MuleESB Design Standards

This document defines MuleESB design standards - the creation and organization of flows and sub-flows for reusability and maintainability.

1. Structure and Organization

Mule applications *need to be structured, planned for reuse and maintenance* as with any application development. Flows must be organized and decomposed logically to aid efficient maintenance and comprehension by developers and architect.

- Larger integrations should be decomposed into separate flow-files, each flow-file named to aid understanding

1.1. Copy/paste

Identify common functions and processing, and organize for reuse by locating logically related parts in sub-flows or private-flows, then reuse via flow-references.

Flow-references can refer to flows/sub-flows that:

- are in the same project
- are imported (via Spring-import) from included library-jars

WARNING	Copying and pasting is not effective or acceptable reuse for the same reason it is not acceptable in any programming language.

1.2. Standard flows reusable by multiple integrations

Flows that can be used by multiple integrations must be standardization, tested, and packaged for effective and reliable reuse. See Modularizing Configuration Files for how to do this.

Reusable, packaged flows must be (for the source) version-controlled and (for the packaged binary jar) be put in an access-controlled binary-artifact repository - see Build System Standards. This approach will give confidence that all users who are using the same versioned-artifact are using the same flow-functionality.

2. Configuration-externalization

Configurations for the following values are to be externalized to property-files.

- host-names
- URLs and base-path of URLs
- port numbers
- file-paths (eg. as used in SFTP connector)
- user-names
- timeout parameters, TTLs, object-store parameters
- passwords ***must be encrypted***

Property-names

- use property-names that are intuitive and informative.
- define a consistent 'hierarchy' for property-groups - examples:
  - `sfdc.marketing.user.name`
  - `sfdc.marketing.user.password`
  - `mysql.productDB.user.read` , `mysql.productDB.user.admin`
  - use lower camel-case
- group related properties together in properties-files; separate groups by a blank-line
- use comments in properties-files at the top of each section
- use only truly industry-standard acronyms as parts of property-names; made up acronyms these confuse users.
  - Exception: customer-mandated/common-usage acronyms
  - correct: sfdc, http, sftp, nio, tcp
  - incorrect: anything that has been made up
- if any customer-mandated property-names are cryptic, put a comment above stating the meaning

3. Legibility of flows

Certain practices can make flows easier to follow and troubleshoot:

- use flow-ref components to limit the vertical and horizontal spread of flow-lines: do not nest any "flow-diverging" (choice, scatter-gather) deeper than 3 levels.

Keep in mind that many Mule developers work on laptops, not 24" monitors. Having to repeatedly scroll left or right in Message Flow view, or scroll up and down pages of XML to trace

a flow impairs ability to understand a flow.

Properly decompose integration-tasks into logical pieces where each flow has a well-defined responsibility within the larger task.

See Mule Coding Style for flow, component, scope naming standards that help to make flows clear.

4. Database insert and update flows

Many applications still rely on relational databases and use deeply nested object-structures. If the inserts are done using sequential JDBC-connectors, ensure that flows which persist deeply-nested objects into databases:

- are well designed and clearly documented
- *use transactions* so that partial database-inserts do not occur.

Some use-cases warrant a more extensible and maintainable approach using ORM tools. See RDBMS pattern.

5. Transactions

- Design flows with data-integrity in mind
- Use transaction-scope effectively by designing flows to minimize the inclusion of extraneous operations and connectors in the transaction-scope.