Coding Guidelines in ADITO - Short Version

AID001

ADITO Academy

21.03.2022

Index

1. Purpose

This document defines rules to customize ADITO.
For explanation, examples and more detailed information see AID001 Coding Styles

2. General rules

- ADITO Code must be written in English
- Naming of all code elements should be self-explanatory
- LOGGING-imports, aswell as LOGS themselves, have to be deleted (see Code Review). The rule does not apply to error-logs.
- If nothing is defined in this document, the usual JavaScript guidelines are to be followed
    - mozilla js guidelines
    - w3schools js conventions
- Always use semicolons at the end of a line if you can
- Do not add personal info in codes
- Do not refer to tickets, CRs or sprint numbers (doesn't apply to file names in Liquibase)
- **Do not change code in default libraries in your project**
    - Copy needed libraries and functions (see project specific)
    - Report bugs to the xRM Teams

3. Coding (technical)

3.1. General

- Never write something twice, rather create a custom function
- In your custom library always start by creating a class
- Wherever possible, use the methods .forEach(), .map() or for(in)
- Dont use prototype functions if not necessary
- w3schools js best practices

Do NOT use "for each … in" loops, they are deprecated. Use the function as mentioned above.

3.2. Liquibase

Liquibase is NOT an ADITO specific tool. For further technical information checkout https://docs.liquibase.com/home.html

3.3. SQL

- Use the SqlBuilder whenever possible
- Always use qualified column names (TABLENAME.COLUMNNAME)
- For relation tables or combined indexes NEVER forget to use BOTH columns (i.e. OBJECT_ROWID & OBJECT_TYPE)
- For new tables always add the columns USER_NEW, USER_EDIT, DATE_NEW, DATE_EDIT (and use them in your entity)
- Set a columnAlias if you are using expressions in a record container to easier analyse your SQL
- Functions that encapsulate only one SQL return the SqlBuilder

4. Coding style (format)

For a semi-automatic formatting of code, use the shortcut [SHIFT] + [ALT] + [F].
Always check your code afterwards, because sometimes the code is misformatted!

- Each curly bracket deserves its own line in your code
- If-Statements should not be written in one line and always need curly brackets
- Make sure your indentation makes code more readable
- Operators, variables, numbers, etc. are separated by blank spaces

5. Spelling & Wording

5.1. Project specific

- For contexts, entities, database tables and libraries use a project specific prefix
- For new fields in existing entities and database tables use the prefix as well

5.2. JDito identifiers

- Each variable must be declared in a seperate row
- Variables must be written in dromedaryCase
- Variable names that only contain one letter should only contain temporarily needed data

- Constants are written in capital letters (i.e. const MYCONST)
- Functions and methods are written in DromedaryCase, their parameters are preceded by the letter "p"

5.3. ADITO models

- Use descriptive English names
- Typical structure: (ProjectPrefix_)ModelCamelCase_suffix
- Exceptions:
    - Contexts do not have suffixes
    - Entity fields: CAPITAL_LETTERS for database fields and CamelCase for all others
    - dromedaryCase is also used for: Executables and record containers
    - Database tables and colums are always written in CAPITAL_LETTERS
- For a more detailed overview and specific examples of ADITO models see longer Coding Guideline documentation.

5.4. Liquibase

- Use your own liquibase structure in a project (folder for project, with sprint subfolders, struct and data)

- Always use DATETIME instead of TIMESTAMP due to the 2038-Problem
- Boolean values have to be stored in TINYINT

Operating content	Prefix	Example
creating a table	create_	create_address.xml
inserting new content (keywords, demo data, etc.)	insert_	insert_keyword_offerstatus.xml
adding columns	add_	add_checklist_position.xml
change structure (table, column, constraint, etc.)	alter_	alter_contact_status.xml
updating content (data sets)	update_	update_keyword_medium.xml

Operating content	Prefix	Example
deleting content	delete_	delete_attribute_newsletter.xml
removing structure (including constraints etc.)	remove_	remove_address_contact_nullConstraint.xml.xml

6. Documentation and Comments

6.1. Comments

Unnecessary parts of the comments have to be removed before committing!

It is useful to think: "Will I know why I did that in a month?" and "Will somebody else understand what I was doing here?".

6.2. Documentation of Functions/Methods

- We refer to JSDoc to comment our Functions
- All functions must be documented
- A function documentation has to have at least:
  - A short description
  - All parameters described (see JSDoc param tag & JSDoc types)
  - The return value described (if existend)

6.3. Documentation Property

- Make use of the documentation properties in fields, params, actions, libraries or entities to describe what it does
- Content of the documenation is written in AsciiDoc (.adoc) files.

You have to write a documentation if the logic behind it is not 100% self explanatory!

7. Reserved (Key-)Words

On http://www.reservedwordsearch.com/ you can enter names to be checked automatically (no

guarantee for completeness!).

Other reserved words lists:

- JavaScript
- Java
- General SQL
- MariaDB
- MSSQL
- MySQL
- Oracle