

2009

GOOGLE SUMMER OF CODE PROPOSAL

By Mostafa Muhammad

[WORKFLOW EXTENSION FOR JOOMLA! 1.6]

Workflow systems can become handy to a Joomla! Administrator in certain scenarios, especially those that involve online magazines or community websites where content comes from different sources. I'm proposing to create a simple extensible workflow extension for Joomla! That will support the core content component and have the capacity to support other components.

Table of contents

Table of contents	2
Biography	3
Project goal	3
Definitions and notes	4
General description	5
Structure	5
Views	7
Preliminary Database structure	8
Work scenario	10
Project plan	11

Biography

My name is Mostafa Muhammad, I was born in Cairo on May the 4th , 1985, I'm an Egyptian Muslim, living in Cairo, I'm a final year Medical student at Al-Azhar university in Cairo, My Medical interests are mainly related to Bionanotechnology and Haematology, My interests in the field of computer are 3D graphics (OpenGL mainly) and the related math "Vector math" , My OpenGL implementations are mainly in C++, I'm also interested in Web development particularly PHP which I have been using actively for more than 5 year now, I was introduced to Mambo shortly before the Mambo/Joomla! transition and have been developing for Joomla! ever since, I also served as a moderator of the Arabic forum for a period of time.

I'm a 2008 [GSoC Alumnus](#), My Project was a [WYSIWYG Forms extension](#) for Joomla! 1.5, it is being actively developed, I'll be releasing a new version by the first week of April Insha' Allah

What I would like to learn for Google Summer of Code

Work more closely with the new 1.6 Framework and the advanced ACL functionality.

Personal Skills

I'm honest, friendly and a team player, I tend to look for generic solutions for problems rather than solutions that fit one area, I also am flexible and tend to adapt to different settings.

Project Goal

Provide a simple extensible workflow extension for Joomla! that will support the core content component and have the capacity to support other components.

Definitions and notes

This section contains definitions of some terms that are going to be used throughout this proposal

Workflow:

A workflow is a depiction of a sequence of operations, declared as work of a person, work of a simple or complex mechanism, work of a group of persons.

From design point of view, a workflow is a set of “stations” through which an “Item” passes through in a particular order; Users attach comments/information to the item along the way.

An Item:

The use of the term “item” rather than “Article” is intentional because the extension will be able to integrate with other extensions beside “com_content”, An Item can be anything, A web form, an image in gallery, a file in a file management system provided that these extensions implement a set of triggers to work with this workflow extension and integrate well with Joomla! ACL.

Station:

A station is a unit that makes up a workflow; it is a “stop” the document makes on its way through any given workflow.

A station has a particular position in the workflow, it also has a set of “fields” attached to it (e.g. rating, comment, file attachment), the station fills these “Fields” and can either pass it to the next station or to the previous one.

A station is simply an ACL Group.

Version Control:

When working with workflows content is likely to be edited several times before reaching its final form, which raises the question of “versioning”, This extension will NOT handle version control of content elements that are being workflowed but will have internal support for version control should it be included in future versions of the content management.

This internal support will be provided through “component handler” plugins which allows each component to have its own way of dealing with version control if it supports it, and to ignore it if it doesn’t support it.

General Description

Document workflow will be broken down into a set of "stations", an item "Article or other item within the CMS" will be passed from one station to another in a particular order until reaching its final station after which an arbitrary operation will be preformed "e.g. Publish article".

Structure

Core:

MVC based, the core will act as a central controller for the whole extension, it is composed of two parts:

1. A conventional component
2. Core plugin: A system plugin that captures J! Triggers related to content saving and passes them down to "Component handlers" (See below)

Hook Handlers:

Hooks will be implemented to allow developers to perform arbitrary actions when certain events occur, initially one hook will be implemented, and this hook will be invoked upon transition from one station to the next.

Field Handlers:

They handle the rendering and storage of the "station" fields (e.g. comment fields, file attachments, etc...)

ACL Handlers:

These will interface with different ACL implementations (Core, jACL, etc..)

Component Handlers:

All component specific code will be placed here, these handlers act as a layer between the extension and the component whose content is being workflowed (e.g. com_content),

Each handler is composed of 2 files

1. An INI file that contains Meta data about the handler and a other parameters, among which is the “onSaveItem trigger name” which defines the name of the “J! Event” that is triggered when an item of the handler type is saved. (E.g. onAfterSave in com_content), the file also contains a list of supported “Final Actions”, which are the actions that can be taken when the item reaches its final destination (e.g. publish, delete, archive, etc...)
2. A php file containing implementation of the handler, it must implement the following methods (Subject to change as needed)
 - lockElement (id, aclSystem)
 - unlockElement (id, aclSystem, gid)
 - onNewEntry ()
 - getItemRevision(id, rev)

More on getItemRevision()

If the component in question (e.g. com_content) does support version control, the method returns the requested version, if it doesn't, then the method simply ignores the \$rev parameter and returns whatever stored in the DB.

The above list of methods is a preliminary list and is subject to change as development progresses

When the workflow is saved, the component's onSaveItem trigger is registered with core plugin so that when the event is triggered the core plugin captures it and loads the correct component handler.

Views

Backend views:

List view

Shows all available workflows

Accessibility: Admin level users

Add/Edit Workflow view

Allows the administrator to define stations and custom fields, this view is will be using more JavaScript to enhance usability

Accessibility: Admin level users

Pending Items list view

Shows a list of items that are in a station to which the user belong

Accessibility: Any user with backend access

Pending Item view

Shows the Item and all the attached custom fields (e.g. comments, attached files) and allows the user to fill these custom fields (i.e. add a comment, attach a file).

Accessibility: Any user from station in which the item currently is.

Frontend views:

Pending Items view

Shows a list of items that are in a station to which the user belong

Accessibility: Any registered user

Pending Item view

Shows the Item and all the attached custom fields (e.g. comments, attached files) and allows the user to fill these custom fields (i.e. add a comment, attach a file)

Accessibility: Any user from station in which the item currently is.

Preliminary Database structure

Workflow Table:

Stores data about the Workflow itself

id	Integer	Workflow id
title	Varchar(200)	Title of the workflow
category	Varchar(200)	Component specific string, e.g. (s=1,2,3 c=4) means section 1,2 and 3 and category 4, this field might be used differently by component handlers
component	Varchar(50)	Name of the “component handler”
acl	Varchar(50)	Name of the “acl handler”
admin_gid	Integer	ACL group id that has administrative privileges over this workflow (FK)
jos_content	Multiple fields	Fields like “created, created_by, modified, etc”
like fields		

Stations Table:

Stores data about “Station” definition

id	Integer	Station id
wid	Integer	Workflow id (FK)
title	Varchar(255)	Title of the stations
task	Text	Description of the task to be performed by the station
fields	Varchar(255)	Active fields on this station
group	Integer	ACL Group id (FK)
order	Integer	Position of this step in the parent workflow

Steps Table:

Stores data about the history of steps an item has went through

id	Integer	Step Id
wid	Integer	Workflow id (FK)
sid	Integer	Station id (FK)
iid	Integer	Id of the item that is being processed (e.g. Article id) (FK)
title	Varchar(255)	Title of the Item “e.g. Article” , I could’ve accessed it using the iid as a key but I prefer this approach to reduce the Database and processing load
created	DateTime	When was this Step created
current	Integer	Whether or not this step is the latest jump in this item workflow life
version	Integer	The version of the item that arrived at this station

Field values Table:

Stores field data “comment text, attached file paths, rating, etc...”

id	Integer	Value id
wid	Integer	Workflow id (FK)
sid	Integer	Station id (FK)
iid	Integer	Id of the item that is being processed (e.g. Article id) (FK)
tid	Integer	Step id (FK)
type	Varchar(255)	Type of the field (e.g. “comments”)
created	DateTime	Time when this value was stored
created_by	Integer	Id of the user who created this entry (FK)
modified	DateTime	Time when this value was last modified
modified_by	Integer	Id of the user who last modified this entry (FK)
value	Text	Value of this field “Comment text, attached file path, etc...”

Work scenario

The Administrator wants to create a new workflow for publishing content in a particular set of categories.

The Administrator will use a backend screen to create that workflow; he can specify the categories/sections he wants to be workflowed.

The Administrator defines the “stations” and the custom fields for each station and saves the workflow.

User xyz posted an article, the onAfterSave event is triggered, and the core plugin captures this event and loads the component handler that corresponds to the content type (com_content handler in this case)

The component handler for “com_content” accesses the loaded ACL handler for the current workflow and locks the article for all groups of users and unlocks it for the ACL group of station 2.

When an item reaches its station a hook is invoked; calling all registered handlers, one of these handlers will send an E-mail to station users about the item that waits processing.

Station 2 user logs into J! And accesses the “pending items” view (Accessible via frontend or backend), station 2 fills the custom fields assigned to the station (comment and file attachment) and passes the article up to station 3.

The core locks the article for everybody except station 3, the article proceeds in similar manner until it reaches its final destination where the “final action” will be decided.

Project Plan

Phase I (1 weeks)

Finalize the project specifications

(This implies that the items in the plan below might change)

Phase II (2 weeks)

- Backend workflow list view
- Backend workflow creation view
- Core plugin
- “com_content” component handler

Phase III (2 weeks)

- “Comments” custom field
- “File attachment” custom field
- Frontend task view
- Backend task view

Phase IV (2 weeks)

- ACL handling for “com_content”
- Debugging/Testing