CodeMagus

orkhestra_attproc: Orkhestra: Attachments to Processes Version 1

CML00117-00

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

CodeMagus

July 2, 2021

Contents

1 Introduction

Design Requirements:

- Remove the requirement to configure/specify and allocate specific ports to the processes involved in an NFT.
- Remove the requirement to clean up processes manually once an NFT completes or has to be restarted.
- Remove the requirement to specify the number of proxies, etc.
- Remove the clash between ephemeral and allocated ports.
- Build on the various structures in place controlling remote agents etc. Local configurations to support processes started by remote requests.
- Direct feedback to orkhestra to know the status of starting, started and failed processes.
- Reporting to the user the status of processes that have been requested to start, running and failed.

Processes:

- A process name will be defined, and that process name will relate to a configuration. That configuration can appear on multiple machines.
- When the daemon receives a SIGHUP signal, the daemon should clean up all groups active, by sending a kill signal to the attached processes.
- The name of a process will have the form of an identifier and could also be the program name.
- Included in the local configuration of a named process are the following attributes:
  - Path to the executable on each specific machine.
  - Name of the executable on each specific machine.
  - Name (and possibly group) to execute the process on each specific machine.
  - Working directory to be set when running the process on each specific machine.
  - Environment variable settings to be set up when running on each specific machine.
  - A process may indicate that it can be restarted. If this is the case, the process will be restarted after an unexpected exit. The client of the specific process should be written in such a way that re-connections are possible.
- Once a group has started, the client can instruct the daemon to start processes on its behalf.
- Starting a process will follow these steps.
  - The daemon listening on an assigned port will be contacted with the start process request.
  - The act of contacting the daemon will start a control group of processes. This is to manage the cleanup of processes.
  - If the connection to the daemon is broken or lost, then that is to be interpreted as a signal that the controlling process has been lost and that there is no more

interest in the processes associated with that group. There are variants to this requirement which we can discuss, for example, when a connection is made from a requesting process (say Orkhestra), the connection can establish itself with a new request or an existing request (that is, signalling a re-connection). In the case of a re-connection, processing continues from where it left off. If this is the model to follow then when a client introduces itself to the daemon it indicates a CONNECT or RECONNECT. In the case of CONNECT, the daemon supplies a group/session handle. and in the case of RECONNECT the client supplies the previously assigned handle. If the handle is not valid then the daemon responds with an error message (orkhestra should display this message and terminate).

- A process start request proceeds as follows.
  - The requesting process (e.g Orkhestra) opens a specific call-back port and makes this a listen port. The port number does not have to be fixed or universally known. It could also be an ephemeral port. This port will be referred to as the connect-back port.
  - In addition to the connect-back port, a connect-back address will also need to be supplied.
  - The request to start a remote process will include the connect-back port number.
  - On receipt of a start request, the daemon interprets the configuration for the process.
  - The daemon does a fork to start another process.
  - The daemon sets up the environment variables.
  - The daemon does a setuid/setgid to the indicated userid and group.
  - The daemon exec's the child process.
  - The connect-back address and port are made available in a standard location. For example, environment variables CODEMAGUS_ORKHESTRA_CONNECT_BACK_HOST and CODEMAGUS_ORKHESTRA_CONNECT_BACK_PORT (something clear and unambiguous).
  - The child process extracts the connect-back address and connect-back port and establishes a connection to the running Orkhestra instance.
  - The connection back to the running Orkhestra should not be done until the child process wants to indicate to the running Orkhestra that it is ready.
  - The daemon makes a record of the child PID and associates this PID with the group. When reaping child processes that terminate, the PID is used to determine which process has terminated and whether a restart is required. The PID is also used when a group is terminated.
- If a re-connection is not made within a specified time interval, then the daemon can assume that all interest in the started processes is lost. In this case, the daemon should kill the processes started as part of the identified group.
- When terminating a child process within a group as part of the group cleanup, the

daemon will send an appropriate kill signal to the child processes. It may further be an idea, as indicated in the configuration file that a specific signal should be used.

- The daemon should not close connections from clients (such as Orkhestra) under normal conditions as this is an indication that there is still interest in the group.

2 Processes Daemon

orkstrprd is the *orkhestra* (orkhestra: Configuration and User Reference Version 1[1]) attachments to processes daemon.

2.1 Synopsis

If *orkstrprd* is started with the parameter '`--help`' it will only display the command line parameters it accepts and a short description of each. The list of parameters along with a complete description of each can be found in the following section.

```
Usage: orkstrprd [OPTION...]
  -p, --port=<port>                  Port to listen to
  -c, --configuration=<file name>    Configuration file
  -v, --verbose                      Verbose processing

Help options:
  -?, --help                         Show this help message
      --usage                        Display brief usage message
```

2.2 Parameters

- '`-p|--port`' Specifies the port number to listen on for incoming connections from clients
- '`-c|--configuration`' Specifies the configuration file for *orkstrprd*, see section 2.3 on page 7.
- '`-v|--verbose`' Directs *orkstrprd* to output diagnostic information during execution.

2.3 Configuration

This section describes the configuration file required by *orkstrprd*. It is specified by the command line option '`-c|--configuration`'.

2.3.1 Elements

The elements of the commands to *orkstrprd* comprise reserved words, identifiers, string literals, comments, numbers and integers. The configuration grammar is free format and white spaces have no grammatical meaning except where they might appear within string literals.

Comments are introduced by using a

- hash ('#') and continue up to the end of the current input line.
- double minus ("--") and continue up to the end of the current input line.

Examples:

```
# orkhestra_attproc.tex
--
-- Environment variables:
#
```

Reserved Words have a special meaning in terms of directing the parsing of commands. The reserved words are:

`arg`	`argument`	`default`	`define`
`description`	`directory`	`external`	`group`
`include`	`log`	`machine`	`mkdir`
`path`	`process`	`userid`	`restart`
`script`	`var`	`variable`	`work`

Table 1: *orkstrprd* reserved words

Identifiers are case sensitive, they start with a letter may be followed by any number of letters, digits or the under-score character.

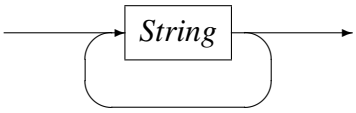
Examples:

```
think_time  ncacrag_123   RecordStaffArrgmntDet
```

Strings are:

- any sequence of characters (except double quotes and the newline character) enclosed by double quotes.
- any sequence of characters (except single quotes and the newline character) enclosed by single quotes.

Strings cannot span source text lines, but they may be concatenated:

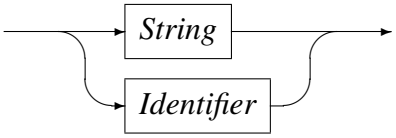


Examples:

```
"\texttt{GigabitEthernet0/0 In Octets"
'$Revision: 1.4 $'
```

Filenames

Filename



A *Filename* is usually written as a *String* but may also be an *Identifier*. Most importantly a *Filename* must conform to any constraints of the underlying file system.

Numbers consist of a nonempty sequence of decimal digits that

- possibly contains a radix character (decimal point '.').
- is optionally followed by a decimal exponent; consisting of an 'E' or 'e' followed by an optional plus or minus sign followed by a nonempty sequence of decimal digits that indicates multiplication by a power of 10.

Examples:

```
1234
0.001
1.2
123.45E-12
```

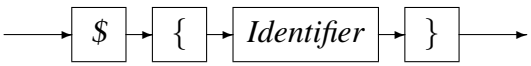
Integers consist of a nonempty sequence of decimal digits '0' through '9'. Integers are a sub set of numbers.

Examples:

```
1234
0
```

Environment Variables

EnvironmentVariable

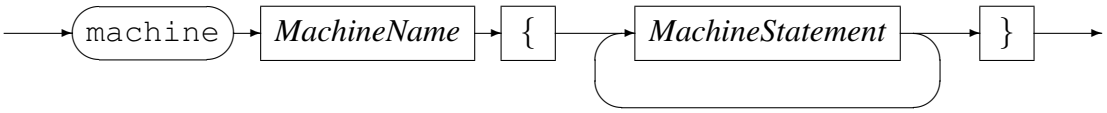


Environment variables are substituted with their value when encountered in any input text. Note: the environment in *orkstrprd* is closed and does not include the environment the daemon inherited.

2.3.2 Machine

The configuration file defines the machine, name and its environment. It also specifies a directory that may contain configuration files to define the NFT groups and their processes.

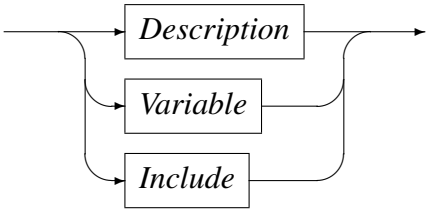
Machine



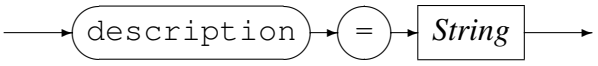
MachineName

MachineName identifies the name of the machine in this configuration.

MachineStatement

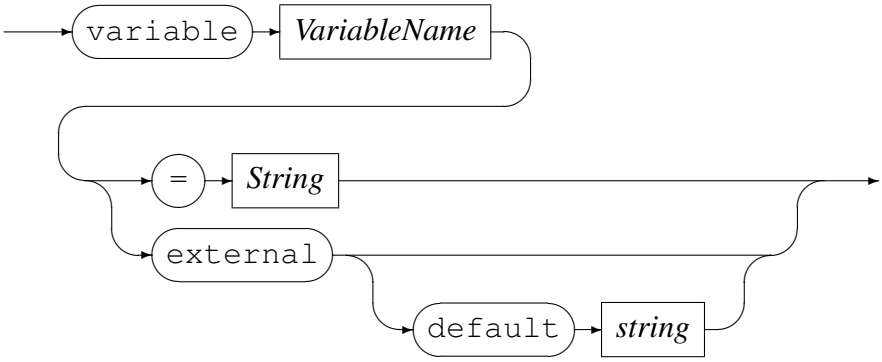


Description

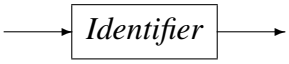


The description provides a mechanism for further documenting the machine and its purpose.

Variable



VariableName



A *Variable* is either assigned to

- a *String* value.
  Note: Environment variables in strings are expanded - see section 2.3.1 on page 9.
- To the value of an environment variable inherited by the daemon.
  If the external environment variable is not defined it will be assigned the default value, if specified.

Configuration directory

This directory holds the NFT group configuration files and all files in this directory will be parsed. Breaking up the daemon configuration into smaller chunks makes it more manageable.

Example:

```
external var CODEMAGUS_HOME default "/usr/local/CodeMagus"
include directory = "${CODEMAGUS_HOME}/orkstrprd/orkstrprd.d"
```

Assuming `CODEMAGUS_HOME` is set to the default value, the configuration directory will be `"/usr/local/CodeMagus/orkstrprd/orkstrprd.d"`.

Configuration example Contents of the configuration of a attachments to processes daemon on machine `cmlserver1`.

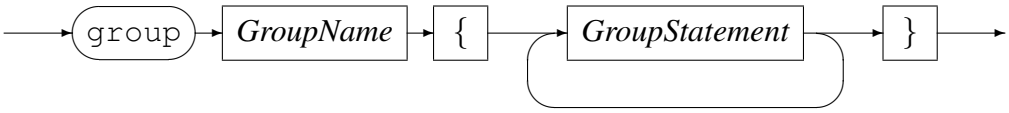
```
machine cmlserver1
   {
   description = "cmlserver1.codemagus.com (10.58.31.178)"
   variable CODEMAGUS_HOME external default "/usr/local/CodeMagus"
   variable CODEMAGUS_LOGS external default "/var/log/CodeMagus"
   variable CODEMAGUS_ORKHESTRA_ORKSTRPRD_PORT external default "62200"
  # Environment variables for Recio
   variable CODEMAGUS_AMDBINS="${CODEMAGUS_HOME}/bin/"
   variable CODEMAGUS_AMDCATPATH="${CODEMAGUS_HOME}/Catalogs"
   variable CODEMAGUS_AMDCATNAME="MASTCAT"
   variable CODEMAGUS_AMDLIBS="${CODEMAGUS_HOME}/lib/"
   variable CODEMAGUS_AMDPATH="${CODEMAGUS_HOME}/bin/%s.amd"
   variable CODEMAGUS_AMDSUFDL=".so"
   variable HOME external
   include directory = "${CODEMAGUS_HOME}/orkstrprd/orkstrprd.d"
   }
```

This is a typical example of a daemon configuration on the CML servers. The environment variables in this example are inherited by the subsequent configurations, both groups and processes as explained further below.

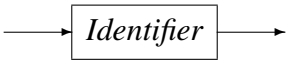
2.3.3 Group

The configuration file defines the Group, name and its environment. It all so specifies a directory that contains configuration files to define the NFT groups and their processes.

Group

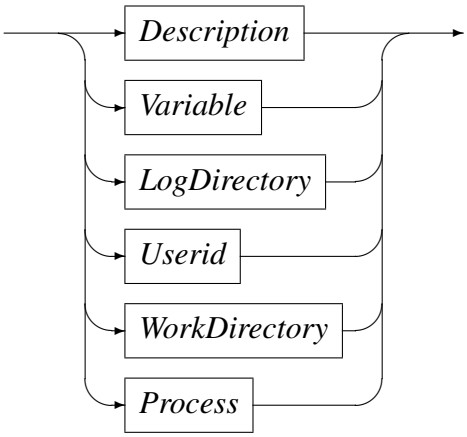


GroupName

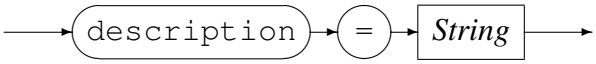


GroupName identifies the name of the group that this configuration is pertinent to.

GroupStatement



Description



The description provides a mechanism for further documenting the machine group and its purpose.

Variable see section 2.3.2 on page 11.

LogDirectory

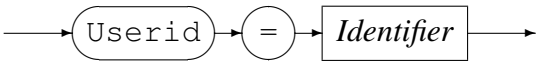
This specifies the log directory for processes running in this group.
The logs per process are:

- *stderr* directed to log file
  `<process name>_Dccyymmdd_Thhmmss_pid<pid>.stderr`
- *stdout* directed to log file
  `<process name>_Dccyymmdd_Thhmmss_pid<pid>.stdout`

Example log file names:

```
/home/app/cmlstress/logs/MDM_V3NFT/
xmlcp_D20181106_T131341_pid31395.stderr
xmlcp_D20181106_T131341_pid31395.stdout
xmlcp_D20181106_T131341_pid31396.stderr
xmlcp_D20181106_T131341_pid31396.stdout

/home/app/cmlstress/logs/MDM_V3NFT/proxy/
xmlcp_http_proxy_D20181106_T131342_pid31674.stderr
xmlcp_http_proxy_D20181106_T131342_pid31674.stdout
```

Userid ;



Sets the user identification to this when running a process in this group.

WorkDirectory

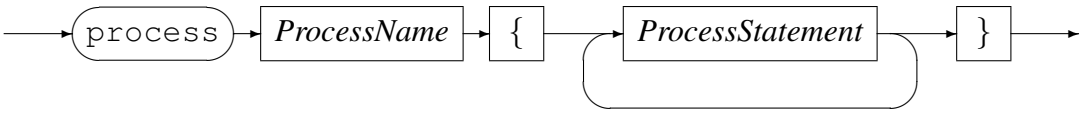
Sets the working directory to this when running a process in this group.

Configuration examples Extract from `mdm_v3.conf` showing group configuration:

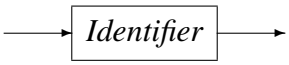
```
group mdm_v3nft
   {
   userid = cmlstress
   description = "MDM V3"

   var BASEDIR="${HOME}"
   var NFT_ID="MDM_V3NFT"
   var NFT="${BASEDIR}/CodeMagus/${NFT_ID}/testdata"
   var CMD_PORT="61128"
   var PREFIX="web_"
   var SOURCES="cmlqa1"

   var BINDIR="${CODEMAGUS_HOME}/bin"
   var CACHE="${NFT}/cache"
   var CODEMAGUS_FORMATS="${HOME}"
   var CPPATH="${BINDIR}"
   var DATA="${NFT}/data"
   var LIBDIR="${CODEMAGUS_HOME}/lib"
   var LOGSPATH="${BASEDIR}/logs/${NFT_ID}"
   var ORKHESTRA="${NFT}/orkhestra"
   var MACHINE="${ORKHESTRA}"
   var MCH_INC="${ORKHESTRA}/include"
   var MCH_NAME="${PREFIX}mdm_v3"
   var MSG="${MACHINE}/${MCH_NAME}.msg"
   var METRICS="${ORKHESTRA}"
   var SCRIPTS="${NFT}/scripts"
   var STATSPATH="${BASEDIR}/stats/${NFT_ID}"
   var system=qa
   var SYSTEM=QA
   var XMLDOCS="${NFT}/xml"

   var OS_DELIM="/"
   var COMMON_FORMATS="${CODEMAGUS_FORMATS}/CodeMagus/CommonData/testdata"

   log  directory = "${LOGSPATH}"
   work directory = "${SCRIPTS}"
   process orkagentr
   ...
```

2.3.4 Process

Process

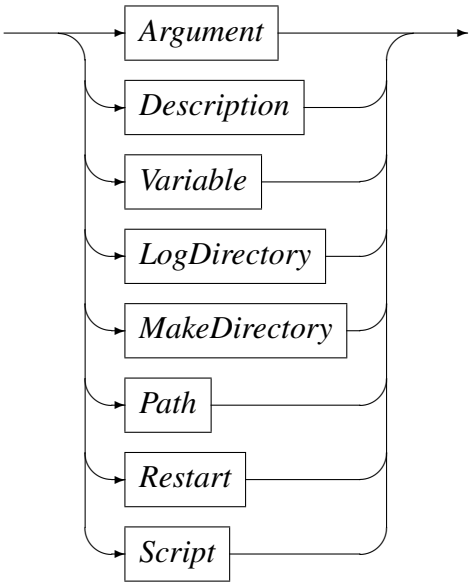


ProcessName

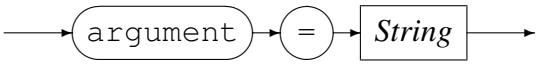


ProcessName identifies the name of the process that this configuration is pertinent to.

ProcessStatement



Argument

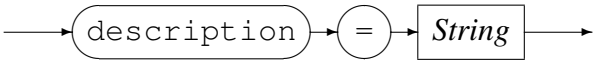


Command line argument(s) to pass to the process at start up. Please note environment variables will be resolved at startup of the process.

Examples:

```
argument = "--pass-copy-num"
argument = "--command='${SCRIPTS}/${system}_web_mdm_v3.cmd'"
argument = "--stats-interval=30"
argument = "--stats-metrics=text(${STATSPATH}/mdm_stats_D${DATE_CCYYMMDD}"
           ".metric,mode=a)"
argument = "--background"
```

Description

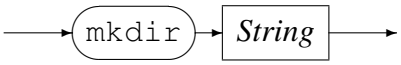


The description provides a mechanism for further documenting the process and its purpose.

Variable see section 2.3.2 on page 11.

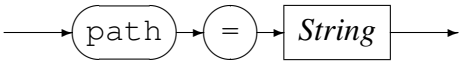
LogDirectory If specified, will override the log directory for this process. see section 2.3.3 on page 14.

MakeDirectory



Create the directory, if it does not already exist. Any parent directories are created as needed.

Path



This specifies the path to the executable.

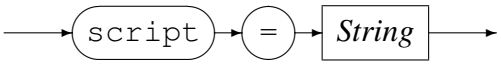
Example:

```
variable CODEMAGUS_HOME external default "/usr/local/CodeMagus"
...
path = "${CODEMAGUS_HOME}/bin/orkagentr_stpr"
```

Restart

If the process terminates or abends it will be restarted. If a process is restarted more than twice in minute, the process will be deemed to have an unrecoverable error and will not be restarted again.

Script



Example:

```
variable NFT_ID="MDM_V3NFT"
variable NFT="${BASEDIR}/CodeMagus/${NFT_ID}/testdata"
variable SCRIPTS="${NFT}/scripts
...
script = "${SCRIPTS}/qa_web_mdm_v3_vars.sh"
```

The script is parsed by the daemon to extract the environment variables and their values in the script. This is a very rudimentary parse and only caters for very simple shell scripting. These extracted variables are applied to the process environment as part of a process start up sequence.

`qa_web_mdm_v3_vars.sh`:

```
. ./setpaths.sh qa web

### Sources and objtypes
export CODEMAGUS_FORMATS=${HOME}
export COMMON_FORMATS=${HOME}/CodeMagus/CommonData/testdata

export MSG=${MACHINE}/${MCH_NAME}.msg
export WSSE=""

Party_ENDP="https://ssg-q.it.nednet.co.za:443"
export URI_Party_P="/services/ent/profilemanagement/Party/v3"
export URI_Party="${Party_ENDP}${URI_Party_P}"

PartyP_ENDP="https://ssg-q.it.nednet.co.za:443"
export URI_PartyP_P="/services/ent/profilemanagement/partyprimarycontactdetail/v1"
export URI_PartyP="${PartyP_ENDP}${URI_PartyP_P}"
```

Note that the variables `MACHINE` and `MCH_NAME` will be expanded to the values defined for them in the daemon configuration.

Configuration examples Extract from `mdm_v3.conf` showing process configurations:

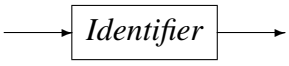
```
   process orkagentr
      {
      #var START_PROCESS_FORK_EXEC="non empty"
      log  directory = "${LOGSPATH}"
      path = "${CODEMAGUS_HOME}/bin/orkagentr_stpr"
      script = "${SCRIPTS}/qa_web_mdm_v3_vars.sh"
      description = "process orkagentr"
      mkdir "${LOGSPATH}/proxy"
      }
   process xmlcp
      {
      #var START_PROCESS_FORK_EXEC="non empty"
      var REALM="AFRICA.NEDCOR.NET"
      var SPN="HTTP/servicesecuritygateway-qa.africa.nedcor.net"
      var SERVICE="${SPN}@${REALM}"
      var  HEADERS="${SCRIPTS}/http_header.txt"
      log  directory = "${LOGSPATH}"
      path = "${CODEMAGUS_HOME}/bin/xmlcp_stpr"
      script = "${SCRIPTS}/qa_web_mdm_v3_vars.sh"
      description = "process xmlcp"
      }
   process xmlcp_http_proxy
      {
      log  directory = "${LOGSPATH}/proxy"
      path = "${CODEMAGUS_HOME}/bin/xmlcp_http_proxy_stpr"
      script = "${SCRIPTS}/qa_web_mdm_v3_vars.sh"
      description = "process xmlcp_http_proxy"
      restart
      }
---------- start orkhestra direct
   process orkhestra
      {
      path = "${CODEMAGUS_HOME}/bin/orkhestra"
      description = " NFT startup"
      argument = "--pass-copy-num"
      argument = "--command='${SCRIPTS}/${system}_web_mdm_v3.cmd'"
      argument = "--stats-interval=30"
      argument = "--stats-metrics=text(${STATSPATH}/${system}_web_mdm_v3_stats_D${DAT
      argument = "--background"
      }
---------- or start orkhestra via a script
   process orkhestra_sh
      {
      path = "${SCRIPTS}/qa_web_mdm_v3.sh"
      description = " NFT startup"
      }
----------
```

3 Orkhestra

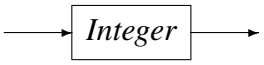
This section describes the *orkhestra* configuration changes required to utilise the attachments to processes daemon for starting the *orkhestra* remote agents and control programs. The following is required:

- The NFT must be configured to use remote agents, see orkhestra: Configuration and User Reference Version 1[1]).
- The NFT group must be configured for the attachments to processes daemons on the required hosts.
- Add the following to the local agent configuration:

GroupName



PortNumber



GroupName identifies the name of the group that is configured for this NFT.

Local agent configuration:

```
agent qa_web_mdm_v3_agent;
 process daemon group mdm_v3nft port 62200;
   open log "text(${LOG}/qa_web_mdm_v3_lagent_D${DATE_YYYYMMDD}"
         "_T${TIME_HHMMSS}.txt,mode=w)";
   remote agent cmlqa0
      host localhost;
      port 22221;
      control programs (xmlcp);
      open log "text(${LOG}/${system}_web_mdm_v3_ragent_D${DATE_YYYYMMDD}"
            "_T${TIME_HHMMSS}.txt,mode=w)";
   end;
  remote agent cmlqa1
     host localhost;
     port 22222;
     control programs (xmlcp);
     open log "text(${LOG}/${system}_web_mdm_v3_ragent_D${DATE_YYYYMMDD}"
           "_T${TIME_HHMMSS}.txt,mode=w)";
  end;
end.
```

For each process started by the daemon there are two logs - see section 2.3.3 on page 14. The command line arguments and the environment for the process started is shown at the start of the *stdout* log. Following is an extract of a *stdout* log for a control program started by *orkhestra* showing this:

xmlcp_D20181106_T133310_pid1794.stdout:

```
Tue Nov  6 13:33:10 2018: Process started
Arguments
   /usr/local/CodeMagus/bin/xmlcp_stpr
   --to-pipe=-1
   --from-pipe=-1
   --copy-num=16
   --message=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra/web_mdm_v3.ms
   --doc-path=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/xml
   --host-addr-file=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts/proxy_hc
   --nc-format=F1234
   --nc-bias=0
   --nc-offset=0
   --log-open-spec=text(/home/app/cmlstress/logs/MDM_V3NFT/qa_web_mdm_v3_D20181106_T1
   --proxy-insert-wsse
   --proxy-kerberos-logon
   --proxy-count=100
   --
   xmlcp_http_proxy
   --verbose
   --not-persistant
   --dll-name=/usr/local/CodeMagus/lib/ncphttp.so
   --headers=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts/http_header.txt
   --format=F1234
   --bias=0
   --kerberos-cache-path=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/cache
   --kerberos-logon-script=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts/k
   --kerberos-realm=AFRICA.NEDCOR.NET
   --kerberos-service-name=HTTP/servicesecuritygateway-qa.africa.nedcor.net@AFRICA.NE
Environment
   CODEMAGUS_HOME=/usr/local/CodeMagus
   CODEMAGUS_LOGS=/var/log/CodeMagus
   CODEMAGUS_ORKHESTRA_ORKSTRPD_PORT=62200
   CODEMAGUS_AMDBINS=/usr/local/CodeMagus/bin/
   CODEMAGUS_AMDCATPATH=/usr/local/CodeMagus/Catalogs
   CODEMAGUS_AMDCATNAME=MASTCAT
   CODEMAGUS_AMDLIBS=/usr/local/CodeMagus/lib/
   CODEMAGUS_AMDPATH=/usr/local/CodeMagus/bin/%s.amd
   CODEMAGUS_AMDSUFDL=.so
   HOME=/home/app/cmlstress
   USER=cmlstress
   BASEDIR=/home/app/cmlstress
   NFT_ID=MDM_V3NFT
   NFT=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata
   CMD_PORT=61125
   PREFIX=web_
   SOURCES=cmlqa1
```

```
   BINDIR=/usr/local/CodeMagus/bin
   CACHE=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/cache
   CODEMAGUS_FORMATS=/home/app/cmlstress
   CPPATH=/usr/local/CodeMagus/bin
   DATA=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/data
   LIBDIR=/usr/local/CodeMagus/lib
   LOGSPATH=/home/app/cmlstress/logs/MDM_V3NFT
   ORKHESTRA=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra
   MACHINE=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra
   MCH_INC=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra/include
   MCH_NAME=web_mdm_v3
   MSG=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra/web_mdm_v3.msg
   METRICS=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/orkhestra
   SCRIPTS=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts
   STATSPATH=/home/app/cmlstress/stats/MDM_V3NFT
   system=qa
   SYSTEM=QA
   XMLDOCS=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/xml
   OS_DELIM=/
   COMMON_FORMATS=/home/app/cmlstress/CodeMagus/CommonData/testdata
   PWD=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts
   REALM=AFRICA.NEDCOR.NET
   SPN=HTTP/servicesecuritygateway-qa.africa.nedcor.net
   SERVICE=HTTP/servicesecuritygateway-qa.africa.nedcor.net@AFRICA.NEDCOR.NET
   HEADERS=/home/app/cmlstress/CodeMagus/MDM_V3NFT/testdata/scripts/http_header.txt
   CODEMAGUS_ORKHESTRA_CONNECT_BACK_HOST=127.0.0.1
   CODEMAGUS_ORKHESTRA_CONNECT_BACK_PORT=54049
   CODEMAGUS_ORKHESTRA_PROCESS_GROUP=mdm_v3nft
   WSSE=
   Party_ENDP=https://ssg-q.it.nednet.co.za:443
   URI_Party_P=/services/ent/profilemanagement/Party/v3
   URI_Party=https://ssg-q.it.nednet.co.za:443/services/ent/profilemanagement/Party/v
   PartyP_ENDP=https://ssg-q.it.nednet.co.za:443
   URI_PartyP_P=/services/ent/profilemanagement/partyprimarycontactdetail/v1
   URI_PartyP=https://ssg-q.it.nednet.co.za:443/services/ent/profilemanagement/partyp
...
```

4 *xmlcp*

This section describes the *xmlcp* control program (**xmlcp**: Web Services NFT Guide and Reference Version 1[2]) configuration changes required to utilise the attachments to processes daemon for starting the *xmlcp* HTTP proxies. The following is required:

- *xmlcp* must have been started by the attachments to processes daemon.
- define the HTTP proxy name, its arguments and number of proxies required. This is defined as part of the control program definition, see the bold text in the following example.

Example control program definition:

```
define control_program xmlcp (
   copies (16)
   path ("${CPPATH}/xmlcp")
   parameter (
   "--message=${MSG} "
   "--doc-path=${XMLDOCS} "
   "--host-addr-file=${SCRIPTS}/proxy_hosts.txt "
   "--nc-format=F1234 --nc-bias=0 --nc-offset=0 "
   "--log-open-spec=text(mdm_v3_D${Y}_T${H}_pid%P.log.txt,mode=w) "
   "--proxy-insert-wsse "
   "--proxy-kerberos-logon "
   "--proxy-count=100 "
   "-- "
   "xmlcp_http_proxy "
      "--verbose "
      "--not-persistant "
      "--dll-name=${CODEMAGUS_HOME}/lib/ncphttp.so "
      "--headers=${HEADERS} "
      "--format=F1234 --bias=0 "
      "--kerberos-cache-path=${CACHE} "
      "--kerberos-logon-script=${SCRIPTS}/kerberos_logon.sh "
      "--kerberos-realm=${REALM} "
      "--kerberos-service-name=${SERVICE} "
  ))
```

- 100 proxies required
- The proxy name and its command line arguments follow the line containing only `"-- "`.

5 Processes Library

This section describes the *orkhestra* attachments to processes daemon library interface. The API interface is defined in the header `orkstrpr.h` (section 5.12 on page 35) and explained in the following sections.

5.1 `orkstrpr_set_fds()`

```
void orkstrpr_set_fds(orkstrpr_instance_t *orkstrpr,int *fdmax,fd_set *readfds,
      fd_set *writefds);
```

Description

This function sets the supplied `readfds` and `writefds` with the library's internal file descriptors that needs monitoring. The client should call this function before calling `select()`.

Parameters

- `*orkstrpr`
  This parameter contains the *orkstrpr* instance.
- `*fdmax`
  This parameter is set to the highest-numbered file descriptor in any of the two sets, plus 1.
- `*readfds`
  This parameter is the read set of file descriptors to be watched.
- `*writefds`
  This parameter is the write set of file descriptors to be watched.

Return Value

`void` function - no return value.

5.2 orkstrpr_check_fds()

```
void orkstrpr_check_fds(orkstrpr_instance_t *orkstrpr,int *fdcnt,
      fd_set *readfds, fd_set *writefds);
```

Description

This function checks the supplied `readfds` and `writefds` for its own internal file descriptors that need attention and processes any actions required for them. The client should call this function after a `select()` is done.

Parameters

- `*orkstrpr`
  This parameter contains the *orkstrpr* instance.
- `*fdcnt`
  This parameter is set to the number of file descriptors needing attention. This is the return value from the `select()` function.
- `*readfds`
  This parameter is the read set of file descriptors that need attention.
- `*writefds`
  This parameter is the write set of file descriptors that need attention.
- `crc32fcs`
  This parameter must be set to zero.

Return Value

`void` function - no return value.

5.3 orkstrpr_error()

```
char *orkstrpr_error(orkstrpr_instance_t *orkstrpr);
```

Description

This function returns a NULL terminated string describing the last error encountered by one of the other functions of the *orkstrpr* library. If the *orkstrpr* instance is supplied as NULL, then the last error returned unrelated to a *orkstrpr* instance is returned (for example, in the situation that an `orkstrpr_open()` failed, then the error message is placed in a global structure. Otherwise for *orkstrpr* instance related errors, the error message is taken from the *orkstrpr* instance structure.

Parameters

- `*orkstrpr`
  This parameter contains the *orkstrpr* instance.

Return Value

Returns a NULL terminated string describing the last error message encountered by any call to the library.

5.4 orkstrpr_open()

Synopsis

```
orkstrpr_instance_t *orkstrpr_open(orkstrpr_notification_t notification,
      orkstrpr_verblog_t func_verblog);
```

Description

This function creates and initialise the orkstrpr instance.

Parameters

- `notification()`
  notification - see section 5.9 on page 32.
- `func_verblog()`
  This parameter is the optional user callback function for verbose processing - see section 5.10 on page 33.

Return Value

This function is expected to succeed and returns the instance created.

5.5 orkstrpr_connect2rdaemon()

Synopsis

```
orkstrpr_rdaemon_t *orkstrpr_connect2rdaemon(orkstrpr_instance_t *orkstrpr,
      void *user_data,char *host,int port,char *group);
```

Description

This function connects to the requested remote processes daemon.

Parameters

- `*orkstrpr`
  This parameter contains the *orkstrpr* instance created by the call to `orkstrpr_open()` - see section 5.4 on page 28.
- `*user_data`
  This parameter is the user data associated with this connection.
- `*host`
  This parameter is the host name/IPADDRESS of the remote process daemon.
- `port`
  This parameter is the port number of the remote process daemon.
- `*group`
  This parameter is the configuration group requested.

Return Value

On success the connection instance structure is returned. On error, NULL is returned and function `orkstrpr_error()` will return a description of the error encountered.

Note

If the connection to the daemon is subsequently lost, this will be notified via the callback registered on the `orkstrpr_open()` call. It is the users responsibility to free all resources acquired by calling the function `orkstrpr_rdaemon_free()` - see section 5.7 on page 30 - see section 5.9 on page 32.

5.6 orkstrpr_rdaemon_close()

Synopsis

```
orkstrpr_rdaemon_t * orkstrpr_rdaemon_close(orkstrpr_rdaemon_t *rdaemon);
```

Description

This function closes the connection and frees all resources acquired on behalf of the remote daemon connection.

Parameters

- `*rdaemon`
  This parameter contains the remote daemon connection instance - see section 5.5 on page 29.

Return Value

Return NULL.

5.7 orkstrpr_rdaemon_free()

Synopsis

```
void orkstrpr_rdaemon_free(orkstrpr_rdaemon_t *rdaemon);
```

Description

This function frees all resources acquired on behalf of the remote daemon connection.

Parameters

- `*rdaemon`
  This parameter contains the remote daemon connection instance - see section 5.5 on page 29.

Return Value

`void` function - no return value.

5.8 orkstrpr_start_process()

Synopsis

```
int orkstrpr_start_process(orkstrpr_rdaemon_t *rdaemon,void *user_data,
      char *process_name,uint16_t port,char **argv);
```

Description

This function starts the requested remote process.

Parameters

- `*rdaemon`
  This parameter contains the remote daemon connection instance - see section 5.5 on page 29.
- `*user_data`
  This parameter is the user data associated with this process.
- `*process_name`
  This parameter is the process name to start.
- `**argv`
  This optional parameter is the arguments to pass to the process. `**argv` is a pointer array, with the last element a NULL pointer.

Return Value

Return zero on success, On error, -1 is returned and function `orkstrpr_error()` will return a description of the error encountered.

Note

If the process terminates, this will be notified via the callback registered on the `orkstrpr_open()` call - see section 5.9 on page 32.

5.9 orkstrpr_notification_t

Synopsis

```
typedef enum
   {
   ORKSTRPR_CLOSED,              /* processes daemon terminated/resetted */
   ORKSTRPR_PROCESS_TERMINATED   /* process terminated */
   } orkstrpr_error_t;

typedef void (*orkstrpr_notification_t)(orkstrpr_rdaemon_t *rdaemon,
      void *rdaemon_user_data, orkstrpr_process_t *process,
      void *process_user_data, orkstrpr_error_t error,char *msg);
```

Description

`orkstrpr_notification_t` is the prototype for the user callback function for when either a process terminated or the circuit to the processes daemon is closed (meaning the daemon terminated/resetted),

Parameters

- `*rdaemon`
  This parameter contains the remote daemon connection instance - see section 5.5 on page 29.
- `*rdaemon_user_data`
  This parameter contains the user data supplied on the `orkstrpr_connect2rdaemon()` call - see section 5.5 on page 29.
- `*process`
  This parameter contains the process instance (see section 5.8 on page 31) if `(*error == ORKSTRPR_PROCESS_TERMINATED)`.
- `*process_user_data`
  This parameter contains the user data supplied on the `orkstrpr_start_process()` call - see section 5.8 on page 31.
- `error`
  - `(error == ORKSTRPR_CLOSED)`
    Connection lost with the remote attachments to processes daemon, the daemon either terminated or resetted.
  - `(error == ORKSTRPR_PROCESS_TERMINATED)`
    A process terminated as described by the parameter `*process`'
- `*msg`
  This parameter contains a NULL terminated string describing the error.

Return Value

`void` function - no return value.

5.10 orkstrpr_verblog_t

Synopsis

```
typedef void (*orkstrpr_verblog_t)(int verbose_level,int length,char *buffer,
      char *format, ...);
```

Description

`orkstrpr_verblog_t` is the prototype for the optional user provided callback function for verbose printing of library interface.

Parameters

- `verbose_level`
  This parameter describes the verbose level pertaining to this call, values are zero,1 or 2.
- `*length`
  This parameter describes the length of next parameter, if supplied.
- `*buffer`
  This parameter is optional binary data to be dumped.
- `format`
  This parameter is a format string in the '`printf`' style.

Return Value `void` function - no return value.

5.11 orkstrpr_query

Synopsis

```
typedef void (*orkstrpr_query_callback_t)(void *userdata,char *buffer);

int orkstrpr_query(orkstrpr_rdaemon_t *rdaemon,void *user_data,char query,
      orkstrpr_query_callback_t callback);
```

Description

This function sends the requested query to the process daemon. If the query is success full zero is returned. This would be after all the query responses are received. Each response results in a call to the provided callback function as per the prototype `orkstrpr_query_callback_t`.

`orkstrpr_query()` Parameters

- `*rdaemon`
  This parameter contains the remote daemon connection instance - see section 5.5 on page 29.
- `*user_data`
  This parameter is the user data to be passed to callback print function.
- `*query`
  This parameter is the query to perform:
  - `query == "c"`
    Display configuration parsed
  - `query == "s"`
    Display the status of current processes running

callback Parameters

- `*user_data`
  This parameter is the user data from the query function.
- `*buffer`
  This parameter is a buffer to print.

Return Value

Return zero on success, On error, -1 is returned and function `orkstrpr_error()` will return a description of the error encountered.

5.12 orkstrpr.h

```
#ifndef ORKSTRPR_H
#define ORKSTRPR_H
  /* File orkstrpr.h
   *
   * This header file provides the interface to the orkhestra attachment
   * to processes daemon. It describes the data structures and functions
   * exposed by this library.
   *
   * Copyright (c) 2018. Code Magus Limited. All rights reserved.
   *
   */

  /* $Author: janvlok $
   * $Date: 2020/02/17 11:34:32 $
   * $Id: orkstrpr.h,v 1.5 2020/02/17 11:34:32 janvlok Exp $
   * $Name:  $
   * $Revision: 1.5 $
   * $State: Exp $
   *
   * $Log: orkstrpr.h,v $
   * Revision 1.5  2020/02/17 11:34:32  janvlok
   * Added prototype orkstrpr_start_process_pers
   *
   * Revision 1.4  2019/07/19 13:24:02  janvlok
   * Make delim generic
   *
   * Revision 1.3  2019/03/11 09:33:21  janvlok
   * Fixed PORT env
   *
   * Revision 1.2  2019/01/16 10:38:01  janvlok
   * Removed crc32fcs from seet/check fds
   *
   * Revision 1.1  2018/10/01 18:15:33  janvlok
   * Take on
   *
   */

#include <network.h>
#include <verblog.h>

typedef struct orkstrpr_instance orkstrpr_instance_t;
typedef struct orkstrpr_process orkstrpr_process_t;
typedef struct orkstrpr_rdaemon orkstrpr_rdaemon_t;

  /*
   * Defines, constants and enums:
   */

#define ORKSTRP_MSG_SZ NETWORK_DFLT_MSG_SZ*2   /* network message length */
#define ORKSTRP_BUF_SZ NETWORK_DFLT_BUF_SZ*2   /* network buffer length */

#define ORKSTRP_CBH         "CODEMAGUS_ORKHESTRA_CONNECT_BACK_HOST"
#define ORKSTRP_CBP         "CODEMAGUS_ORKHESTRA_CONNECT_BACK_PORT"
#define ORKSTRP_PROC_GROUP  "CODEMAGUS_ORKHESTRA_PROCESS_GROUP"
#define ORKSTRP_DAEMON_PORT "CODEMAGUS_ORKHESTRA_ORKSTRPD_PORT"
#define ORKSTRP_DELIM       0x01

typedef enum
   {
   ORKSTRPR_CLOSED,             /* processes daemon terminated/resetted */
   ORKSTRPR_PROCESS_TERMINATED  /* proceess terminated */
   } orkstrpr_error_t;
```

```
  /*
   * Call back function definitions.
   */

  /* orkstrpr_notification_t is the prototype for the user callback function
   * for when either a proceess terminated or the circuit to the processes
   * daemon is closed (meaning the daemon terminated/resetted),.
   * *rdaemon is the processes daemon instance.
   * *rdaemon_user_data is user_data as supplied on instance open call.
   * *process is the process if the error is ORKSTRPR_PROCESS_TERMINATED and
   * *process_user_data is user_data as supplied on start proceess call.
   * error is the error thah occured, with *msg describing it.
   */

typedef void (*orkstrpr_notification_t)(orkstrpr_rdaemon_t *rdaemon,
      void *rdaemon_user_data, orkstrpr_process_t *process, void *process_user_data,
      orkstrpr_error_t error,char *msg);

  /* orkstrpr_verblog_t is the prototype for the user provided function for
   * verbose printing of orkstrpr library.
   */

typedef void (*orkstrpr_verblog_t)(int verbose_level,int len,char *buf,
      char *format, ...);

  /*
   * Data structures.
   */

  /*
   * struct orkstrpr_mhdr describes the communcations interface header between
   * the start process daemon and clients.
   * This header preceeds the payload.
   *
   * ORKSTRPR_CONN
   *   Req -> <hdr> <str group>
   *   Res -> <hdr> <string error/message>
   * ORKSTRPR_QUERY
   *   Req -> <hdr>  -- subtype = actual query
   *   Res -> <hdr> <string payload>
   * ORKSTRPR_SPROC
   *   Req -> <hdr> <sproc> <process name> <options>
   *   Res -> <hdr> <string message>
   * ORKSTRPR_SPROC_EXIT
   *   From Server -> <hdr> <string message>
   *
   */

typedef struct orkstrpr_mhdr orkstrpr_mhdr_t;
typedef struct orkstrpr_sproc orkstrpr_sproc_t;

#define  ORKSTRPR_CONN        0     /* connection request/response */
#define  ORKSTRPR_QUERY      'q'    /* inquery */
#define  ORKSTRPR_SPROC      's'    /* start process */
#define  ORKSTRPR_SPROC_PERS 'S'    /* start process */
#define  ORKSTRPR_SPROC_EXIT 'x'    /* remote process terminated */
struct orkstrpr_mhdr
   {
   unsigned char type;         /* message type */
   unsigned char subtype;      /* message sub type */
   unsigned char cont;         /* > 0 - continue */
   unsigned char is_error;     /* error is in payload */
   unsigned char echo[8] ;     /* echo/handle */
   };
struct orkstrpr_sproc
   {
```

```
   uint16_t port;              /* connect back port */
   uint32_t argv_offset;       /* offset in message to options string */
   };

  /*
   * Exposed functions.
   */

  /* Function orkstrpr_error() returns a message relating to the last error
   * returned by one of the other function of the orkstrpr library. If the
   * orkstrpr instance is supplied as NULL, then the last error returned
   * unrelated to a orkstrpr instance is returned (for example, in the
   * situation that an orkstrpr_open() failed, then the error message is
   * placed in a global structure. Otherwise for orkstrpr instance related
   * errors then error message is taken from the orkstrpr instance structure.
   */

char *orkstrpr_error(orkstrpr_instance_t *inst);

  /* orkstrpr_set_fds() set readfds and writefds for the internal fds
   * that needs attention.
   * orkstrpr_check_fds() check readfds and writefds after the
   * select().
   */

void orkstrpr_set_fds(orkstrpr_instance_t *nc,int *fdmax,fd_set *readfds,
      fd_set *writefds);
void orkstrpr_check_fds(orkstrpr_instance_t *nc,int *fdcnt,fd_set *readfds,
      fd_set *writefds);

  /* Function orkstrpr_open() Creates, initialise the orkstrpr instance.
   *
   * notification() is the user callback function for proceess/daemon
   * termination.
   * func_verblog() is the optional user callback function for verbose
   * processing.
   *
   * On success this function returns the instance created.
   * On error NULL is returned and function orkstrpr_error() will return
   * a description of the error encountered.
   * This function is expected to succeed.
   */

orkstrpr_instance_t *orkstrpr_open(orkstrpr_notification_t notification,
      orkstrpr_verblog_t func_verblog);

  /* Function orkstrpr_connect2rdaemon() connects to the requested remote processes
   * daemon.
   *
   * *orkstrpr is the orkstrpr instance
   * *user_data is user data accociated with this instance.
   * *host and port is the address of the proceesses daemon.
   * *group is the process group.
   *
   * On success this function returns the remote daemon connection instance.
   * On error, NULL is returned and function orkstrpr_error() will return
   * a description of the error encountered.
   *
   * If the connection to the daemon is lost, this will be notified via
   * the call back registered on the orkstrpr_open() call.
   * It is the users responsibility to free all resources acquired by calling
   * the function orkstrpr_rdaemon_free()
   */

orkstrpr_rdaemon_t *orkstrpr_connect2rdaemon(orkstrpr_instance_t *orkstrpr,
      void *user_data,char *host,int port,char *group);
```

```
  /* Function orkstrpr_rdaemon_free() will free all resources acquired
   * on behalf of the remote daemon connection.
   */

void orkstrpr_rdaemon_free(orkstrpr_rdaemon_t *rdae);

  /* Function orkstrpr_rdaemon_close() will free all resources acquired
   * on behalf of the remote daemon connection.
   * The connection to the daemon will be closed.
   * Return NULL.
   */

orkstrpr_rdaemon_t * orkstrpr_rdaemon_close(orkstrpr_rdaemon_t *rdaemon);

  /* orkstrpr_query_callback() is the prototype describing the callback
   * function for the responses triggerred by orkstrpr_query() call.
   */

typedef void (*orkstrpr_query_callback_t)(void *userdata,char *buffer);

  /* Function orkstrpr_query() send the requested query to the process daemon.
   * If the query is successfull zero is returned, after all the query response .
   * are received. Each response result in a call to the provide callback
   * function.
   *
   * *rdae is the remote daemon connection
   * *user_data is user data accociated with this call, and is passed on the
   * callback function.
   *
   * On error -1 is returned and function orkstrpr_error() will return
   * a description of the error encountered.
   */

int orkstrpr_query(orkstrpr_rdaemon_t *rdae,void *user_data,char query,
      orkstrpr_query_callback_t callback);

  /* Function orkstrpr_start_process() starts the requested remote processes.
   *
   * *rdaemon is the remote daemon to start the process on.
   * *user_data is user data accociated with this process.
   * *process_name the process to start.
   * port is the connect back port.
   * Optional **argv is the arguments to pass - pointer array, with the last
   * element a NULL pointer.
   *
   * Return zero on succsess, On error, -1 is returned and function
   * orkstrpr_error() will return a description of the error encountered.
   *
   * If the process terminates, this will be notified via the call back
   * registered on the orkstrpr_open() call.
   * Function orkstrpr_start_process_pers() starts the requested remote processes
   * and will orphane the process when the requestor closes the connection.
   */

int orkstrpr_start_process(orkstrpr_rdaemon_t *rdaemon,void *user_data,
      char *process_name,uint16_t port,char **argv);
int orkstrpr_start_process_pers(orkstrpr_rdaemon_t *rdae,void *user_data,
      char *process_name,uint16_t port,char **argv);

#endif /* ORKSTRPR_H */
```

6 Processes Query program

orkstrprq is a program to query configuration and status information of an *orkhestra* attachments to processes daemon instance.

6.1 Synopsis

If *orkstrprq* is started with the parameter '`--help`' it will only display the command line parameters it accepts and a short description of each. The list of parameters along with a complete description of each can be found in the following section.

```
Usage: orkstrprq [OPTION...]
  -p, --port=<port>                       Remote daemon port
  -h, --host={<host name>|<IPADDR>}       Remote daemon host
  -q, --query={c|s}                       Remote daemon query command
  -v, --verbose                           Verbose processing

Help options:
  -?, --help                              Show this help message
      --usage                             Display brief usage message
```

6.2 Parameters

- '`-p|--port`' Specifies the port number of the *orkstrprd* instance to query.
- '`-h|--host`' Specifies the host name/IPADDRESS of the *orkstrprd* instance to query.
- '`-q|--query`' Specifies the query to do:
  - = '`c`' display configuration parsed.
  - = '`s`' display the status of current processes running.
- '`-v|--verbose`' Directs *orkstrprd* to output diagnostic information during execution.

A *orkhestra* Implementation

This section describes *orkhestra*'s interface to the attachments to processes daemon, using the attachments to processes library *orkstrpr*. The configuration requirements are outlined in section 3 on page 21. The source for *orkhestra* is in CVS module `orkhestra` on `software.codemagus.com/home/cvs/cvsroot`.

Open *orkstrpr* library instance - `orkagentl.c function init():`

```
   if (glb->rdaemon_port)
      glb->orkstrpr = orkstrpr_open(rdae_notification,verblog);
```

Callback function `verblog() - defined in verblog.h(libverblog.a).`

Callback function `rdae_notification() - orkagentl.c:`

```
  /* Function rdae_notification() is the user callback function
   * for when either a process terminated or the circuit to the processes
   * daemon is closed (meaning the daemon terminated/resetted).
   * *rdae is the processes daemon instance.
   * *rdaemon_user_data is user_data as supplied on instance open call.
   * *proc is the process if the error is ORKSTRPR_PROCESS_TERMINATED and
   * *proc_user_data is the user_data as supplied on start process call.
   * error is the error that occurred, with *msg describing it.
   */

void rdae_notification(orkstrpr_rdaemon_t *rdae,
      void *rdaemon_user_data, orkstrpr_process_t *proc, void *proc_user_data,
      orkstrpr_error_t error,char *msg)
   {
   rem_agent_t *rema;

  /* Note both user_data arguments are (rem_agent_t *).
   */
   rema = rdaemon_user_data;

   if (error == ORKSTRPR_CLOSED)
      err2pt(ORKAGENT_FAILED,-1,"Connection to remote start process daemon "
            "on %s failed: %s",rema->name,msg);
   else
      {
      err2pt(ORKAGENT_FAILED,-1,"Remote process %s %s : %s ",rema->name,
            RAGENT_NAME,msg);
      }
   return;
   } /* rdae_notification */
```

Start the remote agents - `orkagentl.c function init():`

```
...
for (i = 0; i < glb->nragents; i++)
   {
   rema = glb->ragent[i];
   if (glb->rdaemon_port)
      {
     /* Open the callback circuit requesting an ephemeral port:
      */
      rema->conn = orkagent_open_circuit(glb->orkagent,rema,
            "0.0.0.0",0,2,remote_status_change,remote_data_received);
      if (!rema->conn)
         return err2pt(ORKAGENT_FAILED,-1,"CallBack connection""failed: %s",
            orkagent_error(glb->orkagent));
      cb_port = orkagent_source_port(rema->conn);

     /* Open a connection to the remote process daemon:
      */
      rema->rdae = orkstrpr_connect2rdaemon(glb->orkstrpr,rema,rema->host,
            glb->rdaemon_port,glb->rdaemon_group);
      if (!rema->rdae)
         return err2pt(ORKAGENT_FAILED,-1,"Unable to connect to remote "
               "daemon on %s: %s",rema->name,orkstrpr_error(glb->orkstrpr));

     /* Start the orkhestra remote agent:
      */
       if (orkstrpr_start_process(rema->rdae,rema,RAGENT_NAME,cb_port,NULL)
             < 0)
         return err2pt(ORKAGENT_FAILED,-1,"Unable to start process %s on %s:"
               " %s",RAGENT_NAME,rema->name,orkstrpr_error(glb->orkstrpr));
      }
...
```

Start a control program - `startcp.c` function `startcp()`:

```
...
  /* Arguments pointer array:
   *   memdebug parameters  (if memdebug was requested -  Only for exec)
   *   Path of executable   (Only for exec - Not for remote start process)
   *   --to-pipe=<fd>       (-1 for remote start process)
   *   --from-pipe=<fd>     (-1 for remote start process)
   *   --copy-num=<integer> (not for start of local agent
   *   c->parm              (Note environment variables are NOT
   *                        expanded for remote start process)
...
   if (sproc_group)
      {
     /* Start is via remote process daemon:
      */
      if (!c->rdae)
         {
        /* Open a connection to the remote process daemon:
         */
         c->rdae = orkstrpr_connect2rdaemon(orkstrpr,c,"localhost",
               sproc_port,sproc_group);
         if (!c->rdae)
            {
            printf("*** startcp() Error Unable to connect to process daemon on "
                  "localhost:%d: %s", sproc_port,orkstrpr_error(orkstrpr));
            return -1;
            }
         }
      strcpy(buf,c->parm);
      if (!c->path_resolved)
         c->path_resolved = strdup(basename(c->path));
      sprintf(to_pipe,"--to-pipe=%d",-1);
      sprintf(fr_pipe, "--from-pipe=%d",-1);
      }
...
   if (sproc_group)
      {
     /* Start is via remote process daemon:
      * Open the callback circuit requesting an ephemeral port:
      */
      cpa->lp = cplink_open_conn_tcp(cplink,c->name,0,
            disconnected, msg_recv,logit,cpa);
      if (!cpa->lp)
         {
         printf("*** startcp() Error Unable to open cplink passive "
               "connection");
         return -1;
         }

     /* Start the control program:
```

```
      */
      printf("*** startcp() Starting %s copy %d Call back port %d",
            c->name,cpa->copy,cpa->lp->lport);
      if (orkstrpr_start_process(c->rdae,cpa,c->name,cpa->lp->lport,c->argv)
            < 0)
         {
         printf("*** startcp() Error Unable to start process %s:"
               " %s",c->name,orkstrpr_error(orkstrpr));
         return -1;
         }
      }
...
```

Set and Check file descriptors - `orkagentl.c`
`function main():`

```
...
  /* Ask the agent and remote daemon interfaces to set there
   * file descriptors in readfds and writefds that needs watching.
   */
   orkagent_set_fds(glb->orkagent,&fdmax,&readfds,&writefds);
   if (glb->orkstrpr)
      orkstrpr_set_fds(glb->orkstrpr,&fdmax,&readfds,&writefds);

   tv.tv_sec = 1;
   tv.tv_usec = 0;
   fdcnt = select(fdmax+1,&readfds,&writefds,NULL,&tv);

  /* Get the interfaces to act on any file descriptors that needs action.
   */
   if (fdcnt > 0)
      orkagent_check_fds(glb->orkagent,&fdcnt,&readfds,&writefds);
   if (glb->orkstrpr && fdcnt > 0)
      orkstrpr_check_fds(glb->orkstrpr,&fdcnt,&readfds,&writefds);
...
```

B *xmlcp* Implementation

This section describes *xmlcp*'s interface to the attachments to processes daemon, using the attachments to processes library *orkstrpr*. This is for starting the HTTP proxies. The configuration requirements are outlined in section 4 on page 24. The source for *xmlcp* is in CVS module `xmlcp` on `software.codemagus.com/home/cvs/cvsroot`.

Set and Check file descriptors - `xmlcp.c function main()`:

```
while (1)
   {
   FD_ZERO(&readfds);
   FD_ZERO(&writefds);
   fdmax = 0;

  /* Ask orkhestra API and the network and remote daemon interfaces to set
   * there file descriptors in readfds and writefds that needs watching.
   */
   orkcpapi_set_fds(glb->orkcpapi,&fdmax,&readfds,&writefds);
   if (glb->nc)
      network_set_fds(glb->nc,&fdmax,&readfds,&writefds);
   if (glb->orkstrpr)
      orkstrpr_set_fds(glb->orkstrpr,&fdmax,&readfds,&writefds);

   tv.tv_sec = 1;
   tv.tv_usec = 0;
   fdcnt = select(fdmax+1,&readfds,&writefds,NULL,&tv);

  /* Get the interfaces to act on any file descriptors that needs action.
   */
   if (fdcnt > 0 && glb->nc)
      network_check_fds(glb->nc,&fdcnt,&readfds,&writefds,0);
   if (glb->orkstrpr && fdcnt > 0)
      orkstrpr_check_fds(glb->orkstrpr,&fdcnt, &readfds, &writefds);
   if (fdcnt > 0)
      orkcpapi_check_fds(glb->orkcpapi,&fdcnt,&readfds,&writefds);
   }
```

C Codemagus Typical NFT configuration

All process configurations are in CVS module `CodeMagus/ORKSTRP`:

```
CodeMagus/ORKSTRP/testdata/machines
      deploy.sh
      orkstrprd.cmlkeep.skel
      README
     /cmlqa0
         orkstrprd.conf
        /orkstrprd.d
     /cmlqa1
         orkstrprd.conf
        /orkstrprd.d
     /codemagus
         orkstrprd.conf
        /orkstrprd.d
     /common
        /orkstrprd.d
            enotes.conf
            glassfish.conf
            mdm_v3.conf
            mdm_v4.conf
            mdm_v5.conf
            soa_security.conf
     /cmlserver1
         orkstrprd.conf
        /orkstrprd.d
     /cmlserver2
          orkstrprd.conf
         /orkstrprd.d
```

Put a new nft in common/orkstrprd.d, use deploy.sh ¡machine name¿ to deploy, then restart the daemon on the deployed machine - check the daemon logs to see if any errors.

Deploy

```
$ ./deploy.sh cmlqa0
scp 5976.cmlkeep root@cmlqa0:/usr/local/CodeMagus/bin/orkstrprd.cmlkeep
 5976.cmlkeep                                  100% 1012     4.9KB/s   00:00
scp cmlqa0/orkstrprd.conf root@cmlqa0:/usr/local/CodeMagus/orkstrprd/
 orkstrprd.conf                                100%  869     3.9KB/s   00:00
scp common/orkstrprd.d/enotes.conf common/orkstrprd.d/glassfish.conf ...
    root@cmlqa0:/usr/local/CodeMagus/orkstrprd/orkstrprd.d/
 enotes.conf                                   100% 1784     8.7KB/s   00:00
 glassfish.conf                                100% 2635    12.7KB/s   00:00
 mdm_v3.conf                                   100% 2595    12.6KB/s   00:00
...
```

Restart the daemon

```
orkstrprd]$ ps -ef|grep orkstr
cmlstre+  3878     1  0 Jul11 ?        00:00:08 cmlkeep --verbose ...
```

```
cmlstre+ 19460  3878  0 12:19 ?        00:00:00 orkstrprd -v ...
[orkstrprd]$ kill 19460
```

Check the latest daemon log file in directory `logs/orkstrprd` ensuring no syntax errors:

```
[orkstrprd]$ cmlcat orkstrprd_D20190806_T121932_00003878.rdwlog|tail -14
Tue Aug  6 12:21:26 2019: conf() Conf file "/usr/.../orkstrprd/orkstrprd.conf"
Tue Aug  6 12:21:26 2019: conf() Conf Dir "/usr/.../orkstrprd/orkstrprd.d"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/glassfish.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/mdm_v3.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/mdm_v4.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/soa_security.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/enotes.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/mdm_v5.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/trmapp.conf"
*** Parsing "/usr/local/CodeMagus/orkstrprd/orkstrprd.d/trmapp88.conf"
Client 000001 0.0.0.0:62200 : Waiting(listen) for connections
UDP FB recv 000002 0.0.0.0:62200 : Opened (UDP recv)
UDP FB recv 000003 0.0.0.0:62200 : Opened (UDP send)
```

D Bibliography

References

[1] orkhestra: Configuration and User Reference Version 1. CML Document CML00041-01, Code Magus Limited, June 2011. PDF.

[2] **xmlcp**: Web Services NFT Guide and Reference Version 1. CML Document CML00116-00, Code Magus Limited, June 2017. PDF.