Clean Workflows

Implementation and User Guide

Kevin McMillen

Orchestra Networks Professional Services

1 Overview 3

2 Setup 4

2.1 Create properties file 4

2.1.1 Constructors 4

2.1.2 JVM parameters 4

2.1.3 Format 5

2.2 Declare service(s) 6

3 Execution 6

4 Design 7

4.1 Config 7

4.1.1 CleanWorkflowsConfig 7

4.2 Config Factory 7

4.2.1 CleanWorkflowsConfigFactory 7

4.2.2 DefaultCleanWorkflowsConfigFactory 7

4.3 Property File Helper 7

4.3.1 PropertyFileHelper 7

4.3.2 CleanWorkflowsPropertyFileHelper 8

4.3.3 CleanWorkflowsPropertyConstants 8

4.4 Service 8

4.4.1 CleanWorkflowsUserService 8

4.4.2 CleanWorkflowsImpl 8

4.4.3 CleanWorkflowsServiceDeclaraion 8

4.4.4 CleanWorkflowsService (deprecated) 8

# Overview

The Clean Workflows service can be used to clean up either active or completed workflows and their associated resources.

In a production environment, you may wish to use this service when:

* There are many completed workflows which are no longer needed, and performance would be improved by cleaning up some of them.
* Major changes were made to workflow models or the code that they reference that aren’t backwards compatible, and all of the old active workflows are now invalid.
* Users haven’t been diligent in completing their workflows, and there are active workflows from a long time ago that are no longer needed. Perhaps all work was supposed to be completed by a particular date, so anything before that date is considered invalid.

In a non-production environment, you may wish to use this service when:

* You’ve been developing and testing for weeks and now you have a lot of workflows that you no longer need.
* You want to convert an integration environment into a user test environment and the existing workflows are no longer valid.
* You want to copy your local workspace and give it to someone else, but want to clean it up beforehand.
* You want to duplicate a production database to create a new test environment that closely reflects production, but you don’t want the workflows from production.

To accomplish this in EBX can be a painstaking process involving selecting just the workflows you want to remove and selecting “Terminate and clean this workflow” from the “Actions” menu. This is further complicated by the fact that you can’t terminate a subworkflow, so you must make sure to only select the master workflows.

Also, a common pattern is to have a workflow associated with a particular master data space and a child data space of that master. Workflow models can be configured to automatically close a child data space upon termination, but not all workflow models are configured in this manner. (In particular, previous versions of EBX didn’t have this feature.) After terminating and cleaning the workflow, those child data spaces will still be open. You have to find them and close them one by one in the Data Spaces tab. Alternatively, you could find and close multiple data spaces at once from the Administration tab, but that also presents difficulty narrowing down which data spaces you wish to close.

There may be other customer project specific activities that need to happen upon cleanup as well.

Orchestra Networks Professional Services developed the Clean Workflows service to provide an easier alternative to this process. It allows you to terminate and clean all workflows of a particular publication based on certain date ranges, and potentially close all child data spaces associated with these workflows. In addition, it can be extended to support additional cleanup activities.

# Setup

## Create properties file

The Clean Workflows service utilizes a properties file to tell it what to clean. Each instance of the service is tied to one properties file, but you can have multiple instances of the service, each tied to a different properties file. By default, the Clean Workflows service will look for a file called clean-workflows.properties in your EBX Home directory. If you have just one Clean Workflows service, you can keep that default. Otherwise, you will need to override this behavior. Even if you have just one service, you still may wish to load the properties file from a different location or give it a different name.

To override this behavior, use the techniques described below.

### Constructors

The location of the properties file(s) can be specified in the constructor of the Clean Workflows service. The Clean Workflows service is itself created by a Service Declaration. (If unfamiliar with the EBX user service framework, see the EBX documentation.) To utilize a constructor other than the default, you can create your own Service Declaration or extend the Clean Workflows Service Declaration and override the createUserService method. Whichever approach you take, here are examples of instantiating the service with its various constructors:

* new CleanWorkflowsService()  
  specifies to use the default file called clean-workflows.properties in the default folder, which is EBX Home, unless otherwise specified by a JVM parameter (as described below).
* new CleanWorkflowsService(“../my-module/properties”, “my-clean-workflows.properties”)  
  specifies to use a file called my-clean-workflows.properties in the folder ../my-module/properties.
* new CleanWorkflowsService(“my-clean-workflows.properties”)  
  specifies to use a file called my-clean-workflows.properties in the default folder, which is EBX Home, unless specified by a JVM parameter (as described below).

Note that there is also a constructor that takes in a Clean Workflows Config Factory as a parameter. In that way, you could construct it to be configured without a properties file, using some other mechanism entirely.

### JVM parameters

The location of the properties file(s) can be specified by JVM parameters upon startup. These will only be used if not explicitly set by the constructor, as described above. For example:

* **clean.workflows.properties.folder**: The folder to find the properties file(s) in.
* **clean.workflows.properties.file**: The properties file to use. This should only be specified if you have one Clean Workflows service. Otherwise, you’ll need to specify multiple files so you’ll need to do that via the constructor (as described above).

For example:

* -Dclean.workflows.properties.folder=../my-module/properties  
  –Dclean.workflows.properties.file=my-clean-workflows.properties  
  specifies to look for a file called my-clean-workflows.properties in the folder ../my-module/properties.
* -Dclean.workflows.properties.folder=../my-module/properties  
  specifies to look for the default file called clean-workflows.properties in the folder ../my-module/properties, unless the file is specified by the constructor (as described above).

### Format

The properties file should look like this:

*Whether to use the working data space of the workflow when determining child data spaces to close. If true, this assumes there is a workflow parameter called workingDataSpace. If the workflows are specified to close data spaces upon terminating, or if they don’t use child data spaces, then this isn’t necessary. If there is also a parameter called masterDataSpace, it won’t close it if those values are equal.*

useWorkingDataSpace = true

*This is the list of workflow publications whose workflows are being terminated and cleaned. You can specify “all” to indicate every workflow in the environment.*

workflowPublications = \

CreateProduct, \

EditProduct

*Whether to include completed workflows when cleaning.*

includeCompleted = true

*Whether to include active workflows when cleaning.*

includeActive = true

*The date before which all workflows should be cleaned.*

createdBeforeDate = 2018-05-01T11:59:59

*The number of days of workflows to keep. Everything prior to this should be cleaned.*

createdBeforeNumOfDays = 730

The following properties mostly exist for backwards compatibility with an earlier version of the Clean Workflows service, but can still be used if desired:

*This is the data space that is the parent of the child data spaces being closed. If useWorkingDataSpace is specified, or if the workflows are specified to close data spaces upon termination, then this isn’t necessary. If this is specified, the service will close ALL child data spaces of this master data space, regardless of if they are involved with workflows or what dates are specified.*

masterDataSpace = ProductMasterDataSpace

*This is a list of child data spaces to skip, if there are static children that you don’t want closed. All data spaces specified must be children of the master data space specified above. It can be left blank if not applicable.*

childDataSpacesToSkip = \  
 CreateProductWorkflowPermissions, \  
 EditProductWorkflowPermissions

## Declare service(s)

The service is declared in the same manner as all data space user services. See the EBX documentation on user services for more information. Essentially, the module registration should declare it. If using the standard PS Module Registration classes, a Clean Workflows service will be declared automatically, configured to use a single clean-workflows.properties file. If something else is desired, it can be extended to remove that and declare different Clean Workflows services, or a different module registration class can be used.

# Execution

Select the Clean Workflows service from any data space that it is configured for. By default, it is configured for all data spaces. This will execute the service and display the form. The fields on the form are initialized with the values from the properties file as defaults.

After submitting the form, the service will notify you when it is complete. The service could take a while to finish if there are a lot of workflows to clean up.

If both a Created Before Date and a Created Before Num Of Days is specified, the service will use whichever date is earliest. Note that the service only considers the status and date of the master workflows. If the master meets the criteria then it will be cleaned, or if the master doesn't meet the criteria it won't, regardless of if the subworkflow is active or completed or what date the subworkflow was launched.

This service does not delete the closed data spaces. You are still responsible for creating deletion requests for whichever data spaces you wish to delete, and executing the purge.

This service also does not unpublish old workflow publications. There is currently no API to allow this.

Be very careful when executing this service because it is impossible to restore the workflows once they are cleaned. You should back up your repository before executing it for the first time in case the configuration isn’t correct, or else execute it on a test environment with workflows that you don’t care about preserving.

# Design

It is not important to know the design of the Clean Workflows service in order to use it. However, if you wish to extend or modify its behavior, the details below should prove useful, in conjunction with the javadoc.

All classes are in package com.orchestranetworks.ps.admin.cleanworkflows, unless otherwise noted.

## Config

The Clean Workflows service uses a configuration (“Config”) class to specify its behavior. The Config mostly consists of attributes with getter and setter methods.

### CleanWorkflowsConfig

This is a class that contains the attributes used by the service.

## Config Factory

Config objects can be constructed using factories. It is not required to use a factory, but it is the easiest way to construct a Config.

### CleanWorkflowsConfigFactory

This is an interface that specifies a method for creating a Config from an EBX repository and an EBX session.

### DefaultCleanWorkflowsConfigFactory

This is a default implementation of CleanWorkflowsConfigFactory. It knows how to initialize attributes of a Config based on properties from a properties file, using a PropertyFileHelper.

## Property File Helper

Helper classes are used to read from properties files and populate values of Config attributes.

### PropertyFileHelper

This is a class that contains methods for reading properties from a properties file. It knows how to look up EBX components, such as data spaces and workflow publications, based on properties that conform to a particular format. This class is outside of the cleanworkflows package in com.orchestranetworks.ps.admin because it can be used for other services and isn’t tied directly to the Clean Workflows service.

### CleanWorkflowsPropertyFileHelper

This is a subclass of PropertyFileHelper that knows how to initialize attributes of a Config based properties from a properties file. It also specifies the default name and location of the properties file.

### CleanWorkflowsPropertyConstants

This is a class that contains static constants defining the property names used by the Clean Workflows service.

## Service

The core functionality of the Clean Workflows service is defined in a user service (conforming to the EBX user service framework).

### CleanWorkflowsUserService

This is the class that defines the Clean Workflows service itself. It creates the Config, displays the form, and on submit, calls the CleanWorkflowsImpl class. After completion, it notifies the user that it is complete and redirects the screen back to where the user was before launching the service.

### CleanWorkflowsImpl

This contains the core functionality of the service. It terminates and cleans the workflows and can be extended to perform other activities as well.

### CleanWorkflowsServiceDeclaraion

This declares the service on data spaces. It should be referenced by a module registration class. (The standard PS Module Registration references it.)

### CleanWorkflowsService (deprecated)

This is deprecated. It is used in older versions of EBX that didn’t have the user service framework. It must be invoked by a JSP and the JSP must be referenced by a data model. It doesn’t display a form, but rather takes the values from the properties file as the final values, rather than defaults. It then calls the Impl class, in the same manner that the CleanWorkflowsUserService does.