Operations Console

The Operations Console is a web application that allows you to create, monitor, and review your workflows in a web browser. You can view workflows as well as pause, resume, remove, interrupt, expedite and even edit them.

The Operations Console is known to perform significantly slower in Internet Explorer than in other browsers. We recommend using one of the other supported browsers (listed in the technical specifications) where possible.

Minimum Requirements

The Operations Console has the same minimum requirements as the Flux engine.

Installing the Operations Console

There are two ways to install and run the Operations Console: using the built-in Jetty 6 container, or using another third-party container (Tomcat, WebSphere, GlassFish, etc.)

You can launch the Operations Console using the built-in Jetty 6 container by running the *start-opsconsole* script from your Flux installation directory. This will start the Jetty container on port 7186, where you can access the Operations Console. This means that once you've launched the Operations Console, you can access it by visiting this URL in your browser:

```
http://<hostname>:7186
```

Where <hostname> is the host name of the machine where the Operations Console was started and 7186 is the default Operations Console port.

If you need the Operations Console to use a different port, just edit the *start.ini* file (found in the root of the Flux folder) and change the start and stop ports (example provided below):

```
--module=deploy
--module=http
--module=logging
--module=websocket
jetty.port=7189
stop.port=7190
stop.key=secret
```

As shown, to access the Operations Console after the change, you'd need to access <localhost>:7189 instead of using the default port.

To install the Operations Console in a separate container, you must first create the *flux.war* file. You can do this by running the *make-war* script in your Flux installation directory (this script is automatically run in the *configure* script as well, so if you have already run that script, there is no need to also run *ma ke-war*).

Once the *flux.war* file has been created, you will be able to find it in the *webapp* directory of your Flux installation. To install the Operations Console, just deploy this file using the preferred technique for your container (more information for deploying WAR files should be included in the documentation for your container).

Connecting to Engines

When you connect to an engine in the Operations Console, Flux will store the information about that engine's location for future use. This information is stored in a file called *.opsconsole8.properties*, which is located under the *.flux* director beneath the home directory of the user who started Flux (on Unix-based systems, this will be something like */Users/myuser/.flux/opsconsole8.properties*, and on Windows, *C:\Documents and Settings\.flux\opsconsole8.properties*).

You can use the add engine button on the Operations Console home page to modify this information. You can also directly edit the *.opsconsole8.properties* file - this is most useful in cases where you just want to copy the engine information to a different machine with a new installation of the Operations Console.

The Operations Console and Engine Clusters

The Operations Console may only connect to one engine cluster at a time. If you add a new engine in the Console, be sure that it is part of the same cluster as all of the engines already connected to the Console, or you will likely experience problems viewing workflows and performing operations.

A separate Operations Console instance must be run for each engine cluster that you want to connect to.

The .flux Folder and opsconsole8.properties

The first time you use the Operations Console, it will create a new hidden folder called *.flux*. This folder will be stored in the home directory of the user who starts the Operations Console.

Within the folder, you'll find a file called *opsconsole8.properties*. This file is used to store all of the engines that the operations console has connected to. As you work with the Console, this file will start to become populated with entries; a typical *opsconsole8.properties* might look like:

```
#Mon Dec 24 16:45:47 MST 2012
engine1=
engine1.host=localhost
engine1.port=7520
engine1.preferencelevel=0
engine1.ssl=true
engine2=
engine2.host=localhost
engine2.port=7521
engine2.preferencelevel=0
engine2.ssl=true

#These last 2 lines must be added to access the logviewer
#The first line below specifies the log directory, the second specifies the log extension
logDir=c:/flux-8-0-x/log
logFilter=*.log
```

As you'll notice, the file contains a few lines for each engine that the Console is connected to. The settings for each engine are denoted by the keyword "engine", followed by an incrementing index, then the '.' character (so for the first engine, all entries are marked "engine1", "engine2" for the second, and so on). A property name for one of the engine's settings will follow, followed by the '=' character and, finally, the value of the property.

These are the properties that the Operations Console uses to look up the engine. The properties are used only for these lookups and do not affect any execution or configuration for the engine itself.

The purpose of each line is as follows:

1. Each engine begins with a blank property setting, such as "engine1=". This blank property indicates to the Console that a new engine's properties are starting.
2. **host** – the machine name or IP address of the machine where the Flux engine is running.
3. **port** – the port number that the Flux engine is bound to.
4. **preferencelevel** – when multiple engines are listed, you can use this property to specify the order in which the Console will attempt to contact engines. A lower number means the Console will try that engine first. This allows you to ensure that if some engines might have network connectivity issues, the Console can always contact the most reliable engine first.
5. **ssl** – indicates if communication with this engine is secured using SSL.
6. **logDir, logFilter** – This points to the log directory and the log file extension in your Flux installation. Only needed if you'd like to see the log files in the browser accessing the Log Viewer (http://localhost:7186/logviewer.html).

Properties 1-5 are then repeated for each connected engine; 6 is only set once per installation.

Valid Connection Settings between Flux Cockpit, Operations Console, and Engine

After changing any of the following settings, restart the Flux engine and opsconsole services.

Setting	Engine Configuration	opsconsole8.properties for the opsconsole webapp	fluxConfig.js for Flux Cockpit
Valid Connection States			
Disable browser access	server=false	N/A - Operations Console disabled	N/A - Cockpit disabled
Secured opsconsole webapp access	server=true and security_enabled=true	ssl=true	ssl=true
Unsecured opsconsole webapp access	server=true and security_enabled=false	ssl=false	ssl=false
Invalid Connection Settings			
Secured opsconsole webapp access	server=true and security_enabled=true	ssl=false (Will fail to connect)	ssl=false (Will fail to connect)
Unsecured opsconsole webapp access	server=true and security_enabled=false	ssl=true (Will fail to connect)	ssl=true (Will fail to connect)

Viewing the DB entries in the Operations Console

Flux 8.0.11 embeds a database viewer for querying the Flux database tables from Operations Console. It can be accessed from this URL: http://localhost:7186/console. It is provided as tool for troubleshooting only.

Performing database updates to Flux database tables without consulting Flux support can lead to inconsistency in workflows. Please refer to our Database Viewer section for all the details.

Running Multiple Operations Console Instances on a Single Server

Each Operations Console instance is tied to the account of the user on the Operating System who starts the Operations Console process. Because of this, Flux only supports one Operations Console instance at a time per OS user.

In order to start multiple Console instances on the same machine, Flux requires that each Operations Console process is started using a different user account on the OS.

Custom JARs and Classes

If your workflows contain user-defined persistent variables, custom triggers, or custom actions, you do **not** need to deploy the corresponding class files to the Flux Operations Console. Any custom JARs, dependencies, or classes required by workflows must be available on the class path of all engines in the cluster, but they do not need to be available on the class path of the Flux Operations Console.

Operations Console and Performance

To conserve system resources and ensure the highest performance possible, it is recommended to periodically shut down the browser window and reopen it when monitoring an engine using the Operations Console.

As long as the Operations Console process itself continues running, closing the browser window will have effect on the connection to the engine or on the data of the engine or Operations Console. Restarting the browser periodically allows resources to be freed up and refreshed to maintain optimal running conditions.

It is also not necessary to keep the browser window open to receive updates on the status or execution of your workflows. When the browser is restarted, it is able to automatically obtain the latest information from the Operations Console server.

Operations Console and Time Zones

Since the Operations Console picks up the Time Zone information from the browser, you must ensure that the system running the Flux engine and Operations console have the same Time Zone settings as the computers being used to access the Operations Console.

Having different Time Zones might lead to unexpected behavior in the Operations Console including the Forecast of scheduled workflows not being shown properly, logs not being shown in the Logs tab, etc.

Operations Console Slowness or Flashing

Settings in the configuration can cause slowness in the load and refresh of the Flux user interface. Consider the following:

- If there are engines or agents defined in the opsconsole8.properties file, the operations console can take a long time to load since it will try repeatedly to connect to each of these engines and agents. Make sure your opsconsole8.properties file is current, or simply delete it and restart the Flux engine and operations console. The file will be recreated.
- If your engine is unsecured - make sure SSL is turned off in the engine-config.properties file, in the opsconsole8.properties file, and starting with Flux 8.0.13, the fluxConfig.js file.
- If at startup the Flux Operations Console (not Flux Cockpit) starts flashing and blinking, alerting the 'All engines are unreachable', delete the opsconsole8.properties file and restart the Flux engine and operations console. The file will be recreated. This can occur if SSL is switched from off to on and the opconsole8.properties file (and starting in Flux 8.0.13 - fluxConfig.js) do not match the SSL settings of the engine.

Setting the Session Timeout

A Session timeout represents the event occurring when a user does not perform any action within Flux cockpit during an interval (defined by web.xml) and instruct Flux to destroy the session (deleting all unsaved in-progress work).

By default, the session timeout is defined in the web.xml for 1440 minutes (24 hours). The value is defined within the session-config section of the web xml, located in the webapps directory of the Flux installation.

```
<session-config>
  <!-- Session timeout = 24 Hours -->
 <session-timeout>1440</session-timeout>
</session-config>
```

You can change this value and restart the Operations Console for a new timeout value.

Setting a Default Search Filter in the Legacy Operations Console

There may be instances where you want to set a default search filter when initially accessing the workflow view on the Operations Console. In such instances the Javascript module that needs to be changed to support a default search filter is located in Module.js in the webapp/javascript directory. Insert the following lines

```
// Initialize the search filter
if (!thisModule.gridFilter.filterApplied && document.getElementById("namespaceFilter").value == "Namespace") {
 queryParams["namespaceFilter"] = "*021*";
 document.getElementById("namespaceFilter").value = "*021*";
}
// End the initialize
```

between the lines:
 var queryParams = [];
 if (flux.lang.isValue(thisModule.gridFilter)) {
Into the code within Module.js below.

```
    getInitialRequestUrl: function (oState, oSelf) {
      var thisModule = flux.lang.isValue(oSelf) ? oSelf : this;

      // Get states or use defaults
      oState = oState || { pagination: null, sortedBy: null };
      var sort = (oState.sortedBy) ? oState.sortedBy.key : "timestampFormatted";
      var dir = (oState.sortedBy && oState.sortedBy.dir === YAHOO.widget.DataTable.CLASS_ASC) ? "asc" : "desc";
      var paginatorParams = thisModule.getPaginatorParams(oState);
      paginatorParams["sort"] = sort;
      paginatorParams["dir"] = dir;
      var queryParams = [];
      // Insert the code above to perform the initialize of the search filter here!

      if (flux.lang.isValue(thisModule.gridFilter)) {
        try {
          if (thisModule.gridFilter.filterApplied) {
            queryParams = thisModule.gridFilter.getFilterParams(true);
          }
        } catch (e) {
          // Do nothing
        }
      }
```

Changing the default per user would look as follows:

```
// Initialize the search by user
if (!thisModule.gridFilter.filterApplied && document.getElementById("namespaceFilter").value == "Namespace") {
  if (flux.session.username.toLowerCase() == "admin" ) {
    queryParams["namespaceFilter"] = "*021*";
    document.getElementById("namespaceFilter").value = "*021*";
  }
}
// End the initialize
```