Mule Implementation Standards

This document describes standards for implementation with the expectation that Mule design and Mule coding-standards have been and will be followed.

1. Defined Standards for Exception Handling in Mule

Proper handling of and exception is as important in MuleSoft stack as it is anywhere.

- Developers and architects must have a complete understanding of Java exception-handling
- Read and understand the standard Mule exception-handling documentation.

1.1. Exception-reporting

As with Java, exceptions must never be swallowed unless there is a good reason to do so, and the point at which the exception is swallowed is clearly documented with reasons for doing so.

Exception-reporting must be, at a minimum, logged to file, and optionally (depending customer preference) sent to an alerting system.

Exception-reporting must include at a minimum:

- message correlationID ***or other uniquely identifying details***
- relevant pieces of the message which will aid in tracing and rectifying.
- a short identifier for the type of error (essential for categorizing exceptions - something that operations-staff can use to build procedure and escalation protocols)

1.2. Importance of proper exception handling in integration and web-services

Web-services clients naturally rely on proper HTTP status codes.

IMPORTANT	Proper exception handling is particularly important in web-services due to the widespread effects of inappropriate HTTP-status codes on every web-service client.

Exceptions must be handled and categorized correctly to return the proper HTTP-status code.

2. HTTP status codes

These are well-documented in RFC723x. Developers must

- be familiar with the status codes 1xx through 5xx
- use response codes appropriately from both server and client perspective - all already well documented in RFC 723x, but brief examples are informative:
    - server
        - Validate requests as early in the flow as possible. When a caller passes in incorrectly formatted data, return 400 because that is what it is - a bad request. Failing to validate request data and allowing a bad request to proceed far enough to cause server-error (and thus 500 status-code) is an antipattern and must be avoided.
        - know and practice the distinction between 400 (bad request) and 405 (method not allowed). These are not the same, and when inappropriately used this can cause wasted time.
    - client
        - when receiving responses from HTTP calls, use the HTTP status code to detect whether or not the response is successful. It is known that many HTTP-based services exist which do not respect HTTP specification. In this situation it is appropriate to "firewall" such responses. See section below.

2.1. Why standard HTTP status-codes are important

HTTP status-codes are international, industry-standards. All major platforms conform to these because this facilitates determination whether a request succeeded or not is direct, using well-known values in a well-known location (the header), and is a simple, quick numeric equivalence-check.

Web-services that do not return trustworthy status codes quickly lose the trust of developers/consumers, and defensive coding [1: reading the body and checking for presence of strings - wasteful and fragile at best) to check the body or content of HTTP requests becomes the norm] becomes the norm. Productivity loss is significant, and lack of trust spreads throughout the user/developer base. The situation becomes worse if the defensive coding anti-pattern spreads past the entry-point into the rest of the code.

2.2. Defense against web-services that do not observe HTTP-status specification

Web services and applications that do not respect proper http-status codes cause substantial productivity loss due to the defensive coding that needs to take place. The appropriate solution is:

- put a proxy at the request-receipt location
- do whatever pragmatic coding has to be done right there in the proxy (this is all that proxy should do). In other words, place a tight seal on the bad behavior to isolate it.
- Once past that intermediary, process the response as it would have been done.

Summary: the proxy does a single job: remediate the http status-code, and then leave the rest to standard http-response parsing.

The reason to do it this way is ensure that if the HTTP response ever does become well-formed, the repairing proxy can be removed without touching any other code.

3. JMS and HTTP headers

- standard headers:
  - use these appropriately for their documented, intended purpose. "Piggybacking" unrelated data into headers is bad practice, a security-risk and can cause requests to be rejected in production-environments
- custom headers:
  - define project-standards, document them clearly, use appropriately, monitor and enforce observance of the standards.

4. Transactions

Transactions in Mule are active only by specific inclusion (not automatically). Developers must plan, define, document and use transactions to ensure predictability of actions taken on or with data.

5. Streaming

Certain connectors support streaming, which should be used whenever it is consistent with requirements (example: if requirements state that a file is to be use for specific reasons, streaming is inappropriate). Processing of large files or database result-sets