Table of Contents

ISSUE-TRACKER SYSTEM GUIDE

1. INTRO

1.1. Purpose

The purpose of this guide is to provide description of the existing issue-tracker System and it's architecture

1.2. Audience

Any givien instance of the issue-tracker should have ONE and only ONE person which is responsible at the end for the funtioning of THIS instance - so think carefully before attempting to take ownership for an instance. The author(s) of the code are not responsible for the operation, bugs or whatever happens to a new instannce. As a responsible owner of an instance you could create, share and assign issues to the authors of the source code, yet there is no service level agreement, nor even promise to help.

2. ARCHITECTURE

2.1. IOCM architecture definition

The Input-Output Control Model architecture is and application architecture providing the highest possible abstraction for allmost any software artifact, by dividing its components based on their abstract responsibilites, such as Input, Output , Control and Model.

2.1.1. The Control components

The Control components control the control flow in the application. The instantiate the Models and pass them to the Readers , Converters and Writers for output.

2.1.2. The Model components

The model components model the DATA of the application - that is no configuration, nor control flow nor anything else should be contained wihing the model.
Should you encounter data, which is not modelled yet , you should expand the Model and NOT provide differrent data storage and passing techniques elsewhere in the code base ...

2.1.3. The Input Components

The Input Components are generally named as "Readers". Their responsibility is to read the application data into Model(s).

2.1.4. The Output Components

The Output Components are generally named as "Writers" Their responsibility is to write the already processed data from the Models into the output media .

2.1.5. The Converter Components

The Converters apply usually the business logic for converting the input data from the Models into the app specific data back to the Models.

2.2. Multi-instance setup

The multi-instance setup refers to the capability of any installed and setup instance of the issue-tracker application to "know" its version , environment type - developement , testing and production) and owner.

2.2.1. Multi-environment naming convention

Each database used by the issue-tracker application has an <<environment abbreviation>> suffix refering to its environment type. Running application layers against different db versions should be supported as much as possible.

3. BUSINESS LOGIC

3.1. Projects management

You can manage multiple projects with the issue-tracker tool. Each project has its own data directories, database storage and configurations. You could also have different envornments named dev,tst,prd for each project separately.
As the tool is backwards compatible you could have differrent instances of the issue-tracker projects with different versions (and set of features) operatiing against differrent project (each one in its own version).

You must pre-set the configuration variables of an issue-tracker project each time you start working on a project from the shell

```
doParseIniEnvVars /vagrant/csitea/cnf/projects/isg-pub/isg-pub.issue-tracker.doc-pub-host.conf
```

3.2. Increase the date for all projects

to increase the date for all the projects at once use the following oneliner.

```
while read -r f ; do doParseIniEnvVars $f ; bash src/bash/issue-tracker/issue-tracker.sh -a increase-date ; done < <(find doParseIniEnvVars /vagrant/csitea/cnf/projects/issue-tracker/ -type f)
```

3.3. Categories

Each issue item could be categorized under one and only one category. One category might have 1 or more issues.
The categories could contain letters ,numbers, dashes

```
Examples:
organisation-it
organisation-it-operations
```

3.3.1. Issues / Issue items / items

Issue item is the shortest possible description of task , activity , note or anything requiring distinguishable and prerferable measurable action or producing verfifiable outcome.
Issues could be of different types - tasks, activities, notes etc.

```
Examples:
go get the milk
do the homework
procurement e-mail discussion follow-up
```

3.3.2. to search for the project daily file

to search for the project daily file run the following liner first to start the dev server of the react mini-app.
Than point your broser at the following url:
http://doc-pub-host:3307/
(Hardcoded for now ...)

```
bash src/bash/issue-tracker/issue-tracker.sh -a mojo-morbo-start
```
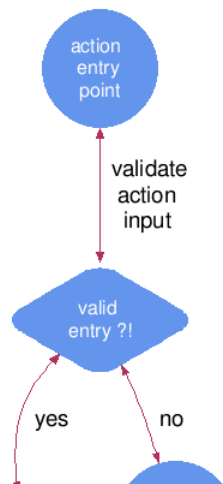
4. APPLICATION CONTROL FLOW

This section provides a generic control flow description for the shell based and ui based control flows.

4.1. Shell control flow

The shell control flow is based on the control model input output architecture.

Figure: 1 issue tracker control flow

dispatcher

stop the action

dispatch correct ctrl func

Controller Ctrl<<Name>>

read the source data

Reader Rdr<<Name>>

read

data read ok ?!

yes   no

Converter Cnr<<Name>>   stop the action

return to controller

Controller Ctrl<<Name>>

write the converted data

Writer Wtr<<Name>>

write to writer media

data write ok ?!

no   yes

stop write error   stop OK

control flow in shell actions' execution