Coding styles standards and design definitions

Standards and agreements for source code format

Format and make up of source codes, modules, files, and project configurations must to comply with specification defined in document CppCodingStyleGuidePoco.pdf and CppCodingStyleGuideGoogle.pdf if definition is not contradicts with first one.

Agreements for comments and documentation tags

1. Comments and documentation tags must to comply with specification from:

http://www.yolinux.com/TUTORIALS/LinuxTutorialC++CodingStyle.html

and accomplish to the doxygen specification.

2. Header files must to have general description and documentation tags.
3. Namespaces must to have short description in one of the module defined.
4. Global declarations and definitions must to have short description that can be grouped for set of similar items.
5. The doxygen documentation for source files must be generated in HTML format with images and stored zip or tgz form.

No complete covering of all structure items like methods, functions, global declarations, and definitions required.

Example of header file general description

```
/**
 * @file admin.hpp
 *
 * ASM Cluster node admin handler.
 * Implements functionality for admin handler server.
 * Processes admin management requests and process them or proxy to the target handlers that are
 * connected by zmq admin socket.
 *
 * @author bgv <bgv.asm.cluster@gmail.com>
 * @link http://asm-search-engine.com/cluster/
 * @copyright Copyright &copy; 2013 IOIX Ukraine
 * @license http://asm-search-engine.com/cluster/license/
 * @package ASM Cluster node API
 * @since 0.1
 */
```

Agreements for examples and tests

1. Classes, modules, libraries and another functionally closed solution must to have at least one example of usage that covers as many functional items as possible and illustrate by mostly understandable way direct purpose of requirements or usability sense.
2. Complete functionality of application module that can be started separately need to be tested under valgrind tool and have no errors.
3. Unit tests must be done for potentially most rescue calls, functional states, inconsistences, wrong input/output data formats and so on…
4. Functional tests must be created in case of complex logic or interactions of several separated components, services, classes and so on and by special requirements. Test implementation must be checked runtime by valgrind tool and has no errors.
5. Performance tests must be done in case of speed, productivity, bandwidth, etc… characteristics are actual parameters and requested in requirements or main functionality limitations.
6. Stability tests must be created in case of high potential rescue of malfunction cause of quantitative but not logical states like uptime, total or per time period number of requests, number of messages, number of memory allocation/de-allocation actions and so on…

No complete covering of all methods and functions required. Another kind of tests needs to be implemented in special cases according with additional specification and requirements (http://www.softwaretestingsoftware.com/all-types-of-software-testing/)

Agreements about code checks

Source codes must be checked with cppcheck tool with most strict rules and have no errors and notes.