******



**Integration Framework  
User Guide**

**1.1**

**Demandware Inc.**

**Purpose of this document**

The purpose of this document is to outline a general concept which describes the workflow, configuration and logging view of the workflows as well as the workflow components.

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comment** |
| Version 0.1 | 07/25/2012 | Dietrich Thie/Danny Gehl/Oz Kokler | Initial version |
| Version 0.2 | 07/31/2012 | Oz Kokler | Structural changes and adjustments |
| Version 0.3 | 08/03/2012 | Oz Kokler | Structural changes and adjustments |
| Version 0.4 | 01/15/2013 | Kay Domachowski | Add description for file logging capabilities |
| Version 1.1 | 06/20/2013 | Holger Nestmann | Added new standard components |
|  |  |  |  |

Table of contents

[Table of contents 3](#_Toc339876392)

[Motivation 4](#_Toc339876393)

[Cartridge Installation 4](#_Toc339876394)

[Configuration 5](#_Toc339876395)

[General concept 5](#_Toc339876396)

[Scheduled Job Configuration 7](#_Toc339876397)

[Monitoring 7](#_Toc339876398)

[List of out of the box components 8](#_Toc339876399)

[Data model and related infrastructure 11](#_Toc339876400)

[Schedule execution 13](#_Toc339876401)

[Asynchronous components 14](#_Toc339876402)

[Detailed-List of components 14](#_Toc339876403)

[Catalog export component 14](#_Toc339876404)

[Catalog import component 15](#_Toc339876405)

[Inventory import component 15](#_Toc339876406)

[Order export component 16](#_Toc339876407)

[Price import 16](#_Toc339876408)

[Price-Book export component 17](#_Toc339876409)

[Import slots component 17](#_Toc339876410)

[Import content component 17](#_Toc339876411)

[Time condition component 18](#_Toc339876412)

[Date condition component 18](#_Toc339876413)

[Date-Time condition component 19](#_Toc339876414)

[Workflow clean-up component 19](#_Toc339876415)

[Clean-up files component 19](#_Toc339876416)

[Transfer to FTP component 20](#_Toc339876417)

[Transfer from FTP component 20](#_Toc339876418)

[Rebuild indexes component 21](#_Toc339876419)

[Custom components development 22](#_Toc339876420)

# Motivation

The growing complexity of the batch processes required to operate a shop, have brought in some cases the existing implementation to its limits. On the one hand side because of the growing number of batch components (Jobs) and on the other hand due to lacking monitoring and notification capabilities.

This concept will define a general way how workflows and its components need to be built so that a common infrastructure can provide facilities for monitoring, logging, configuration, scheduling etc.

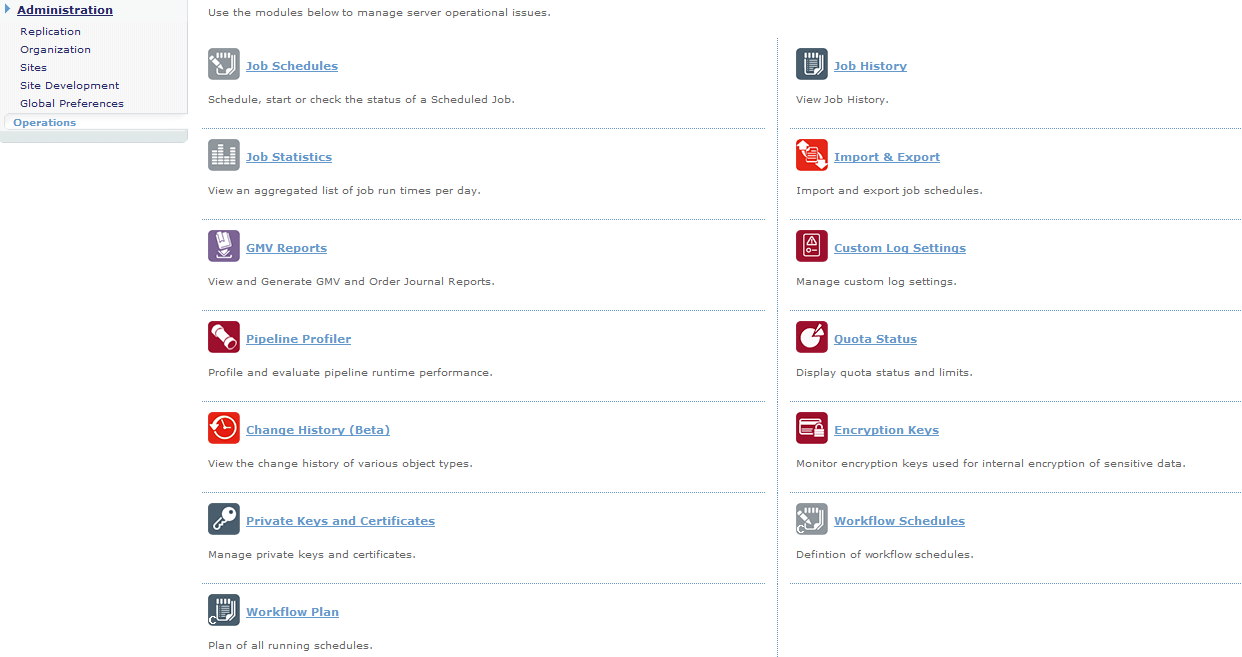
# Cartridge Installation

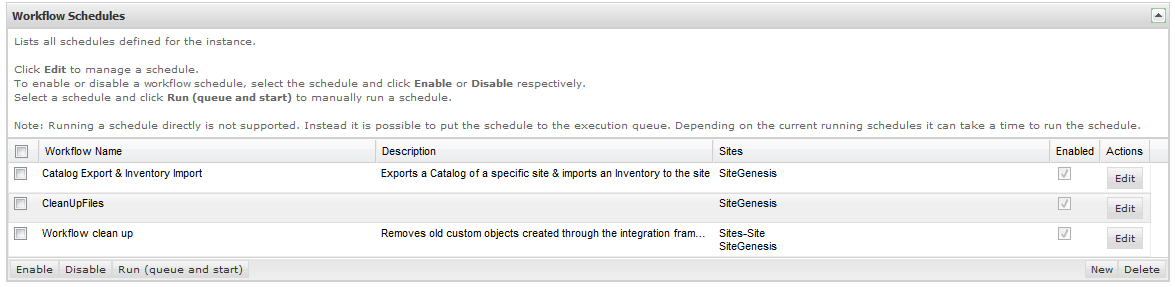
To install the Integration Framework for the first time on your instance, you may follow the following steps:

1. Download integrationframework.zip from Demandware XChange
2. Extract archive to your local file system – e.g. the cartridge folder of your project
3. Import cartridges into your workspace and link them to the Server Connection
4. Import site\_template.zip into your instance (Meta Data and Custom Objects)
5. Assign the bc\_integrationframework cartridge to the organization and all sites the jobs, using the Integration Framework, are running in – e.g. app\_storefront\_refapp:bc\_integrationframework
6. Assign the bm\_integrationframework cartridge to the organization – e.g. bm\_integrationframework:bc\_integrationframework:bm\_custom\_plugin
7. Assign Business Manager Modules ‘Workflow Schedules‘ and ‚ ‘Workflow Plan‘ to respective roles
8. You can now make use of the cartridge

# Configuration

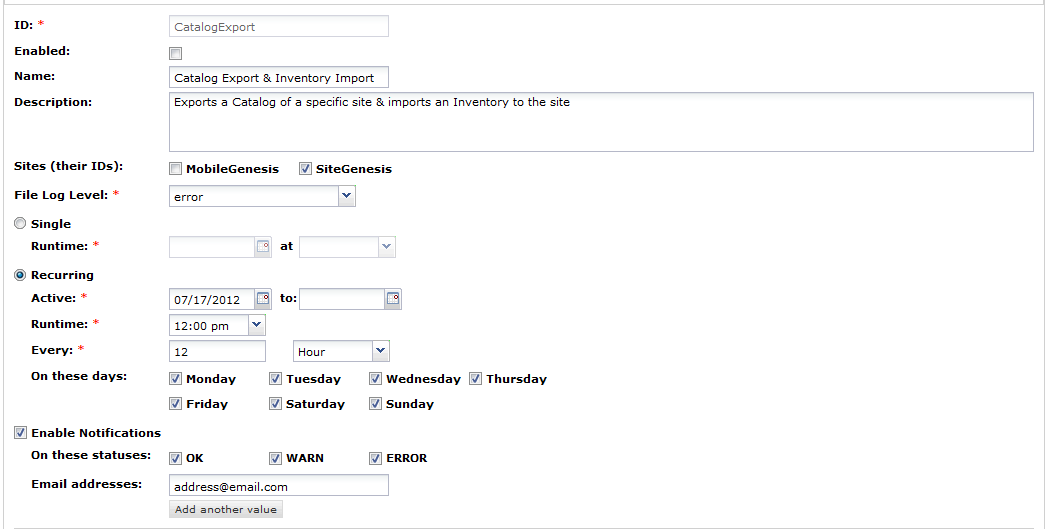
Under Administration-Operations-Workflow Schedules it is possible to create, edit, delete and manually run workflow schedules:



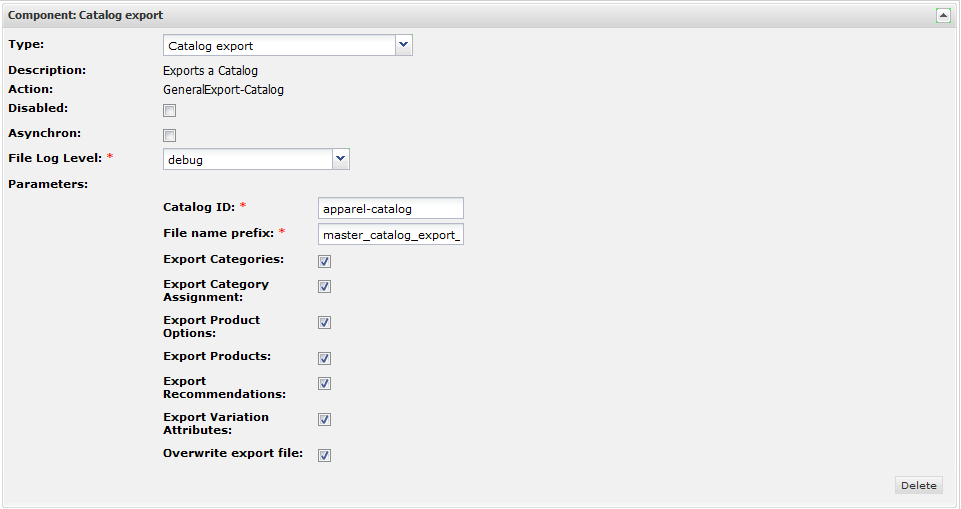


## General concept

Every workflow that needs to be executed is modelled as a WorkflowSchedule, containing main information about frequency, run times, enablement, file log level, etc.

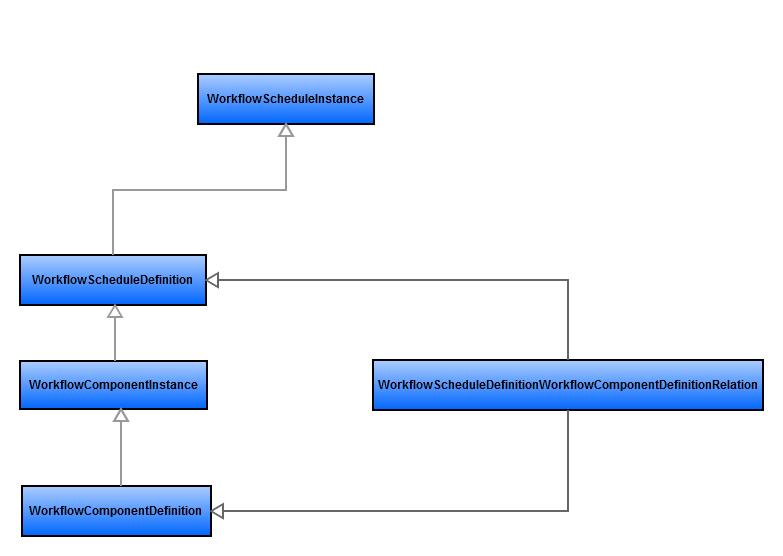


Every such WorkflowSchedule consists of simple WorkflowComponentInstances (created from the existing WorkflowComponentDefinitions), describing the steps within the workflow and their individual configurations.



For recurring schedules the schedule itself and its components are cloned for each schedule run.

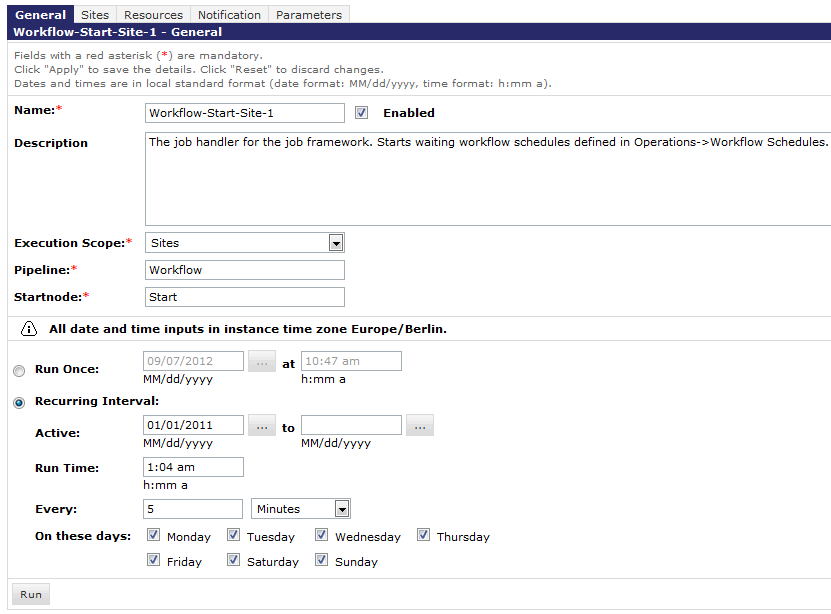
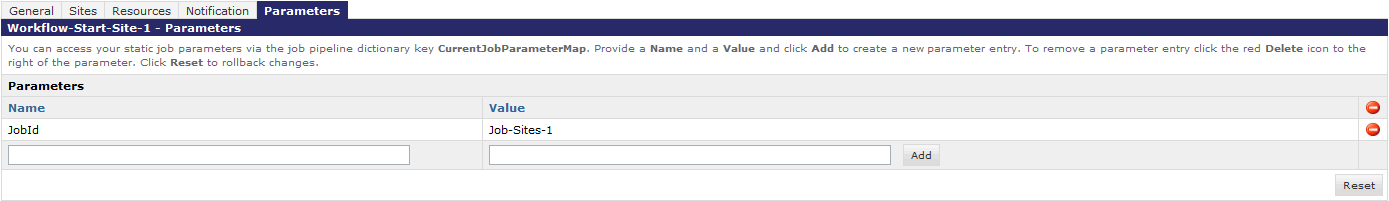
In Demandware platform terminology, a WorkflowSchedule consists of a custom object defining the workflow and a number of job schedule definitions in the Demandware BM which check for workflows to execute on a regular base. Every WorkflowComponentInstance relates to WorkflowSchedule objects as well.



## Scheduled Job Configuration

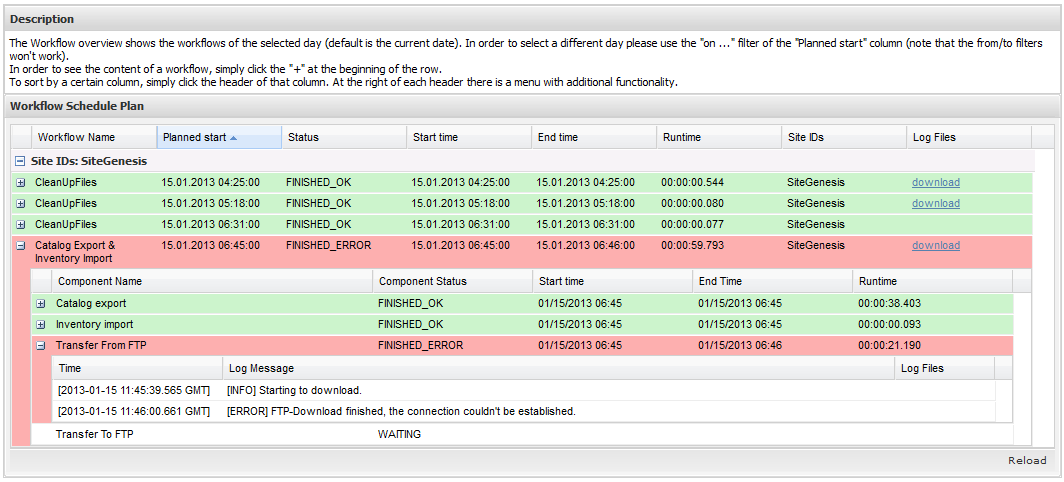
Standard Demandware jobs are used to trigger the workflow engine (the job handler framework starts waiting workflow schedules defined in Operations for a specified scope). All other calculation is done by the framework itself. This allows for example the displaying of upcoming workflows even days in advance.

To setup a scheduled-Job:

1. Navigate to *Administration 🡪 Operations 🡪 Job Schedules*
2. Create a new or edit an old Schedule
3. Set the Execution Scope to either ‘Sites’ or ‘Organization’ depending on the Scope your pre-defined Workflow-Schedule run in. *If your Workflow-Schedule should run in both scopes, please create a Scheduled-Job for each Scope.*
4. Make sure that pipeline ‘Workflow’ and start node ‘Start’ are configured  
   
5. Provide parameter ‚JobId‘ to associate a workflow component with the job which triggers the frameworks processing engine  
   
6. The Integration Framework is now completely configured and ready for use.

# Monitoring

A monitoring cockpit allows viewing the schedule of the current day as well as the schedule for a given date. For the selected day a list of Workflows that are scheduled is shown as well as their components are shown. All of them state their name, the current status and the status message.



# List of out of the box components

|  |  |
| --- | --- |
| **Components** | |
| **Component** | Description |
| **Generic Import** | The component allows to import all demandware standard xmls based on a webdav folder and file pattern. |
| **Catalog export** | The component “Catalog export” allows exporting of a specific Catalog into a designated local directory (relatively to IMPEX/). |
| **Order export** | The component “Order export” allows exporting orders (all or confirmed and paid) to a designated local directory (relatively to IMPEX/). |
| **Price-Book export** | The component “PriceBook export” allows exporting of a specific Price Book into a designated local directory (relatively to IMPEX/). |
| **Import content** | The component “Import content” allows importing of data from a designated local directory (relatively to IMPEX/) via XML-Files which corresponds to a specific naming pattern into a specified library and in a specified mode. |
| **Time condition** | The component “TimeCondition” allows ensuring that the workflow will not be continued in case a configured time is in the past or that the execution will be put on hold until a certain time is reached. This can be useful in case a replication or other workflow needs to happen before a certain point of time in order to not impact the production system performance. |
| **Date condition** | The component “DateCondition” allows ensuring that the workflow will not be continued in case a configured date is in the past or that the execution will be put on hold until a certain date is reached. This can be useful in case a replication or other workflow needs to happen before a certain date in order to not impact the production system performance. |
| **Date-Time condition** | The component “DateTimeCondition” allows ensuring that the workflow will not be continued in case a configured date and time is in the past or that the execution will be put on hold until certain date and time are reached. This can be useful in case a replication or other workflow needs to happen before a certain time in a certain day in order to not impact the production system performance. |
| **Workflow clean-up** | The component “Workflow-CleanUp” allows the removal of old custom objects which were created through the workflow framework. |
| **Clean-up files** | The component “CleanUpFiles” allows the removal and archive of old files in specific folders. |
| **Transfer to FTP** | The component “Transfer To FTP” allows the copy of files from a local directory (relatively to IMPEX/) to a FTP-Location. |
| **Transfer from FTP** | The component “Transfer From FTP” allows the copy of files to a local directory (relatively to IMPEX/) from a FTP-Location. |
| **Rebuild indexes** | The component “Rebuild indexes” allows the rebuilt of the configured indexes. This component can be useful after the import of a catalog for example. |

For detailed information regarding the components and their Attributes please refer to *Detailed-List of components*.******

**Appendix  
Developer Guide**

**1.0**



**Demandware Inc.**

# Data model and related infrastructure

|  |  |
| --- | --- |
| **WorkflowSchedule** | |
| **Attribute/Method** | Description |
| **id (PK)** | A unique ID which represents the schedule |
| **enabled** | Flag that can be used to temporarily enable/disable schedules |
| **name** | A generic display name for this workflow |
| **type** | One of “RECURRING” or “SINGLE” |
| **executionTime** | The time this workflow shall be executed |
| **executionInterval** | The interval in which the workflow shall be executed |
| **executionIntervalType** | One of hourly, daily, weekly, monthly |
| **executionWeekdays** | The days of a week this schedule will be executed on |
| **validFrom** | The time this schedule is valid from |
| **validTo** | The time until this schedule is valid to |
| **status** | The current status (“”,”WAITING”,”RUNNING”,”FINISHED”, “ERROR) |
| **endTime** | The workflow’s end time |
| **processingJobId** | The ID of the processing Demandware job |
| **start()** | Starts the schedule |
| **suspend()** | Suspends the schedule. The status will be set to “WAITING” and the processingJobId will be deleted so that the next scheduler job will continue the execution |
| **end()** | Ends the schedule and derives the status |
| **getNextExecutableComponent()** | Returns the next component which needs to be executed   * the method will skip components in status “FINISHED\_OK” or “FINISHED\_WARN” and return the first “WAITING” component * in case a component has status “ERROR” the parent workflow will be set to “ERROR” and the execution will be stopped * in case a component has status “RUNNING” the execution will be stopped |
| **getRuntime()** | Returns the runtime in milliseconds |
| **<static> getSchedules(date)** | Returns all schedules for the given date. This includes all explicit single-time schedules as well as instances of recurring schedules |
| **<static> getSchedulesForExecution()** | Returns all schedules which need to be executed at the current time |

|  |  |
| --- | --- |
| **WorkflowComponentDefinition** | |
| **Attribute/Method** | Description |
| **name (PK)** | A generic display name for this workflow component |
| **description** | A detailed description of how this component should be used, if there are any dependencies or other important thing that need to be respected |
| **action** | defines the pipeline to execute for this component |
| **parameters** | contains a JSON definition of the required parameters in the following format:  {[  {  “id” : “attr1”,  “name” : “A readable name”,  “type” : “string/number/date”,  “required” : true/false  },  ...  ]} |

|  |  |
| --- | --- |
| **WorkflowComponentInstance** | |
| **Attribute/Method** | Description |
| **id (PK)** | An auto-generated ID (<name of workflow component definition> of schedule <ID of schedule>) |
| **name** | A generic display name for this workflow component instance which is inherited from the definition |
| **action** | defines the pipeline to execute for this component which is inherited from the definition |
| **parameters** | contains the values for the defined parameters in the following format:  {  “attribute1 id” : “attribute1 value”,  “attribute2 id” : “attribute2 value”,  ...  } |
| **parentSchedule** | The ID of the parent schedule inherited from the definition, in case the schedule is recurring the ID is modified to <scheduleId>\_<scheduleStartTime (yyyymmddhhMMss)> during cloning |
| **position** | A number that defines the execution order of the components which is inherited from the definition |
| **status** | The current component status (“”,”WAITING”,”RUNNING”,”FINISHED\_OK”, ,”FINISHED\_WARN”, “FINISHED\_ERROR”) |
| **statusMessages** | Contains a list of status messages, the last item is the current status |
| **startTime** | The component’s start time |
| **endTime** | The component’s end time |
| **init()** | Initializes the component |
| **start()** | Indicates that a component has started |
| **finish(String)** | Indicates that a component has been finished |
| **finishWithWarn()** | Indicates that a component has been finished but has warnings (shortcut for finish(“WARN”)) |
| **finishWithError()** | Indicates that a component has been finished with an error (shortcut for finish(“ERROR”)) |
| **setStatusMessage(String)** | Sets the current status message shown in the monitoring UI. This is the way a workflow pipeline should log its current status. |
| **getParameter(String)** | Returns the component parameter for the given id |
| **getRuntime()** | Returns the runtime in milliseconds |
|  |  |

## Schedule execution

One or more global job configuration(s) will execute a pipeline Workflow-Start which will check periodically if a workflow needs to be started. Therefore the method WorkflowScheduleInstance.getSchedulesForExecution() is called which checks all schedules whose start time is in the past and whose status is “WAITING” or “RUNNING” (all existing single-time schedules will be checked, the cloning for recurring schedules whose start time is after the last execution time and before now is performed). For each schedule that was found the execution will be done as follows:

1. Get WorkflowScheduleInstance instance (wraps the custom object and provides additional functionality)
2. Check if the attribute **processingJobId** is set
   1. If not, set the ID of the current scheduler job
      1. if the parameter is missing log an fatal error and exit
   2. if the ID is equal to the current scheduler, there must have been an error
      1. the last last component in status running will get a new status message “Execution interrupted due to server restart” and the component will be re-executed
3. Schedule will be added to dictionary as CurrentWorkflowScheduleInstance
4. Run the start() method
   1. Will set schedule status to “RUNNING”
   2. will run init() method of all components in case the initialization did not yet happen
      1. clones WorkflowComponentInstance objects for all components in case the WorkflowScheduleInstance is “RECURRING”
      2. sets all WorkflowComponentInstance statuses to “WAITING”
5. Iterate over all WorkflowComponents using WorkflowScheduleInstance.getNextExecutableComponent()
   1. Component will be added to pipeline dictionary as CurrentWorkflowComponentInstance
   2. start() method of component is invoked
      1. sets component status to “RUNNING”
   3. dynamic pipeline call to component pipeline is made
      1. Jobs can obtain parameters by using CurrentWorkflowComponentInstance.getParameter()
   4. finish() method is called in case the execution was successful, the finishWithError() method will be called in case of an error (connector name has to be “ERROR”) , the finishWithWarn() method will be called in case there were significant warnings (connector name has to be “WARN”)
6. Run the finish() method of the schedule
   1. Derives the status of the schedule from component statuses

## Asynchronous components

In case a component needs to execute asynchronous subroutines it needs to call the WorkflowScheduleInstance.suspend() method while the component itself is in status “RUNNING”. When the asynchronous component(s) finish, they need to update the component so that it will be set to “FINISHED\_OK”. The next run of the scheduler job will then pick up the schedule and skip all finished components continuing with the first waiting component.

# Detailed-List of components

The migration of each sub-workflow includes the creation of the corresponding WorkflowComponentDefinition as well as the migration of the actual pipelines to utilize the framework functionality such as logging etc. and hardening the sub-workflows so that they can run independently. The following set of components is already ready for usage after the initial installation:

## Catalog export component

The component “Catalog export” allows exporting of a specific Catalog into a designated local directory (relatively to IMPEX/).

|  |  |
| --- | --- |
| **Catalog export** | |
| **Attribute** | Description |
| **Catalog ID** | ID of the catalog which will be exported |
| **File Name Prefix** | Prefix which will be used as part of the export-files name |
| **Export Categories** | Parameter indicating whether categories should be exported or not. |
| **Export Category Assignment** | Parameter indicating whether category assignments of the exported categories are exported or not |
| **Export Product Options** | Parameter indicating whether shared product options should be exported as part of the catalog export |
| **Export Products** | Parameter indicating whether products are exported or not. |
| **Export Recommendations** | Parameter indicating whether recommendations should be exported or not. |
| **Export Variation Attributes** | Parameter indicating whether shared variation attributes should be exported as part of the catalog export |
| **Overwrite export file** | Optional flag indicating whether to overwrite an existing export file or not |
|  |  |
|  |  |
|  |  |

## Standard import component

The component “StandardComponents-Import” allows importing any Demandware XML *(abtestabtest, activedata [as CSV], catalog, coupon, customer, customergroup, customobject, giftcertificate, inventory, library, order, pricebook, productlist, promotion, shipping, slot, sourcecode, store, tax)* from a designated local directory (relatively to IMPEX/) which corresponds to a specific naming pattern.

|  |  |
| --- | --- |
| **Catalog import** | |
| **Attribute** | Description |
| **objectType** | Select which file should be imported |
| **workingFolder** | Working folder in IMPEX (e.g src/catalog). Not mandatory and falls back to the Demandware standard folders (e.g. src/marketing for promotions) |
| **filePattern** | Naming pattern of catalog files. Uses regular expressions e.g. ^cat\_.\* |
| **importMode** | Enum of String with the values "MERGE","REPLACE","UPDATE","DELETE" |
| **noFileFoundStatus** | Defines how this component should end in case no files were found for import:  OK: Workflow continues – no emails are sent  WARN: Workflow continues –emails are sent  ERROR: Workflow stops – emails are sent |
| **importFailedStatus** | Defines how this component should end in case a file to import is invalid or import exists with error:  WARN: Workflow continues –emails are sent  ERROR: Workflow stops – emails are sent |
| **catalogImportConfig** | Catalog specific import settings – see <https://documentation.demandware.com/display/DOCAPI132/Pipelet+ImportCatalog> for further details |
| **afterProcessAction** | Defines how the imported files are handled after they have been imported.  DELETE\_FILE: Deletes file  KEEP\_FILE: Does nothing, so that file are staying in the defined import folder  ARCHIVE\_ZIPPED\_WITH\_LOGS: Creates a zipfile of files that were processed in that import including a log file  ARCHIVE\_PLAIN\_FILE: Moves all files that were processed in that import into a date/site/importtype file structure including a log file |
|  |  |

## Order export component

The component “Order export” allows exporting orders (all or confirmed and paid) to a designated local directory (relatively to IMPEX/).

|  |  |
| --- | --- |
| **Order export** | |
| **Attribute** | Description |
| **Input Order Status** | Enum of Strings with the values “All” and “Confirmed and Paid” which determines the set of orders to export |
|  |  |
|  |  |

## Price-Book export component

The component “PriceBook export” allows exporting of a specific Price Book into a designated local directory (relatively to IMPEX/).

|  |  |
| --- | --- |
| **PriceBook export** | |
| **Attribute** | Description |
| **Price Book ID** | ID of the price book which will be exported |
| **File Name Prefix** | Prefix which will be used as part of the export-files name |
| **Overwrite export file** | Optional flag indicating whether to overwrite an existing export file or not |
|  |  |
|  |  |
|  |  |

## Time condition component

The component “TimeCondition” allows ensuring that the workflow will not be continued in case a configured time is in the past or that the execution will be put on hold until a certain time is reached. This can be useful in case a replication or other workflow needs to happen before a certain point of time in order to not impact the production system performance.

|  |  |
| --- | --- |
| **TimeCondition** | |
| **Attribute** | Description |
| **time** | The configurable time |
| **mode** | Enum of string with values “BEFORE\_NOW” (means that the current time has to be before the configured one), “AFTER\_NOW” (means the execution must not be continued until the current time is after the configured time) |
|  |  |

Note: If AFTER\_NOW is selected it cannot be guaranteed that the workflow continues exactly at this time, the granularity is limited to the run frequency of the global scheduler jobs.

## Date condition component

The component “DateCondition” allows ensuring that the workflow will not be continued in case a configured date is in the past or that the execution will be put on hold until a certain date is reached. This can be useful in case a replication or other workflow needs to happen before a certain date in order to not impact the production system performance.

|  |  |
| --- | --- |
| **DateCondition** | |
| **Attribute** | Description |
| **date** | The configurable time |
| **mode** | Enum of string with values “NOW\_BEFORE\_X” (means that the current date has to be before the configured one), “NOW\_AFTER\_X” (means the execution must not be continued until the current date is after the configured one) |
|  |  |

## Date-Time condition component

The component “DateTimeCondition” allows ensuring that the workflow will not be continued in case a configured date and time is in the past or that the execution will be put on hold until certain date and time are reached. This can be useful in case a replication or other workflow needs to happen before a certain time in a certain day in order to not impact the production system performance.

|  |  |
| --- | --- |
| **DateTimeCondition** | |
| **Attribute** | Description |
| **Date and time** | The configurable time |
| **mode** | Enum of string with values “NOW\_BEFORE\_X” (means that the current date has to be before the configured one), “NOW\_AFTER\_X” (means the execution must not be continued until the current date is after the configured one) |
|  |  |
|  |  |

## Workflow clean-up component

The component “Workflow-CleanUp” allows the removal of old custom objects which were created through the workflow framework.

|  |  |
| --- | --- |
| **Workflow-CleanUp** | |
| **Attribute** | Description |
| **Maximum age of integration framework object before deletion** | An integer which specifies the maximum age in days of a integration framework object before it is being deleted |
|  |  |

## Clean-up files component

The component “CleanUpFiles” allows the removal and archive of old files in specific local folders. File access is limited to certain virtual directories. These directories are a subset of those accessible through WebDAV.

|  |  |
| --- | --- |
| **CleanUpFiles** | |
| **Attribute** | Description |
| **File Name Pattern** | File name pattern to search for in the source directory (default “.\*” for all files) |
| **Directories to purge** | The directories from which the files will be removed |
| **Directories to archive** | The directories in which the files will be archived |
| **Number of days to keep old files** | An integer which specifies the number of days the files should be kept before removal or archive |
|  |  |

## Transfer to FTP component

The component “Transfer To FTP” allows the copy of files from a local directory (relatively to IMPEX/) to a FTP-Location.

|  |  |
| --- | --- |
| **Transfer To FTP** | |
| **Attribute** | Description |
| **Remote folder of FTP Server** | The folder on the remote FTP-Server to which the files will be transferred |
| **Local folder with files, relatively to IMPEX/** | The local folder from which the files will be copied |
| **When file is uploaded, move it to archive folder?** | Should the copied files be locally archived? |
| **Type of FTP connection** | Enum of String with the values "FTP" and "SFTP" |
| **Host URL** | URL of the FTP-Host |
| **User Name** | User Name (credentials) |
| **Password** | Password (credentials) |
| **Input File Pattern** | File name pattern to search for in the source directory (default is \"^[\\w\-]{1,}\\.xml$\" (\*.xml)) |
|  |  |
|  |  |

## Transfer from FTP component

The component “Transfer From FTP” allows the copy of files to a local directory (relatively to IMPEX/) from a FTP-Location.

|  |  |
| --- | --- |
| **Transfer From FTP** | |
| **Attribute** | Description |
| **Host URL** | URL of the FTP-Host |
| **User Name** | User Name (credentials) |
| **Password** | Password (credentials) |
| **Type of FTP connection** | Enum of String with the values "FTP" and "SFTP" |
| **Input File pattern to search in local folder** | File name pattern to search for in the source directory (default is \"^[\\w\-]{1,}\\.xml$\" (\*.xml)) |
| **Source Folder** | The folder on the remote FTP-Server from which the files will be transferred |
| **Local Folder** | Local folder in which files will be placed, relatively to IMPEX/ |
| **When file is uploaded, delete or keep it?** | Should the transferred files be deleted on the remote location? |
|  |  |
|  |  |

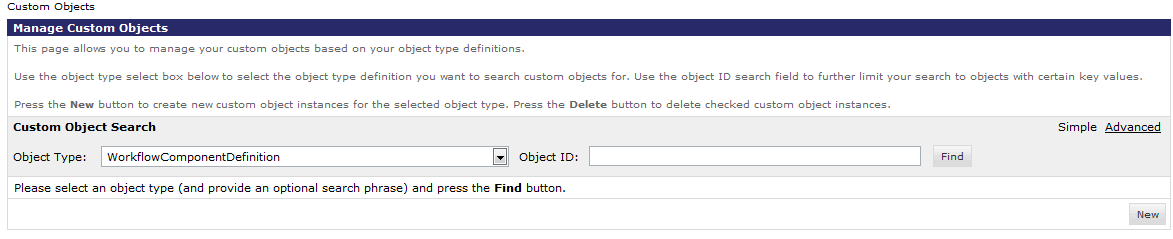
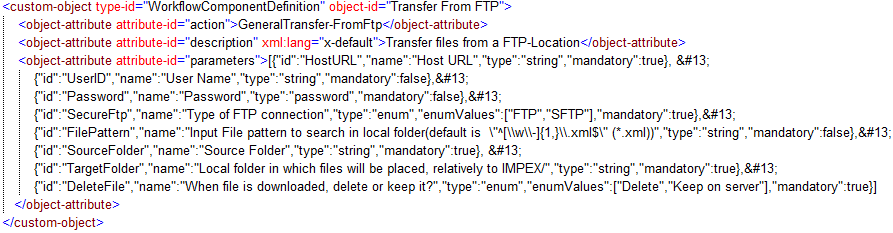
## Rebuild indexes component

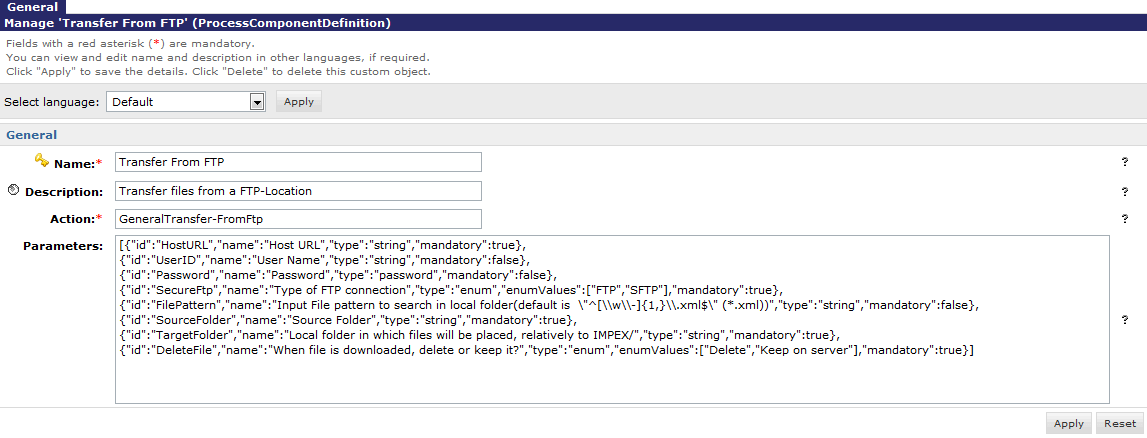
The component “Rebuild indexes” allows the rebuilt of the configured indexes. This component can be useful after the import of a catalog for example.

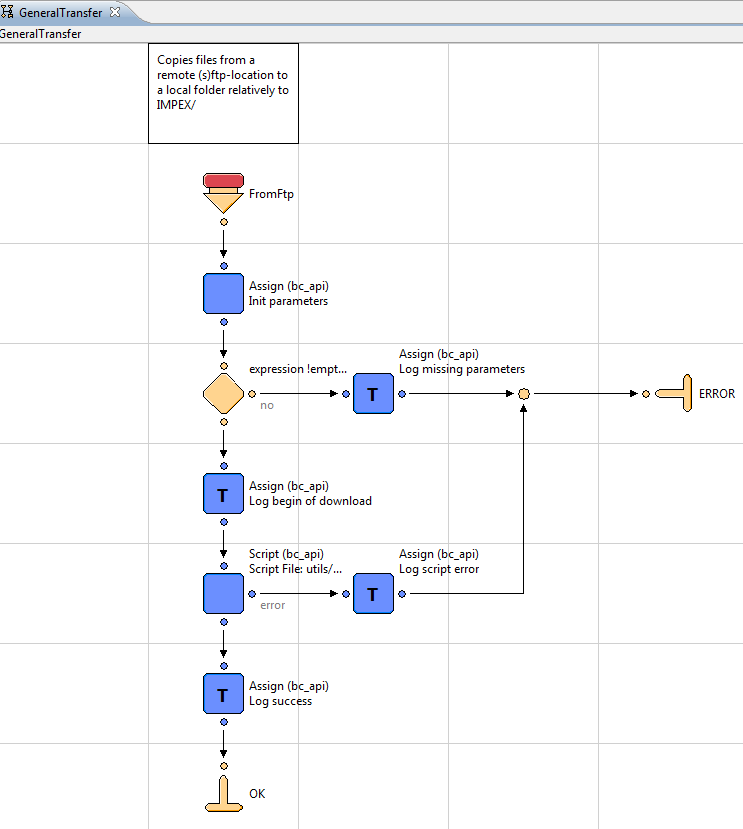
|  |  |
| --- | --- |
| **Rebuild indexes** | |
| **Attribute** | Description |
| **Rebuild product related search indexes** | Should the products, suggests and synonymy indexes be rebuilt? |
| **Rebuild availability search indexes** | Should availabilities indexes be rebuilt? |
| **Rebuild content search indexes** | Should contents search indexes be rebuilt? |
|  |  |

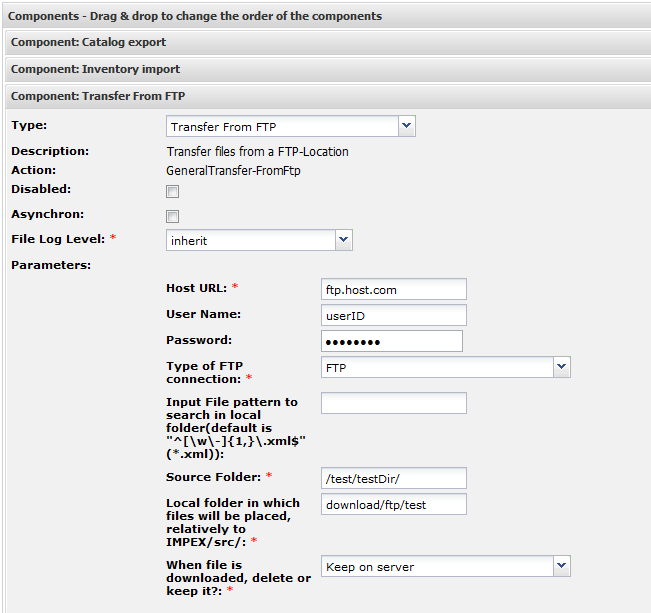
# Custom components development

In order to create a custom component and add it to the out of the box set of components in the framework, please follow the following steps:

1. *Create a ComponentDefinition*
   1. In Business Manger under Site🡪Custom Objects🡪Custom Object Editor select WorkflowComponentDefinition as an ‘Object Type’ and click ‘New’:  
      
   2. On the Edit-Page insert the Name, Description, Action (the pipeline which will be called upon execution) and the Parameters (a JASON-Object-Structure).
   3. Click ‘Apply’  
      
   4. Alternatively a XML-File with the object definition can be created and imported through the Import & Export functionality of the Custom Objects Module. 
2. Create the Action-Pipeline
   1. In the bc\_integrationframework cartridge create the pipeline as entered in the Action-Parameter of the WorkflowComponentDefinition.





1. Add the component to a workflow schedule  
   

In some cases you may want to provide the operator with additional insights into what is happening or what has happened during the execution of the component. To be able to do it you set the File Log Level on schedule level to something that is != ‘none’ and on component level to something that >= the log level on schedule level. E.g. when you pick ‘error’ on the schedule level and ‘inherit’ on component level, all error and fatal messages will be logged for the current component. When setting the schedule level to ‘error’ and the component level to ‘debug’, all fatal, error and debug messages will be logged. The other way around – ‘debug’ on schedule level and ‘error’ on component level will cause the component to log all fatal, error and debug messages. In a sense always the finest grained log level wins.

There are two ways to use file logging inside your components:

1. Using function CurrentWorkflowComponentInstance.addMessage(message , scope). This function adds a message entry to the custom object (up to 50 entries allowed) and at the same time logs the entry in the log file if file logging has been enabled and has the respective level.
2. Using the logging methods you know from Logger. Each CurrentWorkflowComponentInstance object is API compatible with Logger. This way both can be used to log messages. For instance:   
   *CurrentWorkflowComponentInstance.error(‘my message’);* is almost equivalent to *Logger.error(‘my message’’)* except for the file the message will be logged in.

Here is a nice example how you can use it in your components in the most flexible way:

**importScript**( 'bc\_integrationframework:workflow/libWorkflowLogToFile.ds' );

**var** cvLogger : Logger = getCurrentWorkflowInstanceFileLogger();

**var** cvNDC : LogNDC = Log.getNDC();

/\*\*

\* The main function.

\*

\* @param pdict : PipelineDictionary The pipeline dictionary.

\*

\* @returns Number If the given directory or the result is empty, PIPELET\_ERROR is returned. Otherwise PIPELET\_NEXT.

\*/

**function** execute( pdict : PipelineDictionary ) : Number

{

cvNDC.push( 'MoveOnlyFilesCopyDirs.ds' );

**if**(pdict.TragetDir != null && pdict.SourceFile != null && pdict.SourceDir != null)

{

**if**(pdict.SourceFile.isDirectory())

{

**var** theDir : File = **new** File(pdict.TragetDir.fullPath + getPath(pdict.SourceFile.fullPath, pdict.SourceDir.fullPath, pdict.CopyBack));

theDir.mkdirs();

cvLogger.debug( 'Ensure existence of ' + theDir.fullPath );

}

**else**

{

**var** path : string = (pdict.TragetDir.fullPath+ getPath(pdict.SourceFile.fullPath, pdict.SourceDir.fullPath, pdict.CopyBack));

**var** aFile : File = **new** File(path);

pdict.SourceFile.renameTo(aFile);

cvLogger.debug( 'Moving file from ' + pdict.SourceFile.fullPath + ' to ' + path );

}

}

cvNDC.pop();

**return** PIPELET\_NEXT;

}

The example above shows that for logging purposes a class variable cvLogger is used. The script is not being executed in context of a WorkFlowComponentInstance, the regular logger is used to log the messages.