Conventions

Overview
This page describes common conventions to be used for the OpenMRS project. Specific topics should be referenced (linked) from this page. Our goal is not to create a long laundry list of rules, but to provide guidance such that if two developers were to create the same functionality, the resulting work would be 80% similar.

"There is no such thing as an undocumented convention" -OpenMRS

Developing Code for OpenMRS
The OpenMRS community has established a set of conventions regarding code style, format and more which aims at keeping the OpenMRS code consistent and in good shape. Refer to the following pages for more details

- Java Conventions
- JavaScript Conventions

Code Repositories and Conventions
We use the Git for version control and GitHub to host our Code Repositories.

Git
To read about how to use Git refer to: Using Git
If you like watching a video try this one here

GitHub
If you want to contribute code you should read: GitHub Conventions

Code Review
If you want to help out in reviewing code this should help: Code Review

Backporting Code
When bug fixes and/or features are applied to earlier versions of the system, this is called "backporting." OpenMRS uses major.minor.maintenance versioning (e.g., 1.7.3). Maintenance releases are specifically designed to allow for bug fixes to be backported. Sometimes features (not just bug fixes) are applied to previous releases.

- Bug fixes and security patches are generally applied to previous releases that are still being supported.
- New features are sometimes a candidate for backporting, but are treated on a case-by-base basis and should be backward compatible within the minor version.

Data Model Design

Naming tables
- We've chosen to use all lowercase names with underscores (_) between words (e.g., patient_address)
- Use the singular form (e.g., patient_address instead of patient_addresses) except where the singular form is a reserved word (e.g., users instead of user, since user is a SQL Reserved Word)
- Where auditing of rows is needed, use the following attributes:
  1. creator (FK to users)
  2. date_created (datetime)
  3. changed_by (FK to users)
  4. date_changed (datetime)
  5. voided (boolean)
  6. voided_by (FK to users)
  7. date_voided (datetime)
  8. void_reason (varchar 255)

Special Table Name Suffices
- _map is used for many-to-many relationship tables
- _type is used for a table that categorizes objects in another table — e.g., location_type
- _attribute is used for a table that extends another table dynamically — e.g., person_attribute
• _property is used to define Java-ish key/value properties associated with a given object — e.g. user_property

Naming columns
• Use lowercase with underscores (_) between words
• Auto-generated numeric primary keys are <table name>_id (e.g., patient_id and encounter_id)

Datatypes and use of Liquibase
• Do not use int(11) (The display width in bracket is MySQL-specific and not useful), instead use int only
• Use type="BOOLEAN" and defaultValueBoolean="<true|false>" instead of smallint, tinyint for boolean columns (Look out for voided, retired etc properties)
• Use DOUBLE PRECISION instead of DOUBLE
• Use dbms="<databasetypename>" when creating database-specific SQL or changeset
• Until 1.9 it is safe to assume that everyone was using MySQL-only, after that all liquibase changesets/database scripts should be database-independent
• Following should be added for generating uuid in Microsoft SQL server:

```xml
<modifySql dbms="mssql">
  <replace replace="CHAR(38)" with="UNIQUEIDENTIFIER NOT NULL DEFAULT NEWSEQUENTIALID()" />
</modifySql>
```

• autoIncrement="true" in liquibase will create a sequence in postgres with name <tableName>_<columnName>_seq. Use this to manipulate sequences during bulk inserts using the following:

```xml
<modifySql dbms="postgresql">
  <prepend value="SELECT setval('encounter_role_encounter_role_id_seq', (SELECT MAX(encounter_role_id) FROM encounter_role)+1);"/>
</modifySql>
```

• Instead of <modifyColumn> tag, use the more specific tags in liquibase 2.0 for column operations. e.g. <modifyDataType>
• Always include a precondition for each changeset. Although liquibase does have a mechanism for applying only missing changesets, it is not always possible to rely on said mechanism due to maintenance workloads or historical reasons.
  • Some basic rules for preconditions:
    a. When adding a new table, column or foreign key constraint, the precondition should check that none already exists with the same name
    b. When adding a new table, it is recommended to add the foreign key constraints within the same changeset.
    c. When dropping a table, column or foreign key constraint, the precondition should check that one actually exists with the same name
    d. When renaming a table or column, the precondition should check that none exists with the new name and that there is one that already exists with the old name
  • If the precondition fails (for example, new table already exists), mark the changeset as already ran (attribute onFail="MARK_RAN")
• Also document the changeset with a short comment
• The ids of changesets should typically be time stamps of the format YYYYMMDD-HHSS or YYYYMMDD-HH:SS e.g 20140716-1415 or 20140716-14:15 respectively. You can optionally append the ticket number e.g 20140716-1415-TRUNK-4402. For modules, it is recommended to prepend the moduleId to the id of the changeset so that it is guaranteed to differ from those in core or other modules e.g calculation-20140716-1415 where calculation is the moduleId

Global and User Properties

Property Names
• All names must be globally unique amongst property names
• Dot notation, similar to Java properties, with specificity increasing from left to right
• Camel case or underscores to delineate words
• No whitespace (spaces, tabs, carriage returns)
• All properties for modules should begin with the module id + "."
• All portlet properties should begin with the portlet name + "."

Modules
• Module Conventions

Web Application Development

User Interface
• Every OpenMRS business object needs to have a format tag for printing it to the screen.
  • These should be used wherever possible on JSP pages: `<openmrs:formatPatient object="${patient}"/>` is much preferable to `${patient.name}`.
  • The tag should be called `<openmrs:formatXXX ...>` and live in src/web/org/openmrs/web/taglib as formatXXXTag.
  • Parameters that every tag should support:
    • `object="${patient}"` --> This is the object to be formatted.
    • `id="1234"` --> This is the ID of the object to be formatted, which will necessitate a call to Context.getYYYService().getXXX(id).
    • `nolink="false|true"` --> If false (the default) then the output might include a link, e.g. formatPatient might link to that patient's dashboard.
    • `size="compact|normal|tooltip|descriptive|bookkeeping"`.
  • To use a tag, you need to provide object or id (object should take precedence if both are non-null).
  • If a business object is voided or retired, it should be struck through: PIH ID 1234-1 Darius Jazayeri.

Size examples:
- compact -> shortest reasonable representation, e.g. just the name of a patient or form, but the shortname of a concept:
  Jazayeri
- normal -> normal representation, and the default when `size` is not specified:
  PIH ID 1234-1 Darius Jazayeri
- descriptive -> The normal representation, plus extra descriptive information, such as a form's description or a patient's age and gender:
  PIH ID 1234-1 Darius Jazayeri, 29 year old Male
- tooltip -> Would look like the normal-sized version followed by an 'additional info' icon. That icon's tooltip would contain the extra description you'd otherwise see when `size=descriptive`:
- bookkeeping -> Like tooltip, but also includes all bookkeeping information (creator, date-created, modified-by, date-modified).

We should create tag files for standard data entry widgets, under `<openmrs_tag:xxxWidget>`.

- Parameters:
  • `id` & `name` for the form field element
  • `size`: popup | in-place

Create these for patient/user/person, concept, location, encounter type, etc.

SpringMVC

- File naming:
  • `user.list` is the name of the page that lists users
  • `user.form` is the name of the page that lets you edit users
  • `UserFormController.java` and `UserListController.java` are the names of the controllers
  • `UserValidator.java` is the validator

- Messages:
  • `Object.property` = label (e.g. Obs.patient = Select a Patient)
  • `error.property` = error description (e.g. error.gender = Invalid patient gender)
  • `general.property` = general label (e.g. general.description.label = Description)
  • `formname.label` = label (e.g. formentry.title = Form Entry)