Chukwa Console Integration Guide

- Introduction
- What is Chukwa Console (HICC)?
- Install Chukwa Console
- User Guide
- Customization
    - View Permission
    - Add Your Own Widget
        - Example 1. Simple Hello World
        - Example 2. Hello Your Name
    - More Detail About The Examples
    - More Examples
    - Charting Widget Component
    - Target Widget Component

Introduction

This document provides a high level overview of the Chukwa Console (aka HICC). It describes how to setup the Chukwa HICC console and integrate the Chukwa console as a standalone web portal. It is targeted to developers who just want to use the Chukwa web UI component to display their own data set.

For information about setting up the full Chukwa system (instead of just HICC), please read Chukwa_Quick_Start.

What is Chukwa Console (HICC)?

HICC is a java based portal platform for service management. The basic features for HICC include:

- Individual user view management
- Basic Drag and Drop Web portal functions
    - Multiple views per user
    - Multiple tabs page per view
    - Multiple widgets per tab page
    - Drag and Drop relayout individual view
- Basic view permission system
- external widgets integration
    - data integration. Using HICC existing UI component to display chart, table and graph.
    - direct UI integration. Component can provide a URL and HICC will display the html page as a widget on the portal page

Install Chukwa Console

Chukwa HICC UI is a Java web application running on top of the Tomcat application server. Here is the steps required to get it up and running.

- Make sure you have mysql installed and running. These directions assume that a user root exists, with no password.
- Building the HICC war file (hicc.war).
    - Check out the Chukwa source code from Hadoop distribution. (You can get the source code as part of the Hadoop Apache SVN repository here)
    - Make sure you have Java JDK and Ant installed in your system. For detail installation instruction, please read the readme.txt file in the distribution. To build the HICC component alone, do the following
        - cd chukwa/trunk
        - ant
    - After the build, you will find the **hicc.war** in the chukwa/trunk/build/hicc.war.
- Installation
    - Install tomcat on your server.
        - Download the latest tomcat
        - unzip the tomcat package
    - copy hicc.war to the **tomcat/webapps** directory
    - start tomcat.
        - tomcat/bin/catalina.sh start
    - setup configuration
        - make a directory for chukwa data and chukwa configuration. You can put it anywhere you want. For this example, I will put them in ~/
            - mkdir ~/chukwa
            - mkdir ~/chukwa/conf
            - echo "test=jdbc:mysql://localhost/test?user=root" > ~/chukwa/conf/jdbc.conf
            - export CHUKWA_CONF_DIR=~/chukwa/conf
            - mkdir ~/chukwa/data
            - mkdir ~/chukwa/data/descriptors
            - cp tomcat/webapps/hicc/descriptors/* ~/chukwa/data/descriptors
            - mkdir ~/chukwa/data/views
            - cp tomcat/webapps/hicc/views/* ~/chukwa/data/views
            - export CHUKWA_DATA_DIR=~/chukwa/data
    - restart the tomcat.
        - tomcat/bin/catalina.sh stop

        - tomcat/bin/catalina.sh start
    - access the web UI
        - goto: http://<host>:8080/hicc

User Guide

The user's views are store in your $CHUKWA_DATA_DIR/views directory. Each view is stored under <name>.view file as a JSON object.

To create a new view, do the following:

- In the UI, click **Workspace Builder**
- Click **Create New Workspace**. It will create a new workspace called **New View**
- You can click on **Change** button on the same row to change the view name. It will change the file in the $CHUKWA_DATA/DIR/views directory to the new view name.

To add more widgets to the view,

- Click **Options** -> **Add Widget**
- Explorer the widget tree and add widgets into the potral.

To remove widget from the view.

- move the mouse to the widget's title bar and click the "X" delete button
- after confirm the deletion, the widget will be removed from the page
- To save the modification, click "Save Dashboard".

Customization

In this part of the documnent, I will describe how to customize the HICC console for your own application.

View Permission

(** TODO **)

This will describe the HICC user based permission system.

Add Your Own Widget

Example 1. Simple Hello World

First, we will build a **Hello World widget**.

- cd $CHUKWA_DATA_DIR/descriptors
- create a hello_world.descriptor file. The name of the file is not important as long as it is unique.
- Inside the hello_world.descriptor file, put the following (remember, the content is in JSON format. Make sure that it is a valid JSON string).

```
{
"id":"hellow_word",
"title":"Hello Word",
"version":"1.0",
"categories":"Test,My Widget",
"module":"iframe/jsp/hello.jsp",
"description":"Hello world",
"screendump":"\/images\/start.png",
"refresh":"15",
"parameters":[
]
}
```

- create tomcat/webapps/hicc/jsp/hello.jsp and put "Hello world" in the file.
- remove the cache files (if they exist)
    - remove $CONFIG_DATA_DIR/descriptors/*.cache
    - remove $CONFIG_DATA_DIR/views/*.cache
- Congralution! You just finish your first HICC Widget.

You can now go to the HICC UI. Click on **Options** -> **Add Widget**. You will see the widget tree has a new category calls "Test". Open it up and you will see your **Hello World** widget. Click **Add**. Then it will add the widget to your view and it will display **Hello world** as in the widget area.

Example 2. Hello Your Name

In this example, we will extend example 1 to take configuration parameter.

- cd $CHUKWA_DATA_DIR/descriptors
- create a hello_world.descriptor file.
- Inside the hello_world.descriptor file, put the following:

```
{
"id":"hellow_word",
"title":"Hello Word",
"version":"2.0",
"categories":"Test,My Widget",
"module":"iframe/jsp/hello.jsp",
"description":"Hello world",
"screendump":"\/images\/start.png",
"refresh":"15",
"parameters":[
{"name":"your name","type":"string","value":"","edit":"1"},
]
}
```

- create tomcat/webapps/hicc/jsp/hello.jsp and put

```
 Hello <% out.print(request.getParameter("your name")); %>.
```

  in the file.
- remove the cache files (if they exist)
  - remove $CONFIG_DATA_DIR/descriptors/*.cache
  - remove $CONFIG_DATA_DIR/views/*.cache
- Congralution! You just finish your second HICC Widget.

You can now go to the HICC UI (* *** refresh the UI so it will load the new descriptor file ****). Click on **Options** -> **Add Widget**. You will see the widget tree has a new category calls "Test". Open it up and you will see your **Hello World** widget. Click **Add**. Then it will add the widget to your view and it will display **Hello** as in widget area. (If you already has the hello widget add in the page from example 1, you can remove it and readd it to your page).

Now, the fun part, in the widget, move your mouse to the widget's title bar. You will see the "edit" option. Click on it and you will see the widget edit option. There is an option calls "your name". Type in your name as "joe" and click "save". The widget will refresh and display "Hello joe". HICC will pass the configuration parameter to the jsp as a request parameter. Hence, the JSP page will be able to get the parameter and display it.

More Detail About The Examples

The HICC UI framework consists of 3 parts.

- The framework and view
  - The framework will handle the view management, user management, javascript level drag and drop. It will also handle descriptor version control and widget loading.
- The descriptor
  - You can think of the descriptor file as a Java class file, PHP class definition or any OO class definition. It provides the id (similar to class name), description, refresh time in second and per widget instance parameters (similar to class's internal variables). When the view loads up a widget, it will pass the parameters to individual URL specified in the descriptor file.
- Your widget
  - Just think of it as a web page. The descriptor file will pass the configured parameter to your URL and you can use the parameter to display the web page.

Inside individual descriptor file, you need to specify the following JSON fields:

Descriptor File Format

Field ID	Decsription	Comment
id	a unique id for the widget	
title	The name/title of the widget. It will be displayed in the widget selection UI.	
categories	A comma separated list. It is used to specified the components category in the widget selection tree.	
module	The called back URL for the widget. The content for the widget to be displayed.	
description	description of the widget. The description is display in the widget selection UI.	
screendump	(optional) A small thumbnail of what the widget look like.	
version	version number of the widget. The format is: MajorVersion.MinorVersion. HICC will use the value to handle widget upgrade	

refresh	the default refresh rate for the widget in minute. For example, 15, means the widget will be refresh in 15 minutes.	
parameters	List of the configuration parameters for the widget (see the configuration parameter table below for detail).	

Configuration Parameter Table

Type	Description	Example	Comment		
Edit Field (String)	Single edit box	{"name":"table","type":"string","value":"ClientTrace","edit":"0"}			
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="255e6b87-06d1-4000-b7d0-62192830511b"><ac:plain-text-body><![CDATA[	Drop Down Single Selection	A drop box to allow user to do single item selection	{"name":"height", "type":"select", "value":"200", "label":"Height","options":[{"label":"200","value":"200"},{"label":"400","value":"400"}, {"label":"600","value":"600"},{"label":"800","value":"800"},{"label":"1000","value":"1000"}]}		]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="ca85eee0-cf04-415f-876d-2229764d7921"><ac:plain-text-body><![CDATA[	Multiple Selection	A list control which allow multiple selection.	{"name":"fruits", "type":"select_multiple", "value":"apple", "label":"fruits you like", "options":[{"label":"Apple","value":"apple"}, {"label":"Orange","value":"orange"}, {"label":"Banana","value":"banana"},]}		]]></ac:plain-text-body></ac:structured-macro>
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="fceccd53-d66b-448f-9ef8-f78d217ab6e8"><ac:plain-text-body><![CDATA[	Radio Box	A radio button control	{"name":"legend","type":"radio","value":"on", "label":"Show Legends", "options":[{"label":"On","value":"on"},{"label":"Off","value":"off"}]}		]]></ac:plain-text-body></ac:structured-macro>

More Examples

(*** TODO ***)

Charting Widget Component

(*** TODO *** Need to provide more detail and/or more example for this)

In order to use the chart component, you need to create a dataMap variable and put the graph data information inside it. Then call the Chart component to draw the chart.

A good example is in: **src/web/hicc/jsp/single-series-chart-javascript.jsp**. Inside the jsp file, it first opens the database and read the data. Then it will call the Chart component and return the chart to the front end.

You can also package your data from rest API or database, then call the Chart as below and return the object to the front end. (TODO: need to link to the Javadoc of the Chart component.)

```
        if(dataMap.size()!=0) {
            if(request.getParameter("render")!=null) {
                render=xf.getParameter("render");
            }
            Chart c = new Chart(request);
            c.setYAxisLabels(false);
            if(request.getParameter("x_label")!=null && xf.getParameter("x_label").equals("on")) {
                c.setXAxisLabels(true);
            } else {
                c.setXAxisLabels(false);
            }
            c.setYAxisLabel("");
            if(request.getParameter("x_axis_label")!=null) {
                c.setXAxisLabel(xf.getParameter("x_axis_label"));
            } else {
                c.setXAxisLabel("Time");
            }
            if(title!=null) {
                c.setTitle(title);
            } else {
                c.setTitle(metrics.toString());
            }
            if(request.getParameter("y_axis_max")!=null) {
                double max = Double.parseDouble(xf.getParameter("y_axis_max"));
                c.setYMax(max);
            }
            if(request.getParameter("legend")!=null && xf.getParameter("legend").equals("off")) {
                c.setLegend(false);
            }
            c.setGraphType(graphType);
            c.setXLabelsRange(labels);
            c.setSize(width,height);
            c.setDataSet(render,dataMap);
            out.println(c.plot());
         }
```

Target Widget Component

(*** TODO ***)