Patient Flags Module

Overview

The Patient Flags module allows users to create "flags" that generate warning messages in the Patient Dashboard.

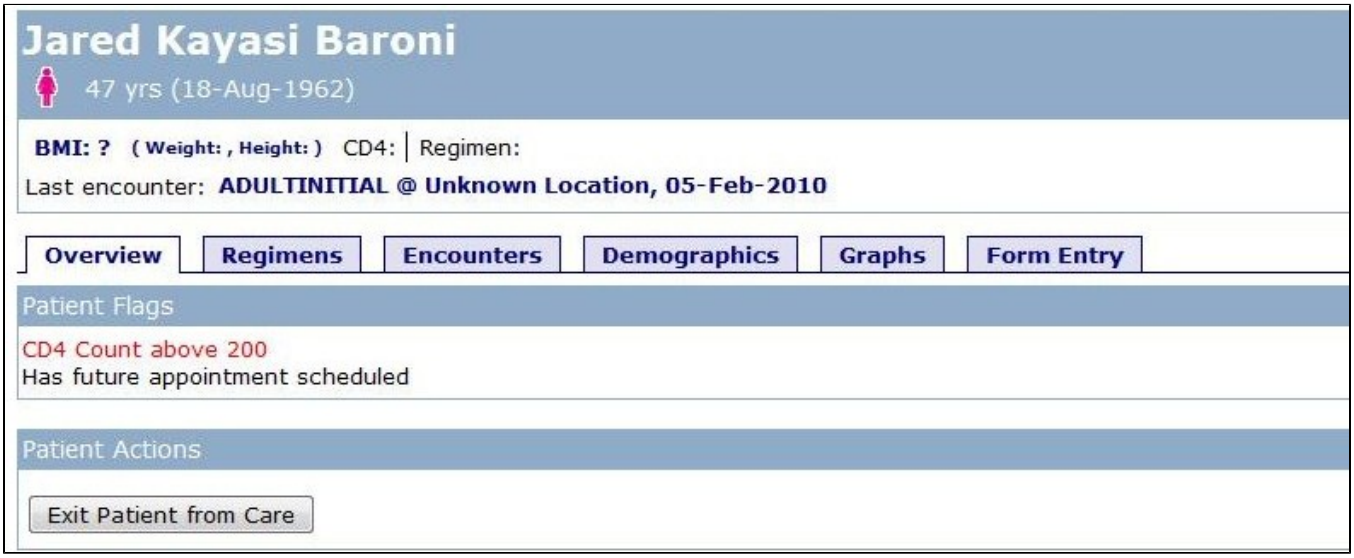
A flag is simply some criteria (stored in a string), and an associated message string. Whenever a Patient Dashboard is loaded, the selected patient is tested against all enabled flags in the system, and for all flags that evaluate true, the associated message string is displayed. The message string can be simple string, or a code for a localized message stored in message.properties.

A flag can also be tested against all Patients in the system via on the Administration page.

Flag messages can be configured to be displayed in the patient header, and/or in a "Patient Flags" box in the Overview section.

An example of a patient flag displayed in the patient header:

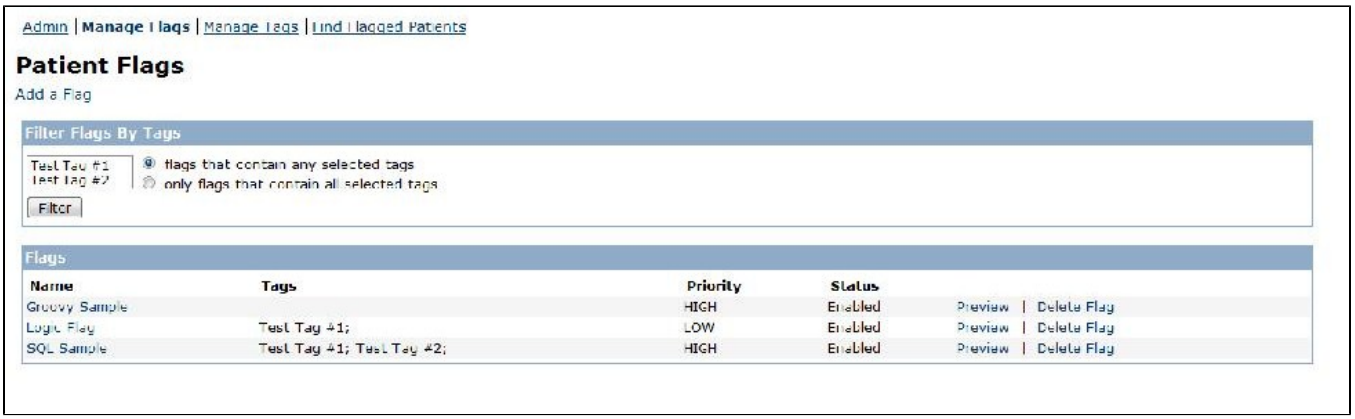
An example of flags displayed in the Overview section:

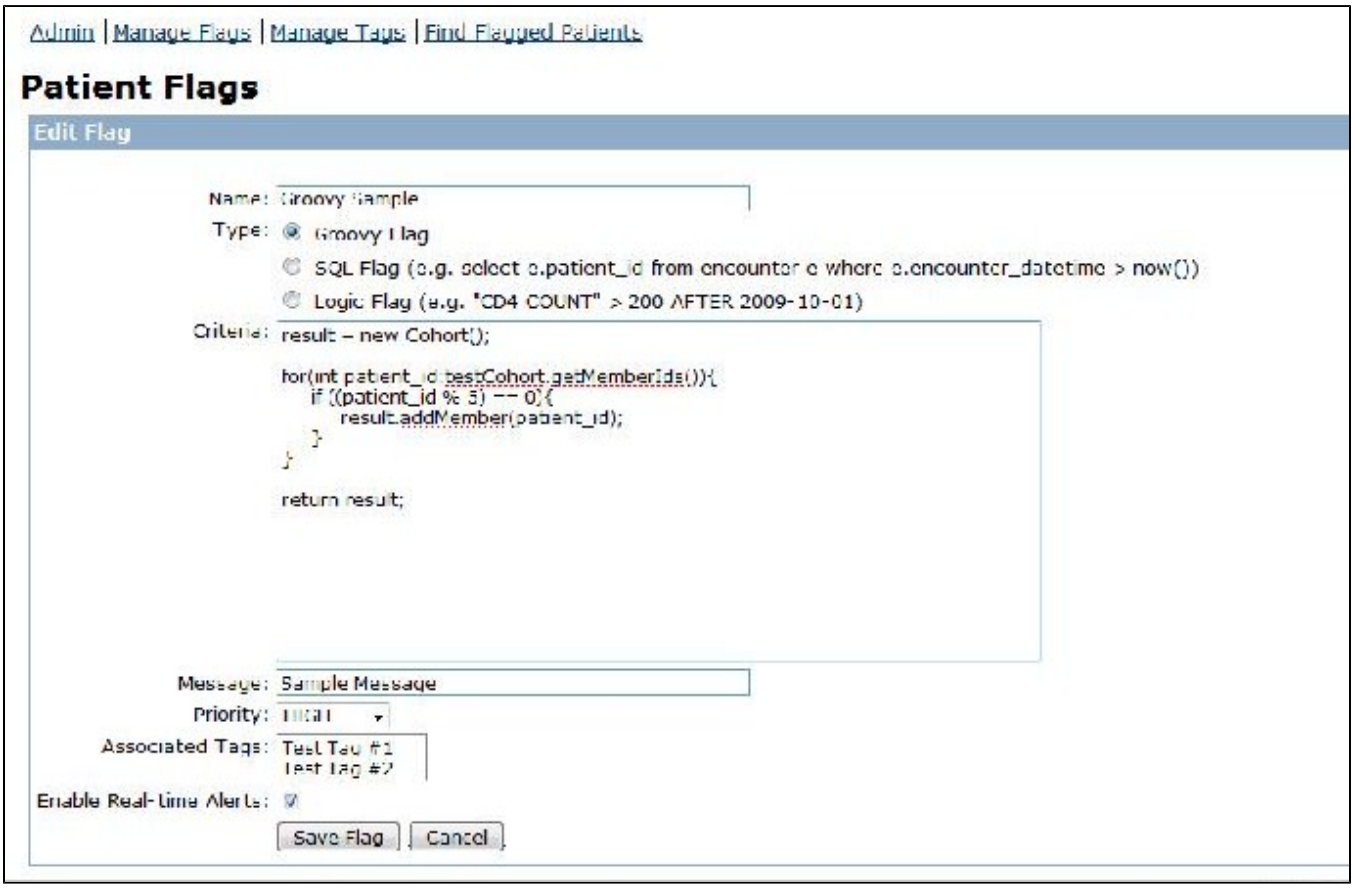


Usage

On the Administration page, the Patient Flags section provides options to Manage Flags, Manage Tags, Manage Priorities, Manage Flag Display, and Find Flagged Patients.

Use the Manage Flags option to create new flags, and to edit and remove existing ones.





When creating or editing a flag, it may be assigned a priority, which determines the order of flags the style of text when flag messages are displayed on the patient dashboard. Priority levels and display styles can be set via the Manage Priorities page. A style for a priority can be any valid attribute that can be used within an HTML tag. For example, to create priority where message text appears in red, set the style property to:

```
style="color:red"
```

Flags can also be associated with one or more tags, which are used to group flags together and to control what specific users (based on role) will see a flag, as well as where the flags are displayed. Use the Manage Tags options to create and edit tags. A tag can be associated with one or more roles and one or more display locations (i.e. the patient dashboard header or patient dashboard overview). Any flag that is associated with a tag will only be displayed in the location(s) associated with the tag, and to a user who has a role associated with the tag.

Note that if a flag has no tags associated with it, it defaults to a "global" flag and is visible to all users in all locations.

The Find Flagged Patients allows one to test a flag, or set of flags, against all Patients in the system.

Flag Types

The module provides for the creation of four types of flags:

Groovy Flag

This flag takes a Groovy script as it's criteria. The script should return a Cohort of all the Patients that should evaluate to "true" for the given flag.

Before execution, the Cohort to test is bound to the variable "testCohort". The shell is configured so that the major OpenMRS services are bound to a variable of the same name, as in the Groovy Module (i.e. the patient service is bound to "patient"). Also, org.openmrs.* is imported by default.

As an example, the following script will flag all female patients in the test Cohort:

```
result = patientSet.getPatientsByCharacteristics("f", null, null);
return org.openmrs.Cohort.intersect(result, testCohort);
```

Note that the evaluator implicitly intersects the results returned by script with the test Cohort so the above script could simply be written as:return patientSet.getPatientsByCharacteristics("f", null, null);

Security issues with Groovy scripts

One potential problem with Groovy flags is that a Groovy script could potentially be destructive. In most cases, we would like to restrict Groovy flags to only have view-related privileges. To that end, a specific username must be associated with Groovy flags. When Groovy flags are evaluated, they are only allowed, by proxy, the set of privileges associated with that user.

As a default, the module sets the Groovy flag username to the username defined in the scheduler.username global property. As this user is most likely a super-user, it is recommended that the Groovy flag username be changed to a user with a more limited set of privileges--perhaps even a dummy "Groovy user" created specifically for this purpose. The Groovy username can be changed in the "Manage Patient Flags Properties" section.

Note that when creating and editing Groovy flags, the privileges allowed are further restricted by those privileges assigned to the current user.

SQL Flag

A SQL flag takes a SQL statement as it's criteria. The SQL should be a SELECT that returns all Patients that match the specific criteria. For example, to flag all Patients that have an upcoming encounter scheduled, the criteria would be:

```
SELECT e.patient_id FROM encounter e WHERE e.encounter_datetime > now()
```

The evaluator implicitly intersects the results returned by the SQL statement with the set of patients being tested.

Note that the SQL statement MUST include "**.patient_id" somewhere within the query (where "**" can be any non-whitespace characters--for instance, the "e.patient" in the above query fulfils this requirement); the evaluator parses out this substring and uses it to append a WHERE (or AND) to the end of the query statement to restrict the query to a single patient when testing against a single patient. For instance, when attempting to test the above criteria against the patient with patient_id 2, the evaluator executes the following criteria: SELECT e.patient_id FROM encounter e WHERE e.encounter_datetime > now() AND e.patient_id=2
Note that when executing the SQL statement, the evaluator sets the selectOnly parameter of the Administration Service's executeSQL method to "true", so that users can't modify the database. Also, while any user should be able to execute a SQL flag, only those users with the SQL Level Access privilege are able to create or modify SQL flags.

Logic Flag

A logic flag takes as it's criteria a logic string compatible with OpenMRS logic. For instance, the criteria **"CD4 COUNT" > 200 AFTER 2009-10-01** will flag all Patients who have a CD4 Count greater than 200 after October 1st, 2009.

Custom Flag

Custom flags can be defined by writing a custom flag evaluator. A custom flag evaluator is any class that implements the FlagEvaluator interface:

```
public interface FlagEvaluator {

        /**
         * Evaluates the given patient against the given flag
         *
         * @param flag the flag to evaluate
         * @param patient the patient to evaluate
         * @return true/false
         */
        public Boolean eval(Flag flag, Patient patient);

        /**
         * Evaluates the given cohort against the given flag
         *
         * @param flag the flag to evaluate
         * @param cohort the cohort to evaluate; evaluators should be implemented so that passing a null
         *            results in testing against all patients in the database
         * @return the subset of patients who evaluate true for the given flag
         */
        public Cohort evalCohort(Flag flag, Cohort cohort);

        /**
         * Validates that the Flag's criteria is valid for this FlagEvaluator
         *
         * @param flag the flag to test
         * @return FlagValidationResult
         */
        public FlagValidationResult validate(Flag flag);
}
  *
```

Display Points

Currently flags can be configured to display in the patient dashboard header and patient dashboard overview. However, it is possible to create new display points in other modules.

First, the display point should be added to the database:

```
FlagService flagService = Context.getService(FlagService.class);


if(flagService.getDisplayPoint("New Display Point") == null){
        flagService.saveDisplayPoint(new DisplayPoint("New Display Point"));
}
```

Then to retrieve a list of flags for a specified patient, user, and display point combination:

```
List results = new ArrayList();
results = flagService.generateFlagsForPatient(patient, Context.getAuthenticatedUser().getAllRoles(),flagService.
getDisplayPoint("New Display Point"));
```

Finally, you can loop through the result set and display the flags as you like. For example:

```
for (Flag flag : results) {
        content = content + "" + flag.getLocalizedMessage() + ", ";
}
```

Global Properties

- *patientflags.defaultPatientLink* - when viewing a list of patients generated using the "Preview" functionality, this defines the link that will be anchored to each patient. The patient id will be appended to end of the link in the format "patientId=123". If no value is specified, it links to the patient dashboard by default.

Required Privileges

- Manage Flags and Manage Tags require "Manage Flags" privilege
- Find Flagged Patients requires "Test Flags" privilege

Download

- Download the Patient Flags module
- View/download the source code from GitHub

REST endpoints

Method	URL	Parameters / Body	Description	Rep Default	Rep Full
Priority					
GET	/ws/rest/v1/patientflags/priority/{uuid or name}		Fetches unique priority by name or uuid	• name • style • rank	• name • style • rank • auditInfo
GET	/ws/rest/v1/patientflags/priority	none	Fetches all priorities		
GET	/ws/rest/v1/patientflags/priority	q	Searches priority by name		
POST	/ws/rest/v1/patientflags/priority	**name, rank** style	Create with properties in request		
POST	/ws/rest/v1/patientflags/priority/{uuid or name}	**name, rank** style	Update with properties in request		
DELETE	/ws/rest/v1/patientflags/priority/{uuid}		Deletes the record		
Flag					
GET	/ws/rest/v1/patientflags/flag/{uuid or name}		Fetches unique flag by uuid or name	• name • criteria • evaluator • message • priority • enabled • tags • resourceVersion	• name • criteria • evaluator • message • priority • enabled • tags • auditInfo • resourceVersion
GET	/ws/rest/v1/patientflags/flag	none	Fetches all flags		
GET	/ws/rest/v1/patientflags/flag	• q (matches starting name), • evaluator (sql | groovy | custom), • enabled (true | false), • tags (comma separated)	Searches data by specified filters.		
POST	/ws/rest/v1/patientflags/flag	**name, criteria, evaluator, message,** priority, enabled, tags (comma separated)	Create with properties in request		
POST	/ws/rest/v1/patientflags/flag/{uuid}	**name, criteria, evaluator, message,** priority, enabled, tags (comma separated)	Update given uuid with properties in request		
DELETE	/ws/rest/v1/patientflags/flag/{uuid}		Delete from database		
Tag					
GET	/ws/rest/v1/patientflags/tag/{uuid or name}		Fetches unique tag by uuid or name	• name • roles • displayPoints	• id • name • roles • displayPoints • auditInfo
GET	/ws/rest/v1/patientflags/tag	none	Fetches all tags		
GET	/ws/rest/v1/patientflags/tag	• q (matches name)	Fetches tag by name		
POST	/ws/rest/v1/patientflags/tag	**name, roles, displayPoints**	Create with properties in request		
POST	/ws/rest/v1/patientflags/tag/{uuid}	**name, roles, displayPoints**	Update given uuid with properties in request		

DELETE	/ws/rest/v1/patientflags/tag/{uuid}?purge		Delete from database		

Display Point

GET	/ws/rest/v1/patientflags/displaypoint	none	Fetches all display points	• name	not supported

- Release Notes
- **Version 1.3.4**

key	summary	status	fixVersion

⚠ Can't show details. Ask your admin to whitelist this Jira URL.

View these issues in Jira

- **Version 1.3.3**

key	summary	status	fixVersion

⚠ Can't show details. Ask your admin to whitelist this Jira URL.

View these issues in Jira

- **Version 1.3.2**
- Reverted change made in 1.3.1 since excluding antlr and asm jars was causing problems with executing Groovy flags
- Added maven plugin to add svn revision number to module version number
- **Version 1.3.1**
- Fixed issue in 1.3.0 where antlr and asm jars were being included in the lib folder of the module (this is redundant since they are being provided by OpenMRS core)
- **Version 1.3.0**
- Updated to use OpenmrsClassLoader when instantiating an evaluator (to facilitate the use of custom evaluators in other modules)
- Mavenized codebase
- **Version 1.2.9**
- Modified implementation so that flags and associated metadata can be shared using the Sync module
- Removed auto-creation of default priorities (to facilitate sync)
- **Version 1.2.8.1**
- Removed "spring-form.tld" from jsp pages to make module compatible with OpenMRS 1.7, since "spring-form.tld" is included in the OpenMRS header as of 1.7
- **Version 1.2.7.1**
- Modified SQL Evaluator so that it will not return voided patients
- **Version 1.2.7**
- Fixed ticket http://dev.openmrs.org/ticket/2264
- **Version 1.2.6**
- Updated to be compatible with OpenMRS 1.6
- **Version 1.2.5**
- Improved Groovy flag evaluator so that privileges given to Groovy scripts can be better customized
- Fixed SQL flag evaluator so that users without SQL Level Access privilege can still execute (but not create or edit) SQL flags
- Added convenience method generateFlagsForPatient(Patient patient, Set<Role> roles, String displayPointName)
- **Version 1.2.1**

- Fixed typo in patientflags_tag_displaypoint table

Version 1.2.0

- Added ability to control flag display based on location and a User's role
- Added ability to specify custom evaluator types
- Misc bug fixes/tweaks

Version 1.1.0

- Added ability to display flags in the patient header
- Added UI for modifying priority levels
- Misc bug fixes/tweaks

Version 1.0.0

- Added priority functionality
- Fixed bugs with SQL evaluator criteria handling

Version 0.9.0

- Initial version with basic functionality

Planned Future Functionality

- Better editor for Groovy scripts
- Improved criteria validation
- UI improvements
- Integration to OpenMRS 2.x UI

Developers

Mark Goodrich