CollabTracker

Table of Contents

[Introduction 3](#_Toc40914739)

[Building blocks of the framework 3](#_Toc40914740)

[How it is structured? 4](#_Toc40914741)

[Processors’ Action Flow 4](#_Toc40914742)

[Content Handlers 5](#_Toc40914743)

[Software Project/Package Architecture 6](#_Toc40914744)

[Project - ColbTrk (Base Components): 6](#_Toc40914745)

[Project - Content Handlers: 6](#_Toc40914746)

[Project - OS Handlers: 6](#_Toc40914747)

[Project - Remote Accessors: 6](#_Toc40914748)

[Project - Extended Server Components: 6](#_Toc40914749)

[Project – Extended Handlers: 6](#_Toc40914750)

[Technology stack 7](#_Toc40914751)

[jdk 1.8 7](#_Toc40914752)

[SWT – Standard windows toolkit 7](#_Toc40914753)

[SQLITE 7](#_Toc40914754)

[Google APIs 7](#_Toc40914755)

[Sardine 7](#_Toc40914756)

[Apache POI 7](#_Toc40914757)

[Log4J 7](#_Toc40914758)

[Maven 7](#_Toc40914759)

[izpack 7](#_Toc40914760)

[Pre-loaded Content Types 8](#_Toc40914761)

[How to install? 9](#_Toc40914762)

[ADMIN’s initial activities: 9](#_Toc40914763)

[User desktop side triggering: 10](#_Toc40914764)

# 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 tool 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 CollabTracker 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.

Its based on the DeClouding principle which 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 CollabTracker attempts to bring 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 user’s desktop, doc centrals, server processors and extended processing servers.

# 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 syncs up the local repositories so the user to review and update seemlessly. Also when the user creates a new draft and declares it to be 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 handle generic house-keeping 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 - ColbTrk (Base Components):

* 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

**General Request Generator**

Acts as a general-purpose requests collation tool. Helps to categorize and take actions.

**Idea Generator**

Captures ideas from team members and rolls them the group buckets. Captures feedback from others and helps to collaborate.

**Timesheet Capture**

Captures timecards in real time. Allows one to create a new time-capture function to track the time spent at specific interval and start tracking the time expended in real time. This module will further evolve to interact with mobile users to capture the time card content.

**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 a All-in-One Dashboard that tracks effort, defects, impediments and minimal view of independent task schedule progress in live. Limitation: doesn’t 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?

* Choose the infra platforms for the catalog server, extended servers and the platform server where your own customized components would reside.
* Unless you intend to create 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.
* At the machines where the application component execute, set the java class path where the JRE is located.
* Set up the servers and then the desktop users using the installation package. The installation package is downloadable from GitHub and it does the following:
  + Captures the Installation folder path, Desktop User name who will be using the application and also the proxy IP and port details.
  + Sets the property files - Common, Client, System, Server and Extendedserver to point to the user’s choices.
  + Stores the executables into the folder and base contents mapped into user’s own desktop folder (i.e. c:\users\Vibeesh).

## ADMIN’s initial activities:

* At the root server side replicate the model folders as in DemoRoot and provide access to users as below:
  + 1\_allmembersreadable: contains artifacts, catalog publications and requests’ responses folders. All users shall be provided read access to this folder.
  + 2\_contributorswritable: contains request drop box and content drop box. Provide write access on the request drop box to all contributors to place their requests. Create subfolders within content drop box with the names of the contributors and provide write access to the corresponding personnel to place their contents.
    - <INSTALL\_PATH>\WindowsRoots\DemoWinContentRoot\2\_contributorswritable \contentdropbox\<ApplicationUserName>
    - e.g. C:\Kannan\Java\ColbTrk\WindowsRoots\DemoWinContentRoot\2\_contributorswritable\contentdropbox\DEMOUSER
  + 3\_behindscene (contain housekeeping folders).
* Configure the relevance structure appropriate to the team in the catalog master database. Add contributing users with ADMIN login using the user-maintenance content type. Few common purpose user IDs are factory set e.g. ADMIN, DEMOUSER, XTDSTDPROC, XTDTMSHPROC, XTDDECKERLITE, XTDDECKERGRPR.
* Assign a server processing machine to execute the Server Orchestrator to perform housekeeping operations on the doc central contents.
  + Install the CollabTracker on this machine and configure the trigger mechanism to initiate the Server Orchestrator.
  + Copy the model catalogMasterDb file from catalogMasterDbFileOf<DemoWinContentRoot> into the new root specific file.
  + Update the tables Relevance and Users in the catalogDb as per need.
* Admin shall run the server orchestrator first so the catalog publication gets initiated and all other users can initiate sync ups before starting to contribute.
* Assign as many extended machines to execute the Extended Server Orchestrators to handle extended functions of the doc central contents.
  + Install the CollabTracker on this machine and configure the trigger mechanism to initiate the Extended Orchestrators.
  + Copy the folder extdSrvrDeckrLite with its content i.e. its own clientdb file.
  + Ensure the placement of the extended catalog db file such as <catalogXtdTmCapture>DbFileOf<DemoGShContentRoot> in extendedcatalogdbfiles folder.
* Optionally, further customization can be done through updating *properties* files placed in the config folder viz. Common, Client, Server, ExtendedServer etc.

## User desktop side triggering:

* Once the admin completes the server side set up, users can invoke the client UI and view published contents via catalog display.
* The client processor needs to be triggered so the user side contents are in sync with server side.