INNOVATIVE ECONOMY
NATIONAL COHESION STRATEGY

UNICORE MONITORING INFRASTRUCTURE PROBES ADMINISTRATOR AND DEVELOPER GUIDE

Mariusz Strzelecki

Document Version:	1.0
Component Version:	2.0.10
Date:	11 10 2011

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.
This work was co-funded by the PL-Grid project under the EU Grant Agreement Nr. POIG.02.03.00-00-007/08-00.

Contents

This document describes part of the Nagios-based UNICORE Monitoring Infrastructure developed in PL-Grid and EMI projects.

1 General Documentation

UNICORE Monitoring Infrastructure Probes (shortly: UMI-Probes) is a package that consists of scripts that can be used to test the functionality of each "main" UNICORE components:

- UNICORE Gateway,
- UNICORE Registry,
- UNICORE/X (including CIP component),
- UNICORE SMS implementations,
- UVOS,
- UNICORE Workflow Factory.
- UNICORE Service Orchestrator.
- UNICORE Common Information Service.

Additionally, a script that checks appropriate functionality of any application installed in the Grid environment is available.

All scripts can be easily used as probes in Nagios-based monitoring environment. Each of them presents the result of the test in well-known Nagios probes format and is compatible with Nagios probes development standards. The main programming languages of probes are Perl, Java and Groovy. The majority of scripts are dependent on availability of standard UNICORE clients: UNICORE Commandline Client and UVOS Commandline Client (they are available in the newest EMI release) and some Perl modules: perl-Error, perl-Sort-Versions, perl-XML-RSS (all available via CPAN).

2 System Administrator Documentation

UMI-Probes component is available as rpm package, available to install in Red Hat-compatible systems. The following guides are ensured to work on Scientific Linux 5.5 and Fedora 14.

2.1 Installation Guide

As mentioned above, probes are written in Perl, Java and Groovy and use UNICORE standard clients. Therefore, before installing probes package (which is provided by ETICS as rpm, src.rpm, binary tar.gz and source tar.gz) administrator need to provide dependencies. Assuming that EMI repository is enabled, there is need to execute one command:

```
# yum install perl-Error perl-Sort-Versions perl-XML-RSS \
unicore-ucc unicore-uvos-clc
```

After this, described component can be installed using simple command:

```
# rpm -i unicore-monitoring-probes-2.0.0-1.noarch.rpm
```

This package provides:

- Nagios commands configuration in `/etc/unicore/monitoring-probes/commands.cfg`
- Documentation and Licence in directory `/usr/share/doc/unicore/monitoring-probes/`
- Probes in directory `/usr/libexec/grid-monitoring/probes/pl.plgrid/UNICORE/`

Each probe is placed in directory named as probe itself and consists of main test program (executable script named as probe with `.pl` extension), readme in text format (with `.README` extension) and readme in html format (with `.html` extension). There can be other files (groovy scripts or java classes) that are internally used by tests.

2.2 Configuration Guide

Each probe needs appropriate configurations:

- configuration of related UNICORE client,
- logging configuration for probe,
- configuration of probe itself.

All configuration-making processes can be easily automated by using package UMI-Autoconf, released by PL-Grid project.

Samples of UNICORE **clients configurations** are attached in their packages and has to be changed to be able to connect to the Grid. This is recommended to test prepared clients configuration by executing commands:

```
$ uvos-clc -b getMyIds
$ ucc list-sites
$ ucc list-storages
$ ucc run /usr/share/doc/unicore/ucc/samples/date.u
$ ucc workflow-submit \
/usr/share/doc/unicore/ucc/samples/workflows/date-with-stageout.swf
```

If all of this commands ends without any error that means that clients configuration is ready to use with probes.

The second step is to prepare log4j **logging configuration**. This is standard log4j configuration file that will be used by UNICORE clients (both UCC and UVOS CLC are written in Java), samples are placed in files `/etc/unicore/ucc/logging.properties` and `/etc-/unicore/uvos-clc/log4j.properties`. Each probe needs two configuration files: for standard execution and for debug purposes. There are strict naming conventions: files has to be named `log4j-[clientname].properties` or `log4j-[clientname]-debug.properties` (where `[clientname]` is `ucc` or `uvosclc`). Every probe at each run looks for appropriate logging configuration to use in directories in the following order:

1. location of probe configuration,
2. location of UNICORE clients configuration,
3. location of logging directory of each probe (the least recommended way).

If configuration is not found, probe will not start and will display suitable message.

Finally, the third type of configuration is **probes configuration**. Every probe gets its "what is to be tested" information from configuration file. In some cases few probes can use the same configuration file (especially if they are used to monitor one grid site). The structure of file is pasted below:

```
# Comments need to be started with hash
# Comment

UCC_PATH="/usr/bin/ucc"

# Above line means that variable UCC_PATH is set to /usr/bin/ucc
# (quotes are mandatory)

LOGS_DIR="/var/log/unicore/monitoring/icm.edu.pl"
```

In each probe configuration there is a section that describes what values need to be set in each probe configuration file.

2.3 Probes Reference Card

All probes are written using Nagios probes standard. That means that every probe

1. uses Perl as the main programming language (but disables the usage of Nagios embedded perl),
2. has definable directory for storing logs and temporary files,
3. has the ability to set timeout for probe execution (option `-t` or `--timeout`)
4. has the ability to set verbosity level (option `-v` or `--verbosity`) to one of
   - 0 → prints only one line with status,
   - 1 → default, prints line with status with optional debug info,
   - 2 → prints data like `-v 1` and additionally information about probe environment (configuration parsing, client running, debug info from UCC),
   - 3 → prints data like `-v 2` and disables deletion of temporary files after even successful execution,
5. shows readme with `-h` or `--help` flag given,
6. shows version of every probe with `--version` option,
7. puts shell command into log file before execution.

Descriptions of all the probes are attached into next sections:

2.3.1 check_application

Usage

```
./check_application.pl [OPTIONS] [-f configuration_file] <job_file> <condition>
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description This Nagios plugin tests any application accessible via UNICORE by executing job and checking given condition on output files. Requires Perl in version 5.10.0 or further and UCC in version 1.4.0 or further.

Condition format is well-known, similar to boolean expressions in C language. You can use `and` and `or` operators, files (local files and files that are output from job - add `#` before filename) and functions:

- `equals(file1,file2)` - returns true if both files exist and are equal,
- `not_empty(file1)` - returns true if file1 exists ans is not empty,
- `valid_pdf(file1)` - returns true if file1 exists and is valid pdf file,
- `not_fully_equals(file1,file2,diff)` - returns true if file1 differs from file2 at maximum diff lines (diff is an integer number),
- `grep(string,file1)` - returns true if a string exists in file1,
- `contains(fragment, output_file)` - checks if output_file contatins fragment.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4.0 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!
- `REGISTRY_URL`: address of the Registry
- `SITE_NAME`: name of site to run jobs on
- `LOGS_DIR`: directory where script should store UCC logs and temporary files

POSSIBLE OUTPUTS WITH MEANING

- `UNKNOWN: Wrong condition syntax! See README file`: Check testing condition, there is probably some error. After this line there will be full explanation.
- `WARNING: Unable to pass application condition string`: A given condition is not fullfilled. Warnings after this line will help to find the problem. Job object will be deleted but all generated files will not be touched and can be revised to find the problem.
- `CRITICAL: Application is not present in IDB`: Job cannot be run because of the unavailability of required application (or missing entry in IDB)
- `CRITICAL: TSS is not available`: Unable to run job because of some intrnal TSS problem or lack of requested resources (CPUs, memory, etc). Detailed output from UCC will be shown.
- `CRITICAL: Error waiting for job to finish`: Job was started but cannot be finished on a target system. Check logs at server side.
- `CRITICAL: Some output files are missing`: Some output files that were marked to be produced by application test were not generated. Probably job failed in an early stage, check the stdout and stderr files.

- `CRITICAL: Cannot submit job`: Unable to submit job - unknown error. See logs to determine the reason.
- `OK: Job submission succeded, time elapsed: XXX`: Everything is OK.

Example

```
./check_application.pl /etc/nagios/PL-Grid/R/r.u "equals(#stdout,/etc/nagios/PL-Gr
```

Checks, if stdout from job is equal to /etc/nagios/PL-Grid/R/r_output.stdout and if file Rplots.pdf from job output is not an empty valid pdf file.

2.3.2 check_cip

Usage

```
./check_cip.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks CIS Info Provider, part of UNICORE/X contaier. Written in Perl, v.5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.4 or higher and a proper configuration file for this client.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains CIP you want to check
- `CIP_URL`: URL of CIP to check
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `OK: CIP works, total jobs: ...`: CIS Info Provider works and gives appropriate information of total processed jobs.
- `WARNING: CIP works, but does not process information`: CIP works, but Job Sensor seems to not process information of total jobs (got value is 0).
- `WARNING: CIP responds but seems to be not initialized`: CIP works, but the response seems to be the default one. That means the lack of CIS initialization at the start of UNICORE/X. See `uas.config`.
- `CRITICAL: CIP does not respond`: CIP probably does not work. See next lines for more information.

2.3.3 check_cis

Usage

```
./check_cis.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks Common Information Servicer. Written in Perl, v.5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.4 or higher with CIS extenstions and a proper configuration file for this client.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains CIS you want to check
- `CIS_URL`: URL of CIS to check
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `OK: CIS works and gives information of ... CIPs`: CIS responds and aggregates information of ... CIPs.
- `WARNING: CIS works, but does not give any information about CIPs`: CIS works, but does not contain any reponse from CIP.
- `CRITICAL: CIS does not work`: Unable to connect to CIS. Next lines will show more detiails on failure.

2.3.4 check_freespace

Usage

```
./check_freespace.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds
- `-w W` or `--warning W`: sets warning threshold to W bytes
- `-c C` or `--critical C`: sets critical threshold to C bytes

Description Plugin checks free space on SMS of given url. Written in Perl, v.5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.4 or higher and proper configuration file for this client. Default values are 10GB for warning and 1GB for critical.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains SMS you want to check
- `SMS_URL`: URL of SMS to check free space
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `WARNING: SMS at ... doesn't provide free space information`: Probe was able to contact SMS, but this service does not publish information on free space. You should consider updating UNICORE/X component.
- `UNKNOWN: Cannot get access to the SMS ...`: Unable to get data from SMS. More information should be provided by check_sms probe.
- `OK|WARNING|CRITICAL: There is ... of free space on ...`: Status of this message depends on the amount of free space on SMS. If there is more than `warning` threshold, it is `OK`, if between `critical` and `warning`, it is `WARNING`, if less than `critical`, it is `CRITICAL`.

2.3.5 check_gateway

Usage

```
./check_gateway.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin version
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks UNICORE Gateway functionality. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Plugin connects to the Gateway web interface (you need to provide valid UCC configuration file path in `check_gateway.config`) and matches pattern to get all information about registered USites (this is done by class `CheckGateway`, compile it (use javac) and place in `check_gateway.pl` folder).

CONFIGURATION FILE FOR PLUGIN

- `JAVA_PATH`: path to `java` command (type simple "java" if java is available via PATH environment variable)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `GATEWAY_URL`: address of the Gateway interface you want to check
- `REQUIRED_VSITES`: List (separated by comma or spaces) containg names of services you are sure to be registered in this Gateway. If at least one of them will not be available, the plugin will exit with WARNING status.

POSSIBLE OUTPUTS AND THEIR MEANING

- `WARNING: Unable to get Gateway output or no services found`: Connection to Gateway is OK, but plugin could not match any service pattern in the Gateway web interface. There are no registered services or Gateway version you are checking is unsupported.
- `OK: Gateway works properly, ... services found`: Everything is OK
- `WARNING: Required services not found: ...`: Connection to Gateway is OK, but the script could not find some services you marked as required. See log for details.
- `CRITICAL: Connection refused`, `CRITICAL: Connection timed out`, `CRITICAL: No route to host, check connection`, `CRITICAL: Unknown gateway host`: Unable to connect Gateway or network problems. Check if Gateway is running and is not blocked by firewall.
- `UNKNOWN: Keystore password you provided is wrong`, `UNKNOWN: Keystore path you provided is not valid`: A given configuration file seems to be invalid, check it with UCC
- `CRITICAL: SSL connection cannot be trusted`: Probe received SSL exception. Check if the Gateway trusts the certificate you are using to connect with.

2.3.6 check_registry

Usage

```
./check_registry.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks UNICORE Registry. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Plugin connects to the Registry using UCC (in version 1.4.0 or higher) and uses command `ucc system-info` to get all services registered in Registry. Next checks if all services marked in configuration file as "required" are available.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4.0 or higher)

- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of the Registry you want to check
- `REQUIRED_SERVICES`: comma-and-space separated list of service that are required to be placed in the Registry at each probe run (can be an empty string)
- `LOGS_DIR`: directory where script should store UCC logs and temporary files

POSSIBLE OUTPUTS WITH MEANING

- `CRITICAL: Unable to connect the Registry`: UCC could not connect to the Registry. Check if it is running and avaialable at the Gateway.
- `CRITICAL: None services found`: Connection is OK, but script could not parse output from UCC, probably UCC version you use is unsupported (or just no service are registered in the Registry yet)
- `OK: Registry works, XXX services found`: Everything is OK.
- `CRITICAL: Some required services not found: XXX`: Some services marked in configuration file as required are not available at the moment.

2.3.7 check_servorch

Usage

```
./check_servorch.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin that checks Grid Resource Information System, one of Service Orchestrator package components. Written in Perl, v.5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.3.1 or higher and proper configuration file for this client. Plugin gets list of computation resources in GRIS.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.3.1 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of the registry that contains Workflow Service you want to check
- `SERVORCH_URL`: URL of Service Orchestrator
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `OK: Service Orchestator appears to work, sites registered in GRIS: ...`: GRIS works and has at least one registered computational resource, so any WorkAssignment sent to ServiceOrchestrator should be executed unless there are problems with resources themselves.
- `CRITICAL: Unable to find any resource in GRIS` There is no Target System in GRIS so Service Orchestrator is unable to execute any job.
- `CRITICAL: Unable to find ... in the Registry`: SO matching given name cannot be contacted.
- `CRITICAL: Unable to find GRIS used by ... in the Registry`: ServiceOrchestrator or associated GRIS is not available in the Registry
- `CRITICAL: Registry ... cannot be contacted`: Unable to connect to the Registry and get GRIS EPR. In the next line you will find cause of failure.
- `CRITICAL: Some VSites are registered in GRIS but do not respond: ...`: Probably some UNICORE/X instances died recently.

2.3.8 check_sms

Usage

```
./check_sms.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin that checks Storage Management Service of UNICORE. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.4.0 or higher and proper configuration file for this client. Plugin generates pseudo-random file with given size in LOGS_DIR. Then puts file (measuers upload speed), gets this file (measures download speed) and compares this files. If they are equal, test ends with OK message.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4.0 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains Workflow Service you want to check
- `LOGS_DIR`: directory where script should store UCC logs and temporary files
- `SMS_ADDRESS`: URL of SMS to be checked, can be given in any ucc-acceptable form, i.e. https:// URL or u6:// meta-URL.
- `FILE_SIZE_KBS`: size of file used to test (default: 1000 Kbytes)

POSSIBLE OUTPUTS WITH MEANING

- `CRITICAL: Unable to generate random file to test`: Unable to initialize test file. Check if device /dev/urandom is available.
- `CRITICAL: Unable to get access to the SMS at XXX`: Unable to connect to the SMS. Check if appropriate UNICORE/X is running and available for monitoring user.
- `CRITICAL: Unable to write file at server side on XXX`: Check directory permissions on the server (IOException).
- `CRITICAL: Unable to upload file to XXX`: Next lines should display reason of inability to upload.
- `CRITICAL: Unable to upload file - XXX is not a valid location`: Probably SMS is not available, check UCC log below.
- `CRITICAL: Download succeeded, but downloaded file is empty (XXX on X-XX)` : Probably plugin tried to download non-existing file. Check is file exists.
- `CRITICAL: Downloaded file differs from uploaded one.`: Should not happen. If occurs, local and remote files are not deleted.
- `OK: SMS at XXX works with U:XXX kB/s, D:XXX kB/s`: Everything is OK.

2.3.9 check_unicorex

Usage

```
./check_unicorex.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks main part of the UNICORE/X containter: TargetSystemFactory avialability and ability of creating TargetSubmissionService instances. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Plugin lists all TargetSystemFactories in the Registry whose name is equal to one od the configuration options The amount of created Target Systems is also checked (if none is available script tries to create one).

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4.0 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of the Registry
- `SITE_NAME`: name of site you want to check
- `LOGS_DIR`: directory where script should store UCC logs and temporary files

POSSIBLE OUTPUTS WITH MEANING

- `OK: Found XXX TSS instances`: TSF works and there is at least one TSS.
- `CRITICAL: Unable to find requested TSF on accessible sites list`: There is no such TSF on list of found in the Registry and accessible sites. Probably the site is not available in the Registry or you do not have valid credential to connect to it.
- `CRITICAL: TSF found but TSS cannot be created`: No TSSs found and UCC is unable to create any. Check server-side logs.

2.3.10 check_uvos

Usage

```
./check_uvos.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin that checks UVOS connectivity and administration interface. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UVOS Commandline Client in version 1.32 or higher and proper configuration file for this client. Plugin executes command `getMyIds` from UVOS administration interface and looks for used certificate distinguished name in the output. Compares it using simple Java class and shows the result.

CONFIGURATION FILE FOR PLUGIN

- `UVOS_CLIENT_PATH`: absolute path to UVOS Commandline Client binary (in version 1.32 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `UVOS_URL`: URL of UVOS server (with scheme, hostname/ip and port!)
- `LOGS_DIR`: directory where script should store UVOS CLC logs and temporary files

POSSIBLE OUTPUTS WITH MEANING

- `CRITICAL: Unable to contact UVOS`: Cannot connect to UVOS service, check UVOS CLC configuration and connection - probably network problems. In next line you will find cause of failure.
- `CRITICAL: Access Denied`: Could not authenticate in UVOS, check user rights.
- `WARNING: Unexpected output (dn not found: ...)`: Connecting to UVOS and executing command succeded, but used DN is not found. Check the output from the command.
- `OK: UVOS works, identity matches`: Everything is OK.

2.3.11 check_versions

Usage

```
./check_versions.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin checks if installed UNICORE components are up-to-date. The newest versions are fetched from UNICORE last releases RSS feed (accessible at http://www.unicore.eu/_script/unicore_releases.rss). Probe is written in Perl, v.5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.4 or higher and proper configuration file for this client.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.4 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry from where services and their versions will be downloaded
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `UNKNOWN: Unable to connect to / parse data from ...`: Unable to connect to some registered service or to parse data from its resources document. More information should be provided by appropriate probes.
- `UNKNOWN: Unable to get retrieve UNICORE releases rss feed`: Unable to download latest versions on UNICORE components from http://www.unicore.eu/_script/unicore_releases.rss. Probe should be updated.
- `OK|WARNING: Up-to-date: ..., outdated: ..., unknown: ...`: Status of this message depends on number of outdated services. If all discovered ones are in the newest versions, status is `OK`. Otherwise it is `WARNING`. In this case next lines will give additional information on these services.

2.3.12 check_workflow

Usage

```
./check_workflow.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description This is the plugin that does integration test of complete UNICORE installation. Requires UCC in version 1.3.0 or higher, because core of plugin is implemented in Groovy as UCC script (it gives eightfold time decrease in comparison to previous version). Plugin executes workflow in UNICORE environment asynchronously. URL of previously submitted instance is saved in cache file. Firstly plugin lists TSFs in the Registry and creates TSSs if there is such need. Next script checks status of workflow submitted before. If it is OK, location of ouptut file is got (by queryint LocationManager) and content of that file is compared to data sent before. Next test uses Tracer service to get workflow execution statistics. If all actions are successsful, both - workflow files and instance are deleted from grid evironment. Finally plugin submits simple workflow to the grid. Status of the new object will be checked in the next run of the script.

If the whole test is successful you can assume that every module of standard UNICORE installation (except CIS or CIP) works fine.

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.3.1 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains Workflow Service you want to check
- `WORKFLOWSERVICE_NAME`: URL of Workflow Service (or any part of this URL while it is not ambiguous)
- `GLOBAL_STORAGE_URL`: URL of Global Storage to use
- `TEST_FILE_PATH`: Location of file with workflow description
- `CACHE_FILE`: Location of file where workflow url will be stored

- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH DESCRIPTION

- `OK: Wokflow succeeded! Time elapsed: ...`: Your UNICORE installation works fine!
- `CRITICAL: Unable to connect to the Registry`: UCC cannot connect to given Registry URL.
- `CRITICAL: Some TSFs are not available in the Registry: ...`: There are TSFs that are used by workflow description as CandidateHosts but there are not available in given Registry, so worfklow cannot be executed successfully.
- `CRITICAL: Unable to upload test file`: Global Storage URL you given is not properly defined (script cannot get supported file protocols) or there is server-side failure in creating file.
- `CRITICAL: WorkflowFactory does not respond`: WorkflowFactory at url given in configuration is not found in the Registry or is registered but cannot be contacted.
- `CRITICAL: Workflow is still running after ... seconds`: Workflow is in running state since previous run. Usually that means that ServiceOrchestrator is not available or cannot accept WA by choosing TSF.
- `CRITICAL: Workflow at ... does not exists` Previously submitted workflow does not exists. Probably termination time of WF instance reached.
- `CRITICAL: Workflow execution does not seem successful`: Execution of workflow failed for unknown reason.
- `CRITICAL: Could not resolve output file location using LocationManager`: Workflow execution succeeded but output file cannot be mapped using LM service.
- `CRITICAL: Unexpected output: ...`: Workflow succeeded and output was retrieved but it differs from the sent file contents. Really strange...
- `CRITICAL: Tracer service is not available`: Unable to find any Tracer service in the Registry, so workflow statistics cannot be retreived.
- `WARNING: Workflow succeded, but some TSFs available in the Registry were not used`: Script found TSFs in the Registry that you do not use. Rerun installation script to avoid this message.
- `CRITICAL: Unable to list target systems from ...`: TSF is available in the Registry but does not respond.
- `CRITICAL: Unable to create TSS in ...`: TSF is available, but none TSS service is available. Probably you do not have access to one of TSFs available in the grid.
- `CRITICAL: There are TSFs registered in the Registry, but unable to contact with: ...`: All TSFs are available in the Registry, but some of them cannot be contacted. Probably Gateways before these TSFs do not accept your certificate.

2.3.13 check_workflowservice

Usage

```
./check_workflowservice.pl [OPTIONS] [-f configuration_file]
```

OPTIONS

- `-h` or `--help`: prints short usage tips
- `-V` or `--version`: prints plugin versions
- `-v X` or `--verbose X`: sets verbosity level to `X` (see UMI docs)
- `-t Y` or `--timeout Y`: sets plugin execution timeout to `Y` seconds

Description Plugin that checks Workflow Service avaialibity. Written in Perl, v5.10.0 (requires commons.pm module in plugin directory or one level higher). Needs UCC in version 1.3.1 or higher and proper configuration file for this client. Plugin gets information about all available workflows in given Workflow Service and counts them

CONFIGURATION FILE FOR PLUGIN

- `UCC_PATH`: absolute path to UNICORE Commandline Client binary (in version 1.3.1 or higher)
- `UCC_CONFIG`: absolute path to valid configuration file for UCC (remember to set absolute paths to keystore and trustore in this file!)
- `REGISTRY_URL`: address of registry that contains Workflow Service you want to check
- `WORKFLOWSERVICE_NAME`: URL of Workflow Service (or any part of this URL while it is not ambiguous)
- `LOGS_DIR`: Path to logs directory

POSSIBLE OUTPUTS WITH MEANING

- `OK: Workflow Service works normally, found workflows: ...`: Workflow Service works and contains 0 or 1 active workflow.
- `CRITICAL: Unable to find ... in the Registry`: Workflow Service matching a given name cannot be contacted. If you do not specified any workflow name that means the Registry does not contain any Workflow Service.
- `CRITICAL: Found more than one Workflow Service that name matches ...-, none of them was checked`: Name of Workflow Service you specified is ambiguous. If you do not given any name that means that Registry contains more than one Workflow Service.

- `CRITICAL: Workflow service ... found in the Registry, but seems to be down`: An exception "Connection refused" was thrown when tried to access WorkflowService that was found in the Registry.
- `CRITICAL: Registry ... cannot be contacted`: Unable to connect to the Registry and get Workflow Factory EPR. In next line you will find the cause of failure.
- `CRITICAL: WorkflowFactory service exists in Registry but does not respond`: Unable to get service properties document from WorkflowFactory service.

3 Developer Guide

3.1 API Documentation

All described probes use one-file Perl library that makes writing new ones quite easy. This file is located at `umi2/commons.pm` and consists of several "public" (exported) functions:

- `exit_plugin` - the most preferred way of exiting probes. Takes two arguments. The first is status line in format `[STATUS]: [message]` where `[STATUS]` is one of: `OK, WARNING, CRITICAL, UNKNOWN`. This line will be shown as probe output in every verbose mode (and of course appropriate exit code of script will be set to meet Nagios API requirements). The second argument are optional debug data. Provided string will be evaluated and displayed by probe in first verbose mode.
- `setup_plugin` - this function has to be called at the beginning of probe execution. It takes two parameters: location of readme file (to display its fragment as help message if `--help` option is provided) and version of probe (that will be displayed when a user calls script with `--version` flag). Procedure gets options from the commandline and stores it in external `%config` variable. Next sets timeout of probe to value specified by a user in command line or 300 seconds by the default. Then loads configuration file and also stores the data in `%config` hash. Finally changes working directory to `[LOGS_DIR]/[plugin_name]` - that is place where log files and temporary files will be stored.
- `message` - shows message to user according to requested verbose mode. Takes two parameters - one is message to display and the second is the least verbose mode to attach this message to the output.
- `check_conditions` - checks some conditions and if any is met, exits probe. Takes one argument - array of conditions. Every element is hash with three keys: `test`, `output` and `show_debug`. First, `test`, is the logical condition that will be evaluated. If evaluation gives false message (empty string or 0) function tests next one. Otherwise it calls `exit_plugin` subroutine with `output` and `show_debug` parameters. If `show_params` is *`0`* then debug messages will not be shown.
- `create_temp_file` - creates a temporary file with name provided as first argument. The file is stored in current directory (set by `setup_plugin`) and is saved to be deleted at the end of probe execution.

- `check_config` - checks if variable required by script execution is available in configuration file. Takes two parameters: first is a comma-separated list of configuration variables. If any of them is not given in configuration, probe exits with `UNKNOWN` status. But if there is a second (optional) parameter defined, the function does not quit probe and just returns a number of undefined options.
- `run` - executes external command with timeout checking. The command can be one of: `ucc`, `uvosclc`, `java` and is given as first argument. The second parameter is a line of arguments to be passed to command (configuration files for UNICORE clients and Registry URL for UCC are included in command line by script). The third argument is path to the file where output will be saved (it is preferred to pass path returned by above described `create_temp_file` or just */dev/null* if probe output does not matter). The fourth parameter is optional and should be set if stderr stream has to be attached to output (if verbose mode is more or equal 2 this flag is set by default). Both UCC and UVOS CLC are executed with appropriate environment variables that sets path to log4j properties file.
- `check_file_existence` - checks if file passed as the first parameter exists in file system. Can be easily used if developer is not sure if script that is to be executed is properly defined.
- `is_debug_enabled` - returns if verbose mode is more or equal two.

Additionally there some other options of library that developers may need:

- `$main::CLEANUP` variable - if set, it is executed at the end of probe execution. Can be set to for example clear Grid objects at the end of each execution (see `check_application` source code).

3.2 Build Documentation

Build of component is done by UNICORE packman tool (in fact modified version of packman, see `packaging/packman-opts.xml` file). There are three main targets of packaging script:

- `./packaging/packman.sh probes-clean` - deletes `.class` files and temporary build workspace directories
- `./packaging/packman.sh probes-compile` - compiles two `.java` classes (used by check_uvos and check_gateway)
- `./packaging/packman.sh all-rpm` - packages component into four files: binary rpm, binary tar, source rpm and source tar.

Documentation is built using UNICORE docman tool. This can be run by command: `./packaging/docman.sh`.

4 Component changelog

4.1 Version 2.0.10 (2011-10-11)

First official EMI release. All probes are tested (by unit- and functionality tests), documentation is reviewed.

4.2 Version 1.9.9 (2011-08-30)

- finished refactoring of all probes, outdated documentation of package

4.3 Version 1.9 (2011-08-12)

- completed ETICS packaging scripts, first EMI release

4.4 Version 1.4 (2011-05-27)

- added new probes: check_freespace, check_versions, check_cip, check_cis

4.5 Version 1.3 (2011-01-25)

- finished refactoring of 4 probes: check_gateway, check_uvos, check_workflow and check_workflowservice
- all UMI scripts separated into two sets: UMI-Probes and UMI-Autoconf

4.6 Version 1.2 (2010-12-01)

- check_servorch probe for checking Service Orchestrator

4.7 Version 1.1 (2010-10-01)

- check_workflow:
  - checking availability of required or unused TSFs
- installation:
  - new format of dependency tree (included in samples)
  - faster problem finder (use of Nagios predictive check, smaller intervals)
  - automated generation of Logger services for each probe

- documentation:
  - added separate readme files for each probe
  - attached documentation in PDF format

4.8 Version 1.0 (2010-06-23)

First UNICORE Monitoring Infrastructure package public release