DeClouder Framework

Table of Contents

[Introduction 2](#_Toc26310554)

[Building blocks of the framework 2](#_Toc26310555)

[How it is structured? 3](#_Toc26310556)

[Processors’ Action Flow 3](#_Toc26310557)

[Content Handlers 4](#_Toc26310558)

[Software Project/Package Architecture 5](#_Toc26310559)

[Project - Base ESPoT (Base Components of Enterprise Single Point of Truth): 5](#_Toc26310560)

[Project - Content Handlers: 5](#_Toc26310561)

[Project - OS Handlers: 5](#_Toc26310562)

[Project - Remote Accessors: 5](#_Toc26310563)

[Project - Extended Server Components: 5](#_Toc26310564)

[Project – Extended Handlers: 5](#_Toc26310565)

[Technology stack 6](#_Toc26310566)

[jdk 1.8 6](#_Toc26310567)

[SWT – Standard windows toolkit 6](#_Toc26310568)

[SQLITE 6](#_Toc26310569)

[Google APIs 6](#_Toc26310570)

[Sardine 6](#_Toc26310571)

[Apache POI 6](#_Toc26310572)

[Log4J 6](#_Toc26310573)

[Maven 6](#_Toc26310574)

[izpack 6](#_Toc26310575)

[Pre-loaded Content Types 7](#_Toc26310576)

[How to install? 8](#_Toc26310577)

# Introduction

The Information System architecture underwent radical changes in recent decades from thick clients of client-servers to thin clients of distributed systems, and now toward cloud.

Unfortunately, the farther the business layers moved from the end users, the more painful it had become to perform the day-to-day functions. One has to deal with the network latencies while accessing remote contents, and the difficulty amplifies if one has to process multiple contents to arrive at an aggregated information.

Of course, there had been workarounds such as AJAX to address latencies. But one has to still wade through myriads of unproductive formalities before doing any intelligent decision making while dealing with the data in cloud.

This paper attempts to address two key challenges in information collaboration.

* Assigning single ownership for data elements at enterprise level and rolling up to wider forms with least human effort.
* Spare the authors from unproductive information exchange steps which machines are well apt to handle. An ideal IT system should extend itself to the users for receiving Information instead of asking users to stretch up to its portals.

The DeClouder is an *open-source* framework for effectively defining information at enterprise level and facilitating seamless collaboration. It provides a simple platform for requesting, authoring and publishing information to all the authorized stakeholders. It is designed such a way that one can seamlessly arrive with newer content-types by extending the base framework.

The idea is similar to De-Normalization where an overly *normalized* database is brought *back* to a state where it becomes easier to fetch data for real life scenarios. The DeClouder does the same by bringing the cloud data closer to the users.

It provides a technology-agnostic foundation to integrate with any DocCentral in market and attempts to address the Web 3.0 goal towards distributed computing by splitting the workload amongst Desktop, Catalog server and Extended processing server.

# Building blocks of the framework

1. Doc Central
   * Any 3rd party content-repository. Current version of this framework is designed to support Google Drive, Windows file system and WebDAV enabled storage services such as Sharepoint.
2. Desktop UI
   * Allows requesters to create base templates and assign ownership to specific authors to prepare content.
   * Allows authors to create new content, update and publish.
   * Displays catalog of available contents to subscribe, review and collaborate.
   * Allows authors and requestors to update status and re-assign ownership.
3. Client synch up Orchestrator
   * Submits contents and requests Doc Central to process it.
   * Downloads refreshed catalogs and the subscribed content from Doc Central.
4. Server orchestrator
   * Processes/aggregates contents received from users and extended processors.
   * Publishes catalog via Sqlite file with the location details of content in Doc Central
5. Extended orchestrators
   * Transforms contents as per each business context.

## How it is structured?

The building blocks are designed with Abstract classes which the users can expand to cater unique needs of their organization.

The three processors i.e. client, server and extended Orchestrators make the magic happen by leveraging four sqlite database files.

* sysdbfile – provides the characteristics of each content type viz. which handler shall be picked to process the content, whether they are of individual type or roll-ups that provide a grouped view, any extended handling involved etc. The updates on sysdbfile such as when a new content type is introduced by the architects it gets automatically propagated to all users by the framework.
* catalogdbfile – provides the available contents at the Doc Central, their location, author etc. in the form of an ERL master record i.e. Enterprise Resource Locator. This is refreshed and re-published by server orchestrator every time the contents are updated, and the clients pick them from a pre-defined location.
* clientdbfile – holds the details of the local drafts that the user creates or reviews, details of subscriptions to the ERL contents and the status of their local availability etc. This stays only at client desktops.
* extendedcatalogdbfiles – data related to any extended processing for contents of specific types. This stays only at extended servers.

## Processors’ Action Flow

The processors continuously monitor for updates and take follow up actions specific to each content type.

The client orchestrator reads the latest catalog from the Doc Central and keep the local repositories current so the user can review and update. Also when the user creates a new draft and declares ready for upload, the client orchestrator pushes the *content* into the Doc Central’s *content drop box*. It also writes a *request file* into the *request drop box* with details of the corresponding content file.

The server orchestrator reads the request files and moves the corresponding contents into their destination folders *as is* or aggregates to its rollup content based on parent or child type. After processing all such requests, it publishes the renewed catalog in the publication folder.

The extended orchestrator mimics a human’s effort for any enrichment such as combining multiple spreadsheets to provide a summary view, persisting time cards or creating a dashboard for a project.

## Content Handlers

While the processors can handle generic base functions, they have no clue on the specific needs of the contents. For that they load and trigger the corresponding *content handlers* at run time.

The framework deploys a lightweight dependency injection function to load content handlers during run time via dynamic class loading feature of Java.

A content can be either an individual type which is maintained as is, or a rollup type which would go sit inside another rolled up content.

The content handlers handle the unique requirements its content type for displaying at client side, processing uploaded contents at catalog server side and for any extended processing.

# Software Project/Package Architecture

## Project - Base ESPoT (Base Components of Enterprise Single Point of Truth):

* Contains the base components to handle the Desktop and Doc Central processes.
* Common routines that simplifies to work of business layer
* Abstracts of content handlers
* Interface definitions to standardize Remote processing and OS handling
* Dynamic loaders of content handlers, OS handlers and Remote processing implementations.
* Refreshers to synch up newer versions of content handlers without manual intervention.
* ReviewHandlers through which the users can log remarks directly on the artifacts

## Project - Content Handlers:

* Provides the handling mechanism for specific content type at client side as well as server side.

## Project - OS Handlers:

* File handling specific OS. Currently available only for Windows OS.

## Project - Remote Accessors:

* Enables to interact with different document collaboration tools viz. Windows file system, Google Drives, WebDAV enabled portals in a consistent manner.

## Project - Extended Server Components:

* Provides the base for Extended processing on special contents that require unique enrichments.

## Project – Extended Handlers:

* Implementations of the extended processing of special contents that require unique enrichments at server side.

**CommonOpenCldFns and XtdCommonOpenCldFns**

* Packages that cater commonly used methods at client/servers and at the extended servers respectively.

# Technology stack

## jdk 1.8

Java

## SWT – Standard windows toolkit

A lightweight framework for desktop UI

## SQLITE

A serverless self-contained database engine used to track the drafts one creates, to publish the available catalog of artifacts to all participants, to track the subscriptions at subscriber’s desktop and to hold the content type configurations

## Google APIs

For accessing a Google Drive based doc central

## Sardine

For accessing WebDAV enabled doc central

## Apache POI

For rolling up excel artifacts

## Log4J

Error Logging

## Maven

For version control of external repositories.

## izpack

For packaging and installation

# Pre-loaded Content Types

**Idea Generator**

Enables team members to submit ideas and rolls them into ideas repositories which others members can review and provide feedback.

**Timesheet Capture**

Simplifies time capturing for team members. Allows one to create a new time-capture function to track the time spent at specific interval and start tracking the time expended nearly real time. In future version of this too, this module will evolve to interact through mobile devices.

**ToDo Generator**

Team leaders can create a template artifact and request and another member to author for completion. The todo items for one gets rolled up into a list-view to make it easier to handle.

**Simple Tracker**

Enables members of a team to maintain consistent data sheets which can be merged onto a DeckerLite rollup sheet.

**Decker Lite**

A simple decking process that combines all users' individual contents into a group level view.

**Project Tracker**

Enables project managers with an All-in-One Dashboard that tracks effort spent against budget, defects, impediments and schedule performance of individual tasks - with a limitation of not auto-capturing impacts from inter-dependency. The project performance analysis will still require a Planning tool such as MS Projects since few essential aspects viz. interdependencies and resource calendars are not covered by this handler.

**Decker Grouper**

Decks up individual contents into summary view. E.g. one can setup a summarizer of all project trackers to provide a high-level view to the leadership.

# How to install?

First choose the infra platforms for the catalog server, extended servers and the platform server where your own customized components would reside.

Until you customize a new content type for your own users, you don’t have to set up a platform server. But you need to suppress the periodic refresh via the flag suppressSysCompRefresh in commons.properties.

Arrive with the Relevance tree structure and authorized users listing.

Then set up the servers and finally the desktop users using the installation package.

For the installer package and the DeClouder tool to execute,

* set the java class path where the JRE is located.

The installation package is downloadable from GitHub and it does the following:

* Captures the Installation folder path, Desktop User name who will be logging on the machine, Application User name and also the proxy IP and port details.
* Set the property files - Common, Client, System, Server and Extendedserver to point to the user’s choices.
* Store the executables into the folder and base contents mapped into user’s own desktop folder (i.e. c:\users\Vibeesh).

Once the installer completes,

* Open the config folder and set the properties files of Common, Client, Server, Extendedserver etc. to point to user’s settings.
* Map the Orchestrators to trigger via the OS schedulers OR trigger on need basis.

Configure the relevance structure appropriate to the team in the catalog master database. Add authorized users into the Users table.

At the root server side, ensure to provide the right access for the users to read content folders and catalog publish folders and the request response folder. Also provide update access to the request drop box and content drop box folders only to the registered authors and add their details in the users table of catalog master.