Runtime Configuration and Runtime Variables

Types of Configuration

There are two configuration components for the Flux engine:

1. Static. Static configuration options are described in greater detail in the Engine Configuration chapter.
2. Runtime. Runtime configuration options typically determine how workflows run (for example, limiting the number of workflows that can run at once) or contain data that the workflows will use while running. Unlike the static configuration, the runtime configuration is dynamically reloaded, so runtime configuration settings can change while the engine is running. Runtime configuration options are applied to nodes, or branches, on the workflow namespace tree, allowing you to define runtime configuration properties on a per-namespace level.

> **Runtime Variables Are Important!**
>
> Runtime variables are a key feature of Flux. They allow your workflow designers to separate the specific values a workflow requires from the workflow itself. In this manner you can separate variables such as server names or file locations from the workflow itself. It is common for Flux users to have different runtime configuration files for different environments, and place these under version control. For instance, production, QA, and developer instances of Flux may all utilize the same workflows, but with different runtime configuration files.

Every engine is controlled by both its static (non-runtime) configuration and its runtime configuration.

The runtime configuration is optional. To enable the runtime configuration, you must first specify the configuration file location, then set the properties you would like to use.

Setting the Runtime Configuration File

To use a runtime configuration file, you can simply add the path to it in the following property to your engine configuration (*nix based systems):

```
RUNTIME_CONFIGURATION_FILE=/path/to/runtime-config.properties
```

For Windows systems, use either of the following syntaxes:

```
RUNTIME_CONFIGURATION_FILE=C:/runtime-config.properties Or RUNTIME_CONFIGURATION_FILE=C:\\runtime-config.properties
```

> Make sure that your runtime configuration file DOES NOT contain duplicate variable names or empty variables.

Refreshing the Runtime Configuration File

Flux can automatically and dynamically pick up runtime configuration changes while the engine is running, without requiring an engine restart. You can adjust how often Flux refreshes the configuration file by setting the RUNTIME_CONFIGURATION_FILE_REFRESH_FREQUENCY like so:

```
RUNTIME_CONFIGURATION_FILE_REFRESH_FREQUENCY=+30s
```

This property is a time expression. The default value is +m (every one minute). This means that once a minute a refresh test of the runtime configuration is initiated. If the runtime configuration properties file has been modified between the last time the refresh test was run and the current time, the refresh of the runtime configuration properties file will be performed. So - for example - if the refresh test is performed at the top of every hour, and the runtime configuration properties file was modified a few seconds before the top of the hour - then the runtime configuration will be refreshed.

Setting Properties in the Runtime Configuration

The runtime configuration file contains a list of key / value pairs. Each line of the file will contain one key / value pair, where the key is a workflow namespace plus a runtime configuration property or variable name, and the value is the intended value of the property or variable to be set.

- **Runtime Configuration Properties**: Configure how workflows are executed on the engine. These include concurrency throttle settings, priority values, FIFO settings, and more. See the List of Runtime Configuration Properties below for a complete list of available properties.
- **Runtime Variables**. Custom variables that provide data to your workflows. Runtime variables let you reuse the same data across multiple workflows (a username or password, for example) while providing a single location for managing that data. You can use variable substitution or runtime data mapping to access the runtime variable values inside a workflow.

Runtime configuration properties are set like so:

```
/namespace/<PROPERTY TYPE>=<value>
```

Setting Variables in the Runtime Configuration

Runtime variables behave like runtime properties. Runtime variables are set like:

```
/namespace/<VARIABLE NAME>=<value>
```

Variables can span lines to make reading them easier.

Spanning multiple lines for clarity

```
/namespace/<VARIABLE NAME>=some text \
some more text \
final bit of text
```

Variables can also span lines and contain line breaks. This is useful for variable strings that may be output as console messages:

Spanning multiple lines and adding line breaks

```
/namespace/<VARIABLE NAME>=some text \n \
some more text \n \
final bit of text
```

And for writing Prescripts and Postscripts that can be reused throughout your workflows. Note that Flux does not perform runtime substitution on a script that is loaded via runtime substitution.

Using a variable to reference a script

```
/namespace/<VARIABLE NAME>=print("Here is some text to output."); \n \
print("Here is some more text to output."); \n \
print("Done outputting text");
```

Note that you may need to escape certain characters if they are being used within your script. Once common character is the backslash "\".

Escaping characters

```
formContents.append("\""); \
        Needs to be escaped as follows:
formContents.append("\\""); \
```

Common Uses of Runtime Variables

Runtime variables are used to define commonly used values within a Flux environment. Server names and user names are two such values that are frequently defined in runtime variables. Since runtime variables are looked up in the tree of configuration properties (see next paragraph), may organizations define their development, QA, and production server names within runtime configuration variables.

For example, the following server name of a customer named Acme, but each environment refers to a different server, In this way the same workflow can be used in different environments without having to individually change the workflows running in each environment. If no namespace is defined on a workflow, the ACME host will default to the development host.

Runtime Variables for Separate Servers

```
/ACME-HOST=acme.server.dev
/PROD/ACME-HOST=acme.server.prod
/QA/ACME-HOST=acme.server.testing
/DEV/ACME-HOST=acme.server.development
```

Tree of Configuration Properties

The runtime configuration is set up as a tree, mimicking the workflow namespace tree. A property set on a branch in the runtime configuration tree is applied to the branch *of the same name* in the workflow namespace tree.

Runtime configuration namespaces are not case-sensitive. For example, the namespace "/MyNamespace" is equivalent to setting "/mynamespace" in the runtime configuration.

Runtime configuration and variable properties can be set on any namespace in the tree, including the root ("/"). Runtime settings applied to a namespace also apply to all child namespaces, and settings applied on the root apply to all workflows across the engine.

Properties set on a child namespace will override any properties set on a parent. For example, if a workflow runs in the namespace "/parent/child/", it will first check the namespace "/parent/child/" for any runtime properties. Properties for the "/parent" namespace would only be applied if they were not set in "/parent/child".

A sample runtime configuration might look like:

```
/MAIL_SERVER_VARIABLE=mymailserver.com
/CONCURRENCY_THROTTLE=8
/namespace/CONCURRENCY_THROTTLE=5
```

Runtime configuration settings can be applied to a specific workflow, a workflow branch, or any workflows matching a specified pattern.

For example, consider an engine that has the following workflows available:

```
/lightweight/MyLightWorkflow1
/lightweight/MyLightWorkflow2
/lightweight/DailyWorkflow1
/heavyweight/MyHeavyWorkflow1
/heavyweight/MyHeavyWorkflow2
```

To specify a runtime variable that applies only to a single workflow, you can specify the full workflow namespace in the runtime configuration, like so:

```
/lightweight/MyLightWorkflow1/MAIL_SERVER=mymailserver.com
```

To specify a runtime variable that applies to an entire namespace, you can include only the namespace branch, like:

```
/lightweight/MAIL_SERVER=mymailserver.com
```

Finally, to specify a pattern, include the portion of the namespace the engine should match. The runtime configuration setting below would apply to the workflows "/lightweight/MyLightWorkflow1" and "/lightweight/MyLightWorkflow2", but not to "/lightweight/DailyWorkflow1":

```
/lightweight/MyLightWorkflow/MAIL_SERVER=mymailserver.com
```

More generally, the namespace portion of the runtime setting is always applied as though the engine is performing a wildcard search.If you'd like to view all workflows on the engine, add a wildcard ("*") character to the namespace parameter and apply it to a filter in the Operations Console (in the case of the example above – using the search pattern "/lightweight/MyLightWorkflow*" would reveal all workflows that the runtime setting would apply to).

Runtime Configuration Properties for Namespaces that Contain a Space

If your namespace contains a space character, like:

```
/Folder A/MyWorkflow
```

You will need to make sure the space is escaped in your runtime configuration by adding a backslash ('\') character before the space in the configuration. For example, to set a runtime configuration property for a workflow with the name and namespace above, you could add the following line to your runtime configuration file:

```
/Folder\ A/CONCURRENCY_THROTTLE=0
```

List of Runtime Configuration Properties

The following properties can be set in the runtime configuration. All of the properties listed can be specified across the entire engine (by setting them on the root namespace), or assigned to an individual namespace.

Property Name	Description
CONCURRENCY_THROTTLE	The total number of workflows that can run concurrently within this namespace. For example, a concurrency throttle of 5 would mean that 5 workflows in this namespace should be allowed to run at once. Note that there is one exception to this: if a workflow contains a split (or, in general, any concurrently executing actions), each action will count toward the total concurrency. For example, if a workflow contains 5 actions running at once, and the namespace has a concurrency throttle of 5, that workflow will use all of the available concurrency slots and no other workflows will be allowed to run in the same namespace until the actions complete. The concurrency throttle setting allows you to conserve limited resources (like processing power and I/O bandwidth) by preventing an unbounded number of workflows from running at once. You can also set a concurrency throttle of 0 to prevent a certain namespace from running at all. For example, if you have workflows that require specific resources available on the machine, you could use concurrency throttles to limit which machines those workflows are allowed to run on. ⚠ **VERY IMPORTANT** The presence of a runtime configuration will cause Flux to ignore the Engine configuration property CONCURRENCY_LEVEL. If no CONCURRENCY_THROTTLE is set for the root namespace ('/') then '/CONCURRENCY_THROTTLE' defaults to 10. If you set concurrencies in the runtime configuration make sure you also add a CONCURRENCY_THROTTLE for the root namespace in the runtime configuration file for best practices.
CONCURRENCY_THROTTLE_CLUSTER_WIDE	Identical to the CONCURRENCY_THROTTLE setting, except that this property specified the total number of workflows in this namespace that can be allowed to run across the entire cluster. For example, if engines A and B are running in a cluster, the CONCURRENCY_THROTTLE_CLUSTER_WIDE for each engine is set to 8, engine A is running 5 workflows, and engine B is running 3 workflows, then neither engine will accept any new workflows until the concurrency slots are available again. This property overrides the individual concurrency throttle for each engine – an engine will not accept a workflow if the cluster wide throttle for that workflow's namespace is full, even if there is room in the engine's CONCURRENCY_THROTTLE settings for that namespace.
DEFAULT_FLOW_CHART_ERROR_HANDLER	The error handler that workflows in this namespace should use. This property should be set as a path to the error handler file. The path can be absolute, like "/path/to/myfile.ffc", or it can be relative to the Java home directory (that is, the directly where the JVM was started, usually the directory containing the script to launch the Flux engine). A relative path may look like "folder/myfile.ffc".

FIFO_SCHEDULING_ENABLED	Adds first-in-first-out (FIFO) scheduling in addition to prioritization and "next execution date" in the workflow execution queue. When enabled, this ensures that workflows will execute in the order they were originally submitted to the engine (though workflow priorities are still respected and always take precedence over FIFO scheduling – see order of execution for a more detailed explanation). For example, suppose 10,000 trucking manifests need to be generated. If the engine's concurrency throttle is 10, the first 10 workflows returned by the database are executed, then the next 10, and so on – with no strict ordering enforced by the Flux engine. Now suppose that the manifests must be generated in order, so that the first manifest submitted is the first generated, and the 10,000 th is the last generated. Enabling FIFO scheduling will ensure that the first 10 workflows submitted are executed first, then the next 10 – continuing in this fashion until all 10,000 are generated. Note that in the case above, workflows are still executed in "groups" of 10, filling the concurrency throttle – to achieve a true queue-like behavior, set the concurrency throttle to one. This ensures that one workflow must either run to completion or reach a trigger before the next flow can execute. Paused workflows are not considered in the FIFO queuing algorithm. ⚠ FIFO scheduling is a system-wide property – therefore, it must be enabled at the root namespace ("/") like so: /FIFO_SCHEDULING_ENABLED=TRUE The configuration setting will not be applied if specified at a lower namespace. FIFO scheduling is disabled by default.
INTERNAL_LOGGER_DEBUG_ENABLED	If this property is set to true, action and trigger properties and variables for workflows in this namespace will be logged in greater detail to the log file and audit trail. As the name implies, this property can only be used if the internal logger is enabled.
LISTENER_CLASSPATH	Specifies the listener class path for workflows in this namespace. This property specifies the listener classpath for workflows, which is used to load listener classes in Java Actions and Dynamic Java Actions.
PRIORITY	Specifies the priority for workflows in this namespace. Priorities are used to give precedence to certain workflows when there are more workflows available to run than concurrency slots open to run them. ⓘ **Note Regarding Priority** When a workflow is submitted to the engine, • it will be assigned the priority specified in the workflow itself, or if none is specified, • it will be assigned the runtime priority, and if that is not specified, • then it will be assigned the default priority of 10. The workflow will run once at the priority specified in the workflow itself, but if a runtime priority is specified for the workflow, all subsequent firings are governed by the runtime priority if one is specified, otherwise it will be assigned the default priority of 10 for future firings.
RUN_AS_USER	The username of a user whose permissions will be used to run this workflow. This can only be used when using the built-in Flux security mechanism.

Memory Usage

It is assumed that your entire runtime configuration tree can be stored in memory. A very large runtime configuration can cause memory problems in Flux, so take care to ensure that you are not setting any unnecessary or extraneous data in the tree.

Storing Encrypted Passwords in the Runtime Configuration

If you have a password value that is used in several workflows across the engine, it often makes sense to store the password in one central location where all of the workflows can easily access it.

In this case, you would normally not want to store the password in plain text in the runtime configuration either (where any user with access to the runtime configuration file or the logs might access it). To help with this, Flux allows you to store an encrypted value in the runtime configuration, protecting your password values and ensuring that they are not available in plain text anywhere on the system.

To store an encrypted password in the runtime configuration and access it within your workflows, follow these steps:

1. Follow the Command Line Interface for generating an encrypted password. This will create an encrypted password which should look something like *Gc4f0oRkPJ4=*.
2. Edit your runtime configuration to store the encrypted password. The encrypted password doesn't require any special syntax and can be set like any other variable:

   ```
   /MyEncryptedPassword=Gc4f0oRkPJ4=
   ```

3. Use runtime substitution to access the value in the password field on your action:

   ```
   ${runtime MyEncryptedPassword}
   ```

That's it! This technique can be used for any action in Flux that requires a password, including file actions and triggers that use a password for authentication to a remote host.

The Runtime Configuration File and Java API

When using the runtime configuration file, it is automatically refreshed from the file system to pick up any changes that might be made. For this reason, it is important that you do *not* make any changes to the runtime configuration using the API if you are using a runtime configuration file (the engine has no way of knowing whether the file or API changes are the "correct" version, so the automatic file refresh assumes that it is correct and will overwrite any changes you have made in the API).

If you are using the runtime configuration file, any changes to the runtime configuration should be performed in the file alone. If you are using the API, the runtime configuration should be managed entirely that way, avoiding the runtime configuration file altogether.