Probe_config_set
    - Writes a configuration file
    - One key at a time, but can specify multiple keys in as_pds PDS.
  - Text_file_get, text-file-put
    - Basic ascii file transfer
  - File_get_start, file_get_next, file_get_end
    - More flexible file transfer

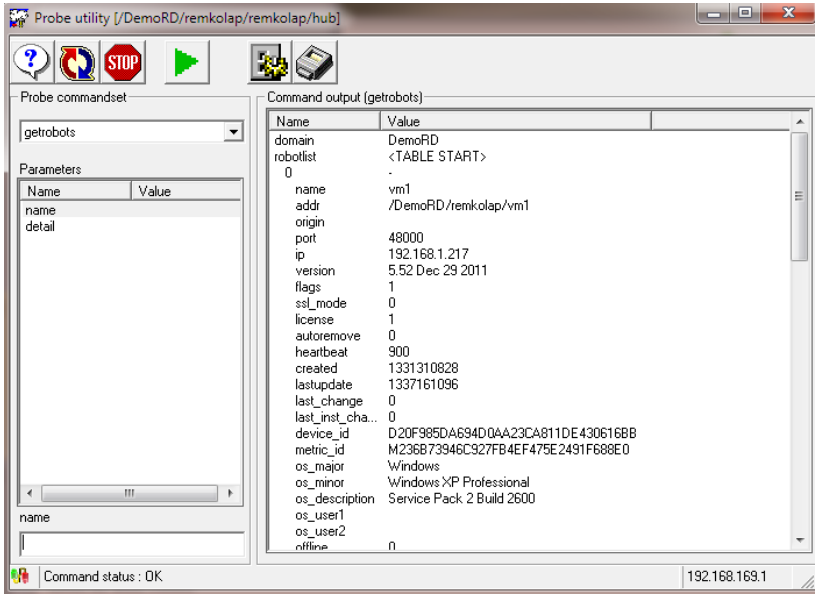
Some useful callbacks - distsrv

- Install a probe
  - Job_add



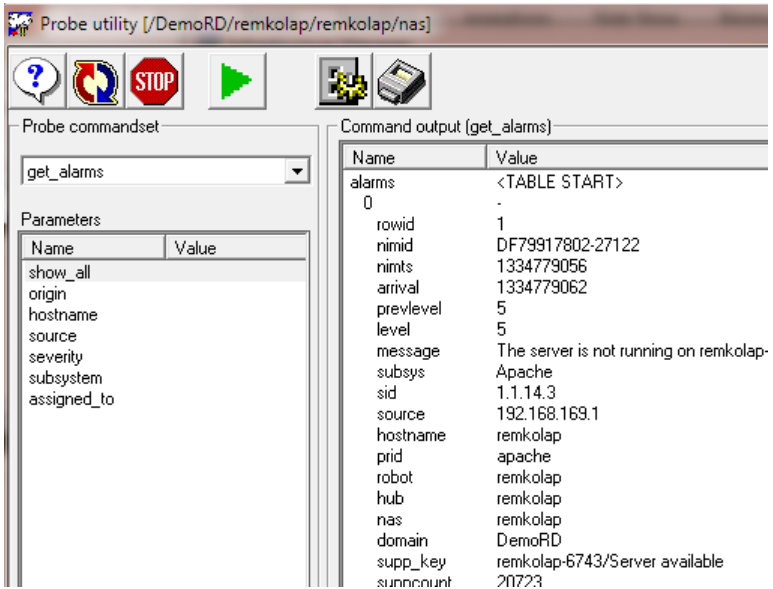
Some useful callbacks - hub

- Gethubs
  - Find out what other hubs there are
- Getrobots
  - List the robots connected to this hub



Some useful callbacks - nas

- Get_alarms
- Assign_alarms
- Close_alarms
- Create_note
- Attach_note



Lab exercise – Exercise05.pl

- Write a client that distributes a probe
- Don’t forget to log in!
- Modify the example, what changes are needed?

Bonus Exercise

- Write a client that lists alarms in NAS

Lab exercise: develop a probe

- Write a probe to monitor process
- Send an alarm if process is down
- Write a probe to count number of files in the directory
- Send an alarm if number of files greater than 3

Marcio.rodrigues@ca.com

Q&A