Introduction & Reference

Document version 0.5

Overview

In this document, we have gathered information about the resources and concepts related to the platform Bosbec WE and its Workflows.

Bosbec WE

Make sure to check out the latest version of this document located at:
https://help.bosbec.io/IntroAndReference.pdf

This document

Throughout this document the Bosbec We-platform will be referred to as Bosbec, Bosbec WE or the platform. If you have questions about the documentation, please get in touch with us: support@bosbec.com

A short introduction to the structure of this document here; First sections will present the platform, and the solutions (called workflows) and how different resources and terminology relates to each other. In the middle, we have a few examples on how to get started and the rest of the document is a reference list with more information on what a job can do and how to configure it.

What are Workflows

Ok, so what we call a Workflow is in fact a template or configuration for how a series of actions (jobs) are set up with input, output, rules and what will initiate the execution of these actions. Workflows can be in two primary states, active or inactive.

When the workflow is active its triggers will react to events and the workflow may execute. But as soon as the workflow state is changed to inactive all instances of the workflow will stop, that is, not just inactivating new executions but any current executing workflow will also stop and not execute jobs further.

There is also a partial-active state that may be used while developing or troubleshooting a workflow. It is possible to have an active workflow and make one or more of the executing instances pause execution between jobs in what we call a Debugging-state.

Workflows always starts with a trigger reacting to an outside event. Such events can be a schedule from the scheduling service, an uploaded file from the file service, incoming messages or incoming API-requests. A more detailed description of the different resources and parts in a workflow will follow. After the trigger has reacted it will create a workflow context (WFC) object, which is kind of a box of resources and information that is used throughout that one execution from the trigger through all connected jobs until there are no more job-connections.

Workflows typically make use of resources from the account and/or temporary resources in the workflow context. Resources from the account could be units, groups and messages and temporary resources during the workflow-execution could be an incoming message or a JsonResource. These resources are described in greater detail further into this document.

Workflows will typically end with some job that send a message to a recipient, be that a text message through app/email/sms or a request to another server. Another common scenario is where the end of a workflow is one or more jobs that will log some data.

Workflows are used for a wide range of purposes. Here are just a few different takes on what have been created with Workflows.

- For empowering Bosbec customers to take control of important processes such as importing units or to configure the message-sending process.
- Logging results or data-input from different sources. For example, connect a temperature-device via MQTT-protocol and store values in the DataLog. Improve that solution by letting the workflow test the incoming value with the average of values from last week and notify user.

- Configure and send questionnaires (forms) to recipients via email, app and SMS and either take different actions depending on answers or export the results as CSV-file and get results on FTP or as email-attachment.
- As the implementation of “Bosbec services”.
- As Group-Chat where everyone can interact with their own preferred medium of message.
- To integrate old or legacy systems into a new setup/environment.

Concepts

In this section, we will describe following concepts from Bosbec terminology.
For a more in-detail description of different jobs and triggers look at the reference section at the end of this document!

Triggers

Triggers are perhaps the simplest concept we have. Their responsibility is only to act as a starting point, a “trigger” if conditions meet the configured setting, then start the execution of a series of jobs.

Some triggers provide unique data when starting an execution of a workflow. For example, the FormAnsweredTrigger will set the FormAnswer in the workflow context and make this available for jobs that can handle FormAnswer-data. The IncomingMessage triggers will provide an IncomingMessage and the HttpTrigger will provide a HTTP-resource.

Jobs

Jobs are the building-blocks that each perform an operation on the resources and data in the current execution of the workflow or on some resource on the account. The jobs can connect to each other and will work together to create more complex solutions for whatever problem the workflow intends to solve.

Jobs come in different sizes and complexity. Almost all the jobs that can be created are available in the workflow-builder for you to use. Some of the jobs does however make sense only in certain cases or as part of supporting old workflow processes.

Jobs will continue to evolve as the platform grows and matures. This means that jobs are built in new and more efficient ways now, compared to a few years ago. It also means that as more capable jobs, such as UnitOperations or DataOperations emerge the specific jobs such as SetUnitsMetaData or GetMetaData gets obsolete and can be replaced.

The MetaData concept in Bosbec

Metadata is a fundamental concept to grasp in order to work with resources and workflows. Metadata in Bosbec is a key and a value. The

key is what we use to indicate that we want the specific data, and the value is the actual data. For example, I may store my first name in a metadata with the key **firstname** and then the value could be **Bosbec** if that is my name.

All resources, workflows, jobs and triggers can store metadata. When adding metadata to a unit, we add describing attributes to it. Examples of such metadata can be the first- and last-names we would add to a unit when that unit represents a person.

With the metadata we can store extra information about an object or a resource that we work with. This information can be accessed during workflow execution and used to replace keywords in message templates and similar resources.

When the process flows through a workflow, the workflow has its own metadata just for that one execution (from the start of the trigger through the series of jobs). And this metadata is used to store and modify data between jobs. An example can be that the first job after my trigger will get the text-data from the incoming message. And then I can have that job modify or just store the text-data in a metadata that is addressed by name. Then it is possible for the second, third or some other job down the line to access this metadata by name and use it or modify it.

To summarize, metadata are variables, data that can be accessed by addressing the name of that data and may be added/removed or modified.

Resources

Workflows uses many different kinds of resources. They may manipulate data on account resources such as Units or Groups. The workflows have jobs and triggers that can be connected to Form resources. Jobs will send messages that are generated from the MessageTemplates.

But workflows can also use more temporary resources like the data in metadata (described above), JSON- and XML- resources. And below is a section on what resources are available and how they may be used.

Units

A Unit is our most generic model. The idea for a unit is that the system should not only dictate how the unit will model important concepts for different customers. One unit may represent a person within an organization and need data like first- and last-names or an email-address and phone-number. Another unit may be used to model an activity in a Bosbec service.

Units are used to model arbitrary abstractions and units are suitable for storing data about that abstraction. Examples of when units are used as models:

- *First come, first served* [1] – workflow, uses units to represent the state and results for each round. One unit is used per round.
- In incident-services, some units are used as a representation of that "activity" and by default this unit is hidden for most users when browsing www.bosbec.io
- IOT (Internet-Of-Things) – devices communicate with a workflow, a unit represents the current state of that device and we store current battery-status, current temperature in the room. One unit is used per IOT-device.
- Customer SMS survey – each unit model one customer.

Units have app, phone, email and IOT (MQTT) endpoints, these channels are default properties for a unit today, but this will most likely change in the future. In most cases one or more of the communication channels are used, but in some cases (like first example given in listing above) we won't need any communication channels configured for the unit. We will not be sending messages to that unit.

Groups

A group is a collection of units. Groups cannot contain anything other than units, but a group can have metadata. Groups can be created manually by adding members (units) to the group in www.bosbec.io or having a workflow add members to a group upon a given event.

Another type of groups called "Dynamic Groups" is useful in some cases. A dynamic group is a group that automatically will add or remove members depending on whether or not they match the configured criteria.

An example of a dynamic group could be where I set tags to my units, and the tag is a text with the name of the city where the modeled receiver is located. So, if I need a group with all my receivers in Stockholm, I would set the criteria to match units with the tag **stockholm**. Each time a unit is added or updated with the tag **stockholm** the group would automatically add that unit as a member.

Most use cases with groups make use of the regular groups and a combination where import-processes and workflows keep groups up to date.

Messages

The messages (MessageTemplates) are just as the name suggests a template. A MessageTemplate may contain one template per message type.

- **App-message** to send messages to Bosbec - iOS and Android Apps.

[1] Typical workflow that solves a case where the employer needs more workers for next shift, send a question/message to off-duty personal and the first to answer will be offered extra work for the next shift.

    - Unique properties are SenderId and InboxId can be configured to send app-messages from different app-users.
- **Bosbec-message**, this is a new kind of message that is about to be released together with an updated version of incident and staffing-services during 2019. More information about this message type will be added after the release. What is important to know is that the Bosbec-message will be used to replace the app-message.
- **Email-message** to send email-messages to receivers with email configured.
    - IsBodyHtml property should be true if sending email with HTML-content.
    - From should be a valid email-address.
- **IOT-message** to send MQTT-messages to a specific service/topic that is defined by the receiving unit.
- **SMS-message** to send SMS text-messages to mobile phone receivers.
    - SenderName must be valid alphanumeric (or mobile-phone number)

To be able to send a questionnaire, a form, which in turn is replaced with a link or ShortLink[2]. When sending a form in an Email or SMS message you need to put the text **[form]** in the body of the message to let Bosbec know where to insert the link to the form.

When sending messages in a workflow, it is possible to insert unique data for each receiver simply by typing the metadata key, for instance **[firstname]** would cause BosbecWE to try to replace the text [firstname] in the message body with data found on the receiving unit.

It is also possible to replace parts or the entire message body with data from the workflow context metadata. Then all receivers will have the same replacement. For example, if I were to have a message body: "**[my-data-1] [firstname]**!" and in some way set the workflow context **metadata.my-data-1** to "**Hello**", then all receivers of my message would have a message saying Hello (from workflow context metadata) and their own first name (defined in each unit's metadata).

Forms

Form Templates can be created with Workflow-Builder or with the form builder that can be found in www.bosbec.io. The form-template is then used when generating unique forms for each recipient.
The reason we are creating unique forms for each recipient is so that we

[2] BosbecWE has a URL-shortener service that creates a 5-character long code and extend the qlnk.se which redirects users when clicking the "sort-link" and logs information about when last visit was and so on. Link would look like http://qlnk.se/A1b2c

can both have unique data/message shown to the customer and so that we can track what unit/recipient has not yet answered.

Given a simple case of how to use forms:

I want to ask a group of customers about their last visit to my store.

Then I would create a Form in Bosbec Workflow-Builder and add a Path (each path contains a question) and select the type of question I need my customer to answer. For each other question I would like my customer to answer I would simply add another Path, drag the arrow from last path to the new one and set up the question. When I am done with setting up the form, I will simply connect it to the message (MessageTemplate) out on the workflow and add the [form] text to the message body.

When dealing with the form answers it is important to think about what the result should be. If I want to act directly on dissatisfied customer then I might use the FormAnsweredTrigger, so that I can test if customer satisfaction is above, let's say 4 in a scale question from 1 to 10. If the customer answers 1, 2 or 3 then I would like to send another message to the customer, and I would like to do this as fast as possible so that I can find out the reasons for low satisfaction.

In other cases, I may be pleased with getting the results from the form on daily or weekly basis. And of course, there are more than one way to do this too.

> If I want a plain CSV-file with the results sent to my email I could set this up with the RequestExportFormAnswers (REFA) -job. The trick to get this right without having to look into the documentation is to drag and drop that job on to the workflow canvas. Then (inactivate and) save the workflow. Refresh. Now the REFA-job will have a default configuration. Make sure to set **UseHeaders** and **AutomaticQuestionHeaders** to **true** and then all you need to do is connect it, activate it and save and activate the workflow again.
>
> If you want to have results, but export the results back into workflow context, so that you are free to send results with any message type, or perhaps take some action depending on the results. Then the job ExportFormAnswersToWorkflowContext (EFATWC) will be more suitable fit.
> EFATWC will require a little bit more setup, look at the detailed information in the reference section below.

DataLog

If you want to store more than just the current/present value, which is what a unit will store. Then the DataLog is a better option. The DataLog has one part called the DataLog-document that can tell what the log-series is, and the DataLogDocument also has the key used to address a DataLog.

The other part of the DataLog is the item. A DataLog item is each time we add something to the log, we add a value and a time is set. It is possible to add metadata and a time indicating a non-system time (for cases such as a device that tracks data but only pushes data once every 24h).

The DataLog is designed for cases where it is interesting to know either what the current value is or some aggregation (max, min, sum, avg) of previous values.

Even though the DataLog can store any data supported by workflows (ex. JSON, Numeric, Plain text ..) the aggregation functionality will only work on numeric values in the DataLog. It is possible to store a mix of entries where some are values, and some are text and still make use of the aggregation functionality.

AuthenticationTokens

When working with Bosbec you will at some point come across the concept of authentication tokens. These are not as available as a unit when it comes to working with resources in a workflow execution, but they are an important part of the system.

The AuthenticationTokens are “keys” that is used to validate requests to APIs. For example, when you log in to www.bosbec.io, the user interface will use authentication tokens behind the scenes. Another case is when we use APIs, such as REST-API or HTTP-in API. When a request has a token, the API knows what account and what administrator that is responsible for the request. In some cases you need a token that is valid during your visit to www.bosbec.io, but in other cases you need something more from a token.

The authentication token can be created with API-requests (via the login-request) or in www.bosbec.io -user interface. If a token is created as a “Persistent token”, then that token will have a longer time to live before it is expired.

The authentication token is always created with a specific rights-configuration so that every system or user can have a limited amount of rights to access the different APIs and API-functions.

Services

Services is an extension of what workflows can do. Services is still only build by Bosbec dev- and support-team. This is because there are no user-interfaces and public APIs/tools available for that yet.

Services is a series of GUI-steps that result in input-data to the execution of an ExecutingWorkflowTrigger. With services it is possible to create a specification of what resources and input-data a workflow needs and let any user execute a workflow without any knowledge or understanding of how it is made or what is required to set up such a workflow.

In combination with having some administrators that only have permissions to view that service, this is a powerful tool for organizations to make sure that functionality is available to all users, but only a selected few may know how to change the configuration of the workflow.

Accounts, Administrators and AppUsers

There are two types of users in the platform; Administrators that may log in and work in the admin GUI and create workflows. And the AppUsers that may use the Bosbec app (web-app or smartphone-app).

Account & Administrator settings

The account is highest up in the hierarchy, which means that each customer will have ONE (1) account and typically have multiple Administrators on that account.
Some settings for example the default SMS-SenderName can be set on the Account level. And may then be overridden on the Administrator level.

Example:
Account has sendername Bosbec, and Administrator_1 has sendername Number1, while Administrator_2 doesn't have any setting that overrides the default.
This would result in a scenario where Administrator_1 will send messages with the default sender name **Number1** while Administrator_2 will send messages with the default sender name **Bosbec**.

The same can also be done with billing-id, so that different administrators may have different payment-procedures or payment methods configured.

CsvResource

The CsvResource can be created when parsing data (e.g. metadata) that contains comma-separated information. The resources are then addressed by the name of the resource followed by brackets and a number. For example, **mycsv[0][0]** would get me the data from the first row and the first column.

FormDataResource

The FormDataResource can be parsed from an incoming http resource if the data is http-post form-data.

FormResource

The FormResource is a resource that can be created of found by jobs. And this resource will let you work with a form.

GroupResource

The GroupResource will be created or found from jobs. And will let you work with one or more groups contained in this resource.

HttpRequestResource

The HttpRequesResource is added to the workflow when a HttpIn-trigger is executed. The resource contains information about the request to the http-in-API (in.bosbec.io)

JsonResource

The JsonResource is a resource that can be parsed from a data source (e.g. metadata). The json resource will then let you address and modify the json structure with a simple syntax such as **myjson.somedata** and get and set data just like WFC metadata.

LogItemResource

The LogItemResource is the resource-version of a DataLog LogItem.

MessageResource

The MessageResource can be created or found via jobs and can be used like the language-packs for the Bosbec services where they are found and dynamically connected to the send-jobs depending on what metadata/setting is provided for the country/language.

MessageResource may modify existing MessageTemplates on the account.

UnitResource

The UnitResource can be one or more units which may be used in jobs such as UnitOperations.

The UnitResource can be used to modify units in the current WFC or on the account.

XmlResource

The XmlResource can for example be parsed and used in cases where incoming XML is provided.

WorkflowContext

The name can be somewhat misleading, we are not just talking about what is near or related to a workflow, but what resources and data are available during one execution of a workflow. It is therefore more accurate to call this the Workflow-Execution-Context, but we will simply refer to it as the WFC in the texts below.

The WFC have the following data available to jobs in one way or another:

- **IsDebugging** → A workflow execution can be set in debug-mode. What it means is that the execution is paused, and no further jobs will be executed until API-call (available in Workflow-Builder GUI) comes in and either un-pauses or just play one step (that is the next job to be executed).
- The **TemporaryGroup** – This is one of the places where workflows put units before processing them or sending messages to. Jobs

that can create or find units will often be able to put the result in the TemporaryGroup.

- The **Workflow Groups** (in some cases referred to as **Workflow Context Groups**) – These are groups that you add to the purple panel to the right in the workflow builder. They are then "lazy"-loaded into the WFC. And will be included in jobs in certain combinations. For example, SendMessageToGroups without any groups attached, will default back to sending to all members in TemporaryGroup and Workflow Groups. But the members in Workflow Groups are NOT loaded into the workflow context temporary group unless it is explicitly done with the ExtractData-job.
- **Incoming Message** – This is only present if the workflow is started with an IncomingMessageTrigger (ex. IncomingSmsMessageTrigger). Incoming message can be used to get to/from or body.
- **Incoming Unit** – The incoming unit (a name to be revised in the future, since it doesn't give a clear meaning to the concept) is set by a few jobs, often when finding/creating units. This unit's metadata is accessible from many jobs that can read or alter the data.
- **Incoming GroupMember** – The difference between this concept and the IncomingUnit is that incoming group member is only set after passing through the accepted path in AllowSendersFromGroupRoute. And a group-member's metadata can override a unit-metadata.
- **FormAnswer** – This is only available after workflow has been triggered with a FormAnsweredTrigger. The answers in the FormAnswer can be retrieved with the GetMetaData-job.
- **ProcessId** – The process id is available in jobs that can access workflow context metadata. And can be used to track what process and what TextLogging-events that indicate how the execution went.
- **Files** – Files is only available when an IncomingEmailTrigger was triggered and has attachments OR when the FileTrigger is the trigger that started the workflow.
- **MetaData –** Metadata is one of the most powerful resources of the workflow context. It is used to store data between, during and after the execution of a job. Basically, you set your metadata in one job and read it later, or even let it be passed along to when we are creating and sending messages so that it may replace the keywords within **[..]** (see section on messages)
- **BillingTags** – The billing-tags can be set in the same way as dealing with metadata. The billing-tags will be forwarded to each order (after they are set of course) in that execution. This is used when execution of the same workflow should be split between one or more invoices.

- **Resources –** A Resource can only be set with jobs that can create resources. Resources can then be addressed by name in WFC, similar to metadata.

The Process and Process-Events

During the execution of a workflow we have this thing we call a Process. The process is mostly, but not entirely a log of what is going on. The process starts at some point, often this is when we register an incoming message or when a workflow is executed from any of the trigger-types. And a process ends when there is nothing more to register. There are not yet anything that set a process to a final state, much like a workflow that has no final or completed state. A workflow execution is over when there are no more jobs to execute and no more evaluations left that can lead to a new execution. Other processes in the system can be related to actions from the API.

When a job is executed it usually sends out a Process-Event. These events can contain different kinds of data, and in most cases, they hold some text-logging data. The text-logging data is a more human-readable information on what happened during the execution of a job.

In some cases, events can cause a workflow to trigger (the ProcessTrigger) or some cases where events are used to filter app-users that have received an app-message from the given process.

Tools

There are a few Graphical User Interfaces (GUI) to aid us when working with Bosbec and workflows.

Administrator-GUI at www.bosbec.io

The first GUI that most customers will visit and learn to understand is the www.bosbec.io. This is the primary interface for managing everything from units, workflows to tokens, rights and billing information. Most simple users will not go any deeper than this.

Guides and tutorials are available and updated to keep users informed on how and what to do in the different sections of this GUI that usually is called “Admin”.

Important sections to be familiar with when creating and working with the workflows are:

- The workflow area, where you are presented with an overview of workflows, their activity and the state.
  By right-clicking the workflows it is possible to edit the workflow in the workflow-builder.
- The statistics such as MessageHistory (used to know what messages and API-requests that came in and out of the account).

- The account settings and billing-information (to know about transactions for your account).
- Services, where Bosbec services are installed and used similar to apps in smartphones.
- Forms, which is where the existing form-templates are located and this is also the way to open/edit and create new forms using the form-builder.

WorkflowBuilder

The workflow-builder it is possible to create, units, forms, messages, groups, add resources to workflow, and to create the series of triggers and jobs that make up the workflow.

A vision is that the workflow-builder should resemble a whiteboard. You should feel free to organize the visual resources in a way that makes sense to you and add descriptions to jobs in order to make a complex solution more comprehendible.

Note: There are more in-detail description on how to work with the workflow-builder in the section with examples and tutorials.

The workflow builder has resources in a toolbox to the left.
There are all the visual resources that may be dragged onto the canvas.

To the right there is a purple area that will expand upon mouse pointer hovering over it, this area is used to add groups as workflow groups.

At the bottom right there are zoom- and pan- controls, which is something that also can be achieved with the right mouse button and the mouse wheel.

There is a menu at the top left, and from this menu you can

- Save the current workflow or create a version. Versions of a workflow is very useful, since this will let you save a working state of the workflow and try out new features without having to lose the previous working version.
- Perform actions such as toggle the active/inactive state for the workflow, and to open the Execute dialog.
- The view-menu will let you see the entire workflow as JSON-data.
- And some shortcuts for complex settings are available in the shortcut's menu.
- And beside the menu there is a search field, which will let you locate a resource on the canvas that matches your search. For example, you can search for a metadata-key and click **Next match** to iterate over matches that in some way address that metadata.

1. The simplest description of how to operate the workflow-builder would be to say that you start with your trigger, drag it out and select the kind of trigger you need.
2. Configure the trigger (refer to the reference-section or click the help “?” tab in the properties window to know more about the trigger)
3. Add the jobs and connect the trigger to the first job
4. From the first job to the second and so on.
5. Connect resources needed from the jobs, for example; the SendMessageToUnits job will need one or more units and a message template.
6. After all parts are created, make sure to save the workflow and correct any issues that occur when saving.
7. Activate the workflow with the switch in the upper right or through the menu.
8. If it is an ExecutingWorkflowTrigger that was selected, then the workflow can be tested out with the action Execute under Actions-menu.

FormBuilder

The form builder can be found at the forms-area. Forms can be created in the workflow-builder as well, but the form-builder is specialized on displaying and building just the forms. It can create new or edit existing forms. You can set, change, remove questions and configure the form in the form builder and then use it in the workflow-builder.

Getting started

This section is about getting started with workflows. A few examples below will help you with most of the concepts and tools and give a basic understanding of what it is like to work with workflows.

Case study 1 – Forwarding information from the incoming message

In this case we are working with a system that automatically generates an email every morning with a system log for System XYZ.

The information in the system-generate email ha the following body:

```
Operation log
Name: System XYZ
Date: 2019-01-01

Events: 8
04:37 Gate 3 open
04:39 Gate 3 closed
10:04 Door 1 open
10:04 Gate 3 open
10:05 Door 2 open
10:05 Door 1 closed
10:05 Door 2 closed
18:04 Gate 3 closed
-

Errors: 0
-
```

The task is to create a workflow that can react to the number of errors and notify the staff if there were any errors during the last 24 hours. What we know about the case:

- If there are no error, the workflow should report that everything is **OK** with **System XYZ** in a text message.
- But if there are errors, the workflow should report the **amount of errors** in a **text message**. **And forward the entire email** as well.
- The system can be configured to send the daily generated email to any email address.
- The email structure will be the same every day. The amount of errors may vary, and they are listed just like the events in case there are any errors.

How to solve this case with a workflow

This guide assumes that you have an account set up and configured that is ready to send messages.

First step is to log in to www.bosbec.io and create a new workflow.

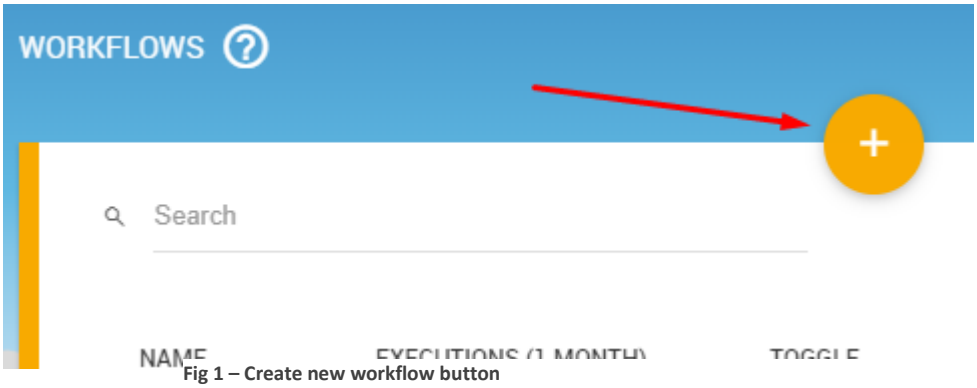


Fig 1 – Create new workflow button

You should browse to your workflows in the menu alternatives and click the yellow button with a plus-sign to create a workflow.
Name the workflow and enter the workflow builder when prompted.

Then we need to consider what we know from the information given in this case; There is a way to get information from System XYZ if we could direct the daily generated output into our workflow.

1. Create a way to catch incoming email

Incoming messages to Bosbec can be reserved by creating a trigger with the desired configuration. If the configuration is marked as suspicious or to generic the support team need to manually review the request for a reservation.

We will in this case be very specific so that the reservation will work.

The mail servers for the Bosbec system have one mailbox open for this type of reservations. By registering an arbitrary email-address at the @qlnk.se -domain such as unique_demo_email_20190101_xxx@qlnk.se in the trigger we should be good to go.

By now you should be sitting with the workflow-builder in front of you with an empty white background and the visual resources to the left.
Drag the yellow icon with a lightbulb onto the canvas and select the Incoming Email Message Trigger.

When we have an incoming email trigger on our workflow, we need to configure the trigger to match the conditions for our case.

When resources are created on the canvas, they often come with default settings that need to change.

By double-clicking the resource (trigger in this case) icon *or first click and select, then clicking the settings-icon*  a new window will pop up with information about that resource.

The properties for the resource are now visible in a way similar to how we're used to see them inside www.bosbec.io and the tab "Properties" are highlighted in the top of the pop-up window.

There are also tabs for help (the question-mark), and for the JSON (*the representation of a trigger/job/.. as JavaScript-object.) The JSON tab is used in some cases where we troubleshoot and work our way around issues with saving or as a way to know what data the APIs will receive when we save*)

- Clear the keyword from the line of zeros (00000000-0000….)

- Set the receiver to a unique email to use in this demo, example can be the current date + your name or a random series of numbers and end with @qlnk.se e.g. 2019-01-01-12345@qlnk.se

- Also set the sender to an email you are in control over for the test / simulating the System XYZ – report mail.

To make the workflow builder know that we are ready to save this trigger, we must close the dialog with the "x" -button in the top right of the pop-up window.

When the window has closed, we should go to the **File** menu and click **Save**. This causes the workflow-builder to save the workflow as is currently is represented. If any errors occur, there will be a red circle around items that could not be saved.

- Close the properties dialog using the x in the top right of the pop-up window

- Click the file menu, and click save

IF YOU'VE ENCOUNTERED ERRORS SO FAR, REVIEW THE STEPS OR CONSULT WITH SUPPORT OR YOUR PERSONAL CONTACT AT BOSBEC!

2. Testing the first steps

We will do everything in baby steps in this first case-study / tutorial and

pause to try out our setup in order to better understand how to troubleshoot workflows further on.

You've successfully added a trigger and saved the workflow.

To make sure everything is in order you should be able to close the workflow-builder and reopen the workflow, to find that the trigger is where you've left it, with the configuration still intact.

Find your way back to the workflow by exiting the workflow builder and re-enter and confirm your setup this far.

And since the workflow is in the same state as we've left it we should go ahead and activate it, which is a precondition if we want to actually send an email that will trigger our first execution of this workflow.

Us the toggle in the top right of the workflow-builder, or better yet see that the toggle change when you go to the **Action** menu and click
Toggle active state

By now we have created, saved and activated the workflow.

We are ready to test with an email.

Send an email form the email address you've configured as **sender** to the target **receiver** that you've configured for the trigger in the steps above.

It shouldn't take more than a few seconds for the email to be delivered to the systems' inbox.

And when the trigger tests and matches the incoming email to what has been registered/reserved for your trigger it will start the workflow. Even if the workflow is just a trigger it will start and create a workflow context (the information we have access to during that execution of the workflow).

To find out what this looks like you could head over to workflows section in www.bosbec.io and right click your workflow. The menu-alternative "Workflow execution contexts" will take you to a list where all workflow executions for that workflow are shown.

In the first listing there is an overview that shows one row for each time that the workflow was executed. You should be seeing one row with information and under the column "Incoming message" you should see a summary the message you've sent (Sender/To/Message).

To confirm that the test was successful go to the workflow listing and right click. And make sure that your test email shows up as one of the rows in the workflow execution contexts.

3. Extending the functionality

Now we're ready to expand and add more functionality to the workflow. If you'd like to save your progress you can use the menu "File" and click "Save version", and you will have created a point that you may go back to. This is an optional step. And it can be used whenever you feel that you have made changes that works, and you would maybe want to go back to this state again.

We begin by inactivating the workflow and the trigger, this reduces the risk of issues when updating and saving triggers (it will otherwise be prompted to allow inactivating during saving).

Back in the workflow builder, inactivate the workflow with the toggle in the top right corner.

The trigger should be dimmed indicating that the trigger is not active. We can use the icons from the toolbar to the left side of the screen or right click the background to add a new job (the red circles with gear-icons).

What wee need to do is, have the workflow read the text from the email body and figure out if there were any errors or not.

First, we need to extract the line starting with the text **Error:** followed by a number. And when we want to operate on some data we should consider if the DataOperation-job can help us.

Drag a red circle with the gear-icon and select the DataOperation job (tip: use the filter and begin typing data operation and you'll find it easy)

The DataOperation will let us perform operations on some data in the workflow context, and our incoming message and its body is available to us since we've used the incoming email message trigger.

Open the DataOperation job and add a "**New operation**", select the "**extract value regex**" and configure it to match the text **Errors: [0-9]*** where the **[0-9]*** is a "Regular expression" / Regex that will match one or more numbers. If you want to know more about regex there are many resources online. However, do note that it is just a subset of regex-operation and dialect that are supported in workflows.

Open settings for the DataOperation-job you've created and add a new operation of type **Extract value regex**.
Configure the operation to get the source text from **incomingmessage.body** and to put the result in the destination **metadata.result**
Finally set the regex pattern to **Errors: [0-9]***
and configure the options so that the **FirstMatch** is check.

With the job we've added we will now get the information from the row that specify how many errors in the last 24 hours. But the text in the result will not only contain the number, but the text **Error:** and we need to remove that before we let the workflow handle different actions depending on the number.

Add another operation to the DataOperation-job. This time select **Substring** and configure the source to be **metadata.result** the destination **metadata.error-count** and the start index **8**.
Also remember to set the order of this second operation to a number higher than the first (eg. Order = 2)

This second operation, the substring-operation, will take a part of the text based on a position defined in the **start index.** If we count the characters including the space between the colon and the number "**Errors:** ", we get 8 characters, and we want the partial text that starts after this.

So, by now we will get the text from the incoming message, find our number and stored the number in the metadata for our workflow context in a location we call **error-count**.

4. Another test (optional)

If you want to test out the new functionality there are a couple of ways to make something happen that will give you instant feedback.

Either you feed comfortable with the JSON-representation of the workflow context data (found in www.bosbec.io at Workflows → Workflow Execution Contexts → Details (the right column called Full context). In this view you will see all information that the workflow used before it finished. It is also possible to view the events with information about the process (each job will report a short summary of the process)

But a more common way to deal with this before getting to know more under the hood would be to make the workflow respond with the current result in an email.

a) ***TIP!***
THIS IS A GOOD TIME TO MAKE A VERSION BEFORE THIS OPTIONAL TEST.
To add a version, click File-menu and the Save Version.
With a saved version you can go back to before this test changes your workflow!

b) Add the job **Answer Sender** and add a **MessageTemplate** (the green circle with a word-bubble)
c) Connect the AnswerSender-job to the MessageTemplate.
d) Double click and open the MessageTemplate and add a **New message** of type **Email.**
e) Configure with something similar to:
    a. Subject = **Bosbec tutorial**
    b. From = **noreply@bosbec.io**
    c. Body = **Hello!**
    The result from the first (regex-)operation was: [result]
    And after the substring we're left with: [error-count]
f) Connect from the **DataOperation** to the **AnswerSender**
g) Save the workflow
h) Activate the workflow
i) Send another test-email to the address defined in your trigger, with the example email-body from the start of the section Getting started.
j) The expected result is that you receive an email with the message configured in the MessageTemplate, but where the items in brackets **[like-this]** is replaced by the metadata from the workflow context.

If you've added the optional AnswerSender-job, you may now inactivate the workflow, disconnect the AnswerSender job and at any time connect and use it to send a reply which will help you troubleshoot and debug your workflow.

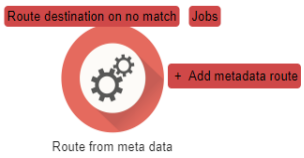
5. The routing

Since we're getting up to speed on how to work with the workflow-builder we may use less details on how to create new jobs and similar actions described above.

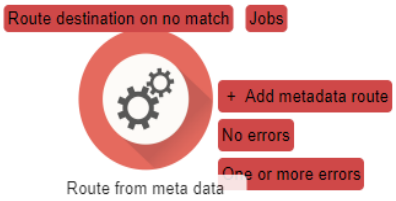
After the DataOperation we will want the workflow to decide if we need an SMS saying everything is fine or if we want to report errors and forward the incoming email.

Add the job RouteFromMetaData, connect DataOperation to the new RouteFromMetaData-job.

The RouteFromMetadata-job has more connections than the average job.



Original



With 2 routes added

Above is the visual example of what we want to do. Add the job, and it will appear in original shape. And then when we add routes, more connections are available.

Open the settings for the job and add two (2) MetaDataRoutes.
Change the name of the added routes. Let one have the name **No errors** and the other **One or more errors**.

In **No errors** set the metadata source to **metadata.error-count** and compare operator to <= and finally the compare value to **0** (the value zero)

For the **One or more errors** configure metadata source to **metadata.error-count** and compare operator to > and finally the compare value to **0** (the value zero)

This job will evaluate the comparison that we've stated and only continue the path of jobs that matches the evaluation.

So, if we send an email where there are no errors (if the email body has the text: **Errors: 0**) then we will continue with jobs that are connected from the **No errors** route.

Or else, if the text in the email has a number higher than 0 (e.g. **Errors 3**) then the workflow will continue with jobs that are connected from the **One or more errors** route.

6. The reporting

To summarize what we've done so far;

- The trigger will react to incoming email if conditions match.
- The DataOperation will let us operate and manipulate data
- The RouteFromMetaData will decide what we should do next.

We need to finish this up with sending messages to the correct groups.

Add the following parts to the workflow:
2 jobs of type SendMessageToGroups
1 job of type ForwardMessageToGroup
3 MessageTemplates
1 Group (blue ring with people-icons)

Connect the 3 jobs to the group.
Open settings and give the group a name (e.g. Staff)

Connect one MessageTemplate to each SendMessageToGroups-job.

Connect the last MessageTemplate to the ForwardMessageToGroup-job

Save the workflow

Go to www.bosbec.io and find your group by clicking the Groups menu alternative.
Add the staff as units/members to the group.
E.g. create a unit with your phone number and email inside that group.

Go back to the workflow builder.

When there are **no errors**, we need the workflow to report that everything was fine, and only send a text-message to the "Staff"-group with the text OK.

Connect the first SendMessageToGroups job from the No errors route.

Open settings for the message template connected to the first SendMessageToGroups job.
Add a new message of type SMS and set the sender name to SystemXYZ set the body to OK.

When there are **one or more errors**, we need the workflow to report that there were errors. And send both a text-message and forward the email to members in the “Staff”-group.

Connect the second SendMessageToGroups job from the One or more errors route.

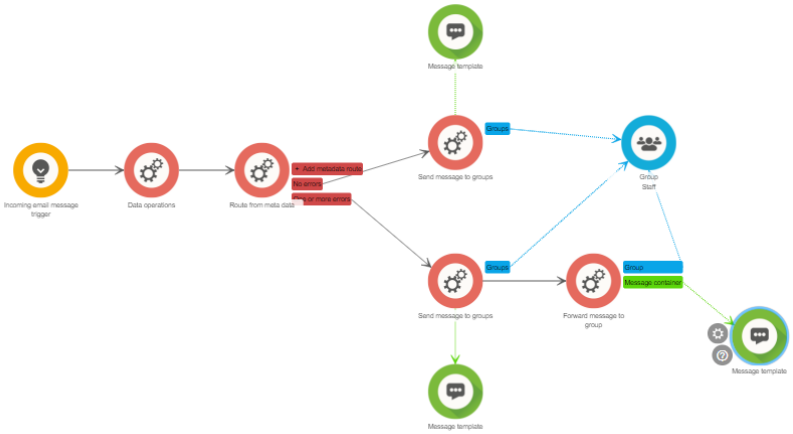
Open settings for the message template connected to the second SendMessageToGroups job.
Add a new message of type SMS and set the sender name to SystemXYZ set the body to
There were [error-count] errors in the last 24h. See email for more details.

Make a connection from the second SendMessageToGroups job to the ForwardMessageToGroup job.

Open settings for the last MessageTemplate and just add a new message of type email.
Configure the from to be noreply@bosbec.io

Save the workflow.
Activate the workflow.
Done!

The workflow should now be complete and look something like the example below.



Test the workflow with different in-data using the template from the beginning of this tutorial.

The reference sections

The following section describes triggers, jobs and such in detail. Some of this information is available directly in the workflow-builder when clicking the question-mark on the properties.

This section of the documentation is not yet completed and you should check for updates to this document at
https://help.bosbec.io/IntroAndReference.pdf

The Triggers

This is a more detailed description of what triggers does and how to use/configure them.

The AccountErrorTrigger

Will start each time an account-error (which is more of a log that ranges from information-messages to explicit error messages).

When an account has processes such as scheduled activities or import processes the system will log warnings that are important for the user to know about. For example, the scheduler will send out a warning each time a scheduled activity is deleted, and the billing-system will send warnings if the account runs out of credits.
This trigger (which is included in the free, AccountErrorNotifier-service that is installed automatically) will let you configure what to do with the warnings.
For example, it can be important to know if an import process failes, and the message could be forwarded to the person responsible for providing the import file.

Notes:
The workflow must be inactive in order to save and update triggers.
Trigger won't react to account-errors while the workflow is inactive.

How to:
Specify any conditions for example set the path pattern and upload a file to the given path and this trigger should start a workflow execution.

The DataLogTrigger

Will start the workflow when a data-log stores a new log item, if that log item matches criteria defined by this trigger.

If there is another process on your account that store information to the DataLog, you may use this trigger to react when a value is stored in the DataLog. The trigger can be configured with the data log document id or key, and with additional criteria for metadata on that same log item.

Notes:
The workflow must be inactive in order to save and update triggers.
Trigger won't react to data-log item updates while the workflow is inactive.

How to:
Specify any conditions.

The ExecutingWorkflowTrigger

Will start the workflow when an API-request to the api2.bosbec.io/workflows/execute is issued.

This trigger is used for almost every service that can be accessed from the services section in www.bosbec.io. The trigger will react to API-request to *api2.bosbec.io/workflows/execute* and to *api2.bosbec.io/workflows/execute-with-parameters.* The latter is preferred, since it will provide the workflow with metadata specified in the request.
The trigger needs no setup. Drag it to the canvas and connect to jobs. The metadata provided by the API-request will be available when the workflow starts.

Notes:
The workflow must be inactive in order to save and update triggers.
Trigger won't execute workflow while workflow is inactive.

How to:
Specify any conditions for example set the path pattern and upload a file to the given path and this trigger should start a workflow execution.

The FileTrigger

Will start the workflow when a file is uploaded to the file service.

The trigger reacts when a file is uploaded to the file service (for example at www.bosbec.io) if the file matches criteria specified for the trigger (e.g. Content type pattern).
The trigger will provide the uploaded file as a file resource.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not react to file uploads while the workflow is inactive.

How to:
Specify any conditions for example set the path pattern and upload a file to the given path and this trigger should start a workflow execution.

The FormAnsweredTrigger

Will start the workflow when a form answer is posted for the selected form.

This trigger will start the workflow and provide a FormAnswer resource to the workflow context, to get information from the form-answer you can combine the trigger with the ExtractData job.
It is possible to configure the trigger to evaluate a metadata statement and by that be more precis in when to initiate a workflow execution.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming form answers while the workflow is inactive.
This trigger will need a form to be connected, or a form template name to be specified.
Provides a Form-Answer resource to the workflow context.

How to:
In the simplest case this trigger will only need you to connect it to a form to be able to start a workflow from an incoming form answer.

The IncomingAppMessageTrigger

Will start the workflow if an incoming app message matches the settings for this trigger.

You can, by creating an IncomingAppMessageTrigger, make a reservation for an arbitrary app-message-receiver.

The sender and receiver should be the Id of an App-User. The keyword is optional and can configure the trigger to react only when the first word in the app-message body matches the keyword.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming messages while the workflow is inactive.
Provides an incoming message to the workflow context.

How to:
Set the **Receiver**, **Keyword** and **Sender** properties, connect the trigger to a job and activate the workflow.

The IncomingEmailMessageTrigger

Will start the workflow if an incoming email matches the settings for this trigger.

You can, by creating an IncomingEmailMessageTrigger, make a reservation for an arbitrary email at the @qlnk.se domain.
For example, you may want to start your workflow when an email is sent to **my_workflow_test_1@qlnk.se**.
To do this it is as simple as putting that email address into the setting for **"Receiver"** in the properties for this trigger.
If you don't care about what email senders may initiate your workflow, you should just leave the **"Sender"** property with a *
Finally, you may also choose to set a keyword as a criteria for when to react and start the workflow. A Keyword is the first word in the incoming email Subject+Body. So if the incoming email is without a subject then the keyword would be the first word in the email body. If there is a subject, then the trigger will try to match the first word in the subject. Setting the **Subject** to * will allow anything as subject and body in the incoming email.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming messages while the workflow is inactive.
Provides an incoming message to the workflow context.

How to:
Set the **Receiver**, **Keyword** and **Sender** properties, connect the trigger to a job and activate the workflow.

The IncomingHttpTrigger

Will start the workflow if a request to the in.bosbec.io endpoint matches the criteria.

This trigger will react to incoming HTTP-requests to the **in.bosbec.io/[YOUR_TOKEN]** where **[YOUR_TOKEN]** is an "AuthenticationToken" or a persisten token that you create in the admin-GUI (located at: *www.bosbec.io*).
The workflow will start with a resource called HttpRequestResource that can be accessed from subsequent jobs in the workflow.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming requests while the workflow is inactive.
Provides an incoming message to the workflow context.

How to:
Set the **Method** to GET / POST or use * to indicate that any method is allowed. Set the **Token** to the id of one of your persistent tokens (created inside www.bosbec.io). Optionally set the resource name.

The IncomingIotMessageTrigger

Will start the workflow if a matching MQTT-message is received.

The trigger will let you configure the incoming topic for the Bosbec MQTT endpoint. Make sure you've got a device registered in order to know what topic to use for communication to and from that device.
Set the receiver to your topic, eg. **1ab45cd8-12efg67h-12i4/my-topic** and the trigger can listen to messages on that topic.
Combine this with a keyword or a sender to be more precis in what to react to.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming messages while the workflow is inactive.
This trigger need device(s) configured.
Provides an incoming message to the workflow context.

How to:
Set the **Reciever** to the topic that you expect messages to be published under. Optionally, set the Sender and Keyword.

The IncomingMessageForGroupTrigger

This trigger is available to support a few scenarios where the sender or receiver is placed as units in a group.

This still requires resevations for incoming channels such as an incoming phone number.

How to:
The recommendation is to use the other triggers, and in cases where till is provided by an installation or a setup from Bosbec Team, advice is to contact support before using or modifying existing setup with this trigger.

IncomingMessageTrigger

This trigger is still available to support old workflows and should not be used in new workflows.

How to:
Don't use, this is still available for legacy purposes.

The IncomingSmsMessageTrigger

Will start the workflow if an incoming sms matches the settings for this trigger.

You can, by creating an IncomingSmsMessageTrigger, make a reservation for a SMS number that can be ordered from support@bosbec.com
For example, you may want to start your workflow when a sms is sent to **12345** (NOTE: this differs from how the email-trigger is configured; you **need** the number to be connected as an incoming number for the bosbec platform).
To do this, you can get in touch with support@bosbec.com and order an incoming number. Or if you already have a number from before, then it is as simple as putting that phonenumber-number into the setting for **"Receiver"** in the properties for this trigger.
If you don't care about what SMS senders may initiate your workflow, you should just leave the **"Sender"** property with a *****
You may also chose to set a keyword as a criteria for when to react and start the workflow.
A Keyword is the first word in the incoming SMS Body.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not catch any incoming messages while the workflow is inactive.
Provides an incoming message to the workflow context.

How to:
Set the **Receiver**, **Keyword** and **Sender** properties, connect the trigger to a job and activate the workflow.

The ProcessTrigger

Will start a workflow if a given event occurs within a process on the account.

This trigger is used to react to the following events:

- **ImportCommandReceived** – When the importer first starts the import
- **ImportFailed** – When an import-process fail to finish with successful result.
- **ImportFinished** – When an import-process has finished.
- **DynamicGroupCreationFinished** – When a putting together a dynamic group is done and members matching the given criteria are members in that group.

Anyone of the bolded words in the listing above are valid values for the EventType.
Apart from setting the EventType you may want to specify metadata key and a regex-pattern to test the **MetaData** of the incoming **event**.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not react to any events while the workflow is inactive.
This trigger needs the event type to be defined.

How to:
Set the **Event type** to one of the items in the list above. Optionally, set the **Meta data key** and **Regex pattern** to have the opportunity to direct different instances, eg. Import 1 triggered in one workflow and import 2 triggered in workflow 2.

The SchedulingTrigger

Will start a workflow on a given schedule.

This trigger is used to start a workflow once or repeat the execution according to a schedule. For example, you may configure one workflow to execute every Monday, while another will execute every hour of every day.
This trigger is useful for scenarios where you have recurring tasks, for example heartbeat and monitoring of some external system or daily import/exports.

Notes:
The workflow must be inactive in order to save and update triggers.
The trigger will not start while the workflow is inactive.

How to:

Set the **Event type** to one of the items in the list above. Optionally, set the **Meta data key** and **Regex pattern** to have the opportunity to direct different instances, eg. Import 1 triggered in one workflow and import 2 triggered in workflow 2.

The jobs

This is a more detailed description of what different jobs can do and how to use/configure them to do different things.

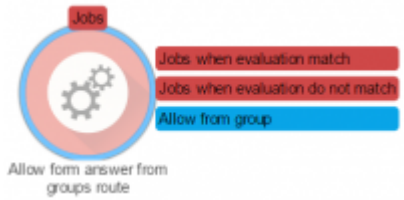
AbandonKeyword

This job is still available to support old workflows and will not do anything in a new workflow.

How to:
Don't use, this is still available for legacy purposes.

AllowFormAnswerFromGroupsRoute

This job is used for evaluating if the unit found in the form-answer is member in any of the given groups. The job then performs routing by making a decision on what next jobs to confinue with.



The job will first make sure that there is a FormAnswer in the WFC. Then that it can match the form answer to a unit.
Then the actual evaluation takes place and the unit is tested whether or not it is a member in any of the given groups. If the unit is a member then the evaluation matches and jobs connected from "Jobs when evaluation match" will be executed, not any other jobs.
Notes:
Jobs connected from the regular jobs-connection will not be executed.
FormAnswerTrigger will provide a FormAnswer in the WFC.

How to:
Make sure that the job is connected after the place that sets the FormAnswer in the workflow (ex. Somewhere in the job-chain after a FormAnsweredTrigger).
Connect next jobs to either one of the two "Jobs when evaluation…" depending on if you want the job to execute when unit was member in any of the groups or not.

AllowSenderFromGroupRoute

This job is used for evaluating if the unit found in the incoming message is member in the given group. The job then performs routing by making a decision on what next jobs to confinue with.

The job will try to find units based on the sender of the incoming message. That implies that there must be an incoming message.
If unit was found in the given group, this job will set the IncomingUnit resource on WFC to the found unit, and also the IncomingGroupMember resource on WFC to the unit as GroupMember in the given group.
Then the actual evaluation takes place and the unit is tested if it is a member in the given group. If the unit is a member then the evaluation matches and jobs connected from “Jobs when evaluation match” will be executed, not any other jobs. If the evaluation result was that the unit is not a member in the given group, then the “Jobs when evaluation do not match” will be executed.
Notes:
Jobs connected from the regular jobs-connection will not be executed.
Any IncomingMessage-trigger will provide the IncomingMessage resource on WFC.
Sets the IncomingGroupMember.
Sets the IncomingUnit.

How to:
Drag the job to the canvas and make sure that is connected after the place that sets the IncomingMessage in the workflow (ex. Somewhere in the after an IncomingMessageTrigger).
Connect next jobs to either one of the two “Jobs when evaluation…” depending on if you want the job to execute when unit was member in group or not.

AllowSenderFromGroups

Similar to AllowSenderFromGroupRoute, but can handle more than one group.

An improved version of the similar job, with the distinction of being able to connect to more than one group.
Notes:
Works in the same way as AllowSenderFromGroupRoute

How to:
See AllowSenderFromGroupRoute and add one or more groups.

AnswerFormQuestion

This job can answer a question in a form if there is already a form-answer created for the incoming unit. The value used to answer the form-question can be for example, set to use some metadata or the IncomingMessage-body.

The job will try to find units based on the sender of the incoming message. That implies that there must be an incoming message.
If unit was found in the given group, this job will set the IncomingUnit resource on WFC to the found unit, and also the IncomingGroupMember resource on WFC to the unit as GroupMember in the given group.
Then the actual evaluation takes place and the unit is tested if it is a member in the given group. If the unit is a member then the evaluation matches and jobs connected from “Jobs when evaluation match” will be executed, not any other jobs. If the evaluation result was that the unit is not a member in the given group, then the “Jobs when evaluation do not match” will be executed.
Notes:
Will abort if required AnswerSource, FormQuestionName or Form is not set. Will also abort if there is no FormAnswers created for the IncomingUnit.
Will publish a form-answered event.

How to:
Set the AnswerSource to a workflow context resource where the job can find the answer (ex. **Incomingmessage.body**). Se the FormQuestionName and make sure that the question-name is the same as the question from the path in the form.
And remember to connect to the Form from this job.

AnswerGroupMember

This job is used to answer (send a message) to the workflow context resource “IncomingGroupMember” which is set in jobs like AllowSenderFromGroupRoute.

The goal of the job is to create an answer/reply to the group-member that is found in the IncomingGroupMember resource on WFC.
The job need the IncomingGroupMember to be available
Notes:
Will abort if required IncomingGroupMember is not set.
Will use defaulting behavior for message.

How to:
Connect to the message template (message to reply with).

AnswerSender

This job is used to answer the sender of the incoming message or “sender” of the form answer.

The goal of the job is to create an answer/reply to the sender of the incoming message or to the “sender” of the form answer.
Prioritizes like this: If there is an IncomingMessage then this will be the sender to answer. If there is no IncomingMessage, but there is a FormAnswer, then answer the “sender” of that FormAnswer.

Incoming message will be set if the execution was started with an IncomingMessageTrigger, and FormAnswer will be available if the FormAnsweredTrigger has started the execution.

Notes:
Will abort if required IncomingMessage or FormAnswer is not set.
Will abort if the "sender"-unit doesn't exist.
Will use defaulting behavior for message.

How to:
Connect to the message template (message to reply with).

AnswerSenderWithFormQuestion

This job is used to answer the sender of the incoming message. And the answer to the sender of the incoming message is fetched from the form's question with the specified name. Can only reply with SmsMessage.

The job will test that there is a sender in the incoming message that and receive the answer/reply. Then make sure that the text message to add as SMS-text-body can be extracted from the form template. Will search for a question with the specified name (`FormQuestionName`) within the form's paths (case insensitive) and use the question's Text property for the message body.

Notes:
Will abort if the "sender"-unit doesn't exist.
Will abort if the "question" cannot be found in the form template.
Can only reply with SMS-message.

How to:
Connect to the form template.
Set the form question name to a question-name from the form's paths.

CalculateSmsAmountByMessageLength

Will calculate how many SMS messages a text (from WFC-resources) would result in. As example, a text with Unicode characters will often result in more SMS-messages than a text with GSM-compatible characters. Job results in setting the calculated value/result in a WFC-resource (ex. metadata.sms-count-result).

The job can be used together with substring- or regex-jobs to control that amount of SMS-message-parts that the workflow will send.
Maybe you want to trim and just give a sample of the text in the SMS message, while sending the full text in the email-body.
Need a resource from WFC in the source and will abort if no such resource is found.
Result is the number of SMS messages that the text would result in.

Notes:

Will abort if the no resource is found in WFC based on the property (Source)
Can only calculate number of SMS-messages, not other types.

How to:
Set the Source to some WFC resource.
Set Destination to where you want the result to end up.

CalculateValueFromMetadata

This job will perform mathematical calculations based a mathematical expression that may contain WFC resources. Will put the result in WFC resource specified in MetadataDestination.

To calculate something based on values in WFC resources this job can be configured with an expression like this:

```
[metadata.counter]+1
```

Will put the result in the specified MetadataDestination (a WFC resource).

Notes:
Will abort if the Expression cannot be parsed/found as WFC resource

How to:
Set the Expression to some expression like the example above and make use of WFC resources.
Set the MetadataDestination to WFC resource where you want the result to go.

ClearWorkflowContext

This job is used to clear parts of the WFC.

Clearing temporary group will remove all members that are read into the WFC as group members.
Clearing Groups will remove all groups from Workflow groups, this means that ExtractData would not find any GroupIds to read group members into the WFC. It also means that there are no Groups that SendMessageToGroups will default back to.
Clearing IncomingUnit will remove the WFC resource IncomingUnit, and jobs that require the IncomingUnit will fail, unless it is set/made available again.
Clearing IncomingGroupMember removes the IncomingGroupMember from WFC, with the similar consequences as removing IncomingUnit.
ClearMetaData is a RegEx expression that will test each key in the WFC metadata and remove if it matches. This means setting ClearMetaData to: **.*** will cause all metadata to be removed.

Notes:

If ClearMetaData is left empty, then the job will NOT clear any metadata.
ClearMetaData is a RegEx-expression

How to:
Set what properties to clear
Remember that ClearMetaData is RegEx-expression and written in text, the others are true/false selected from the list.

CountRecipients

Counts recipients from selected sources.

Will count recipients from the defined sources (RecipientSource). The sources can be Temporary group (TemporaryGroup), the Workflow groups (WfContextGroups) or both (All).
You may set where to put the result of the recipient-counting with MetadataDestination.
Use the true/false properties to set what kind of receivers to count.
Following count-operations are not yet implemented/supported:
??

Notes:
Some of the counting-operations are not yet fully implemented or supported.

How to:
Set where to put the result in the metadata destination.
Select what source of recipients to count.
Select what kind of receivers you need to count

CreateFileFromMetadata

Let you create a file that is uploaded to the file service.

This job will let you create a file, which is then saved to the file service and can be accessed from www.bosbec.io or from other jobs/workflows on the account.
Specify a source where you may use the syntax **Hello {metadata.test}** to concatenate the text **Hello** with the data from workflow context **metadata.test**.

Notes:
The source with format will be the contents of the file, any replacements from workflow context resources such as metadata should be within curly brackets {…}

How to:
Set the **source with format** and the file name, other parts are optional. **Result destination** will be where the job can report status on the file -creation / -upload.

CreateMessageTemplateAsResource

Will let you create a message template "on the fly".

With this job it is possible to create a message template. Specify body for all or configure unique settings for different message types with the settings.

Notes:
Will create a message template on the account.
The created message template id will be set in the metadata 'metadata.created-template-id'

How to:
Set the fields and optionally add unique settings.

CreateNewFormAnswer

This job initializes a new form answer, that doesn't mean the answer to a question. Every time this job executes a new set of answers can be made for a "Unit" that answers questions for a form. Related to the AnswerFormQuestion-job.

So in order to use the AnswerFormQuestion you need to initialize the form answer with this job. You "Create a New Form Answer" with this job. And you "Answer Form Question(s)" with the other job. There are no other settings for this job than the connector to a form-object.

Notes:
This job need to work together with the AnswerFormQuestion-job.
Must be connected to a form-resource.

How to:
Connect to the form (also dragged out on the workflow canvas)

CreateOrUpdateUnits

The purpose of this job is to create a custom formula to "import" units from a text.

This job will create or update units based on what is provided in a WFC resource.
It will find a text in a WFC resource based on what is set as "Search source".
It will try to parse each row in that source as a Unit, and make use of the customizable "Mapping Formula".
Can set the IncomingUnit resource.
"Find distinct for …" makes sure that the job won't find more than one unit with the same email/phone.
Reserved words in the Mapping formula are (reserved word → will map to unit property):
phone, phonenumber → Unit.PhoneNumber

email, emailaddress → Unit.EmailAddress
tags → will map to one (1) tag, use "tags" for each column to map as tag.
Anything else will be mapped as metadata with the same name as the header.
It is possible to use multiple delimiters, but make sure the one in the mapping-formula is present in the list of MappingDelimiters.
Places the resulting units in the temporary group.
MappingDelimiters will default to: `;,`
MappingFormula will default to: `phone;email;firstname;lastname`

Notes:
CreateOnly and UpdateOnly are to be implemented and not yet supported!
Can set the IncomingUnit resource.
Make sure to use the same delimiters in mapping formula as set as delimiters.
Each character in MappingDelimiters will be interpreted as a delimiter.
Places the resulting units in WFC temporary group.

How to:
Only thing that you must set is the **search source**, the other optional settings should be explained in the text above.

CreateToken

Will create an authentication token.

This job can create an authentication token with permissions to execute workflows as the current administrator. The token will be created and put into the metadata defined by the **destination**

Notes:
The result is set to the destination

How to:
Set the **Time to live** and **Destination**.

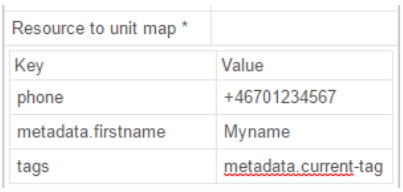
CreateUnitFromData

This job can create units from data. Set the data in the "Resource to unit map" or make use of data in WFC resources. The result of the job can set the Incoming Unit.

The Resource to unit map is the key to making the most of this job. Set a key which will map to a property on a Unit, and set the value direct or to some WFC resource's-value.
It is possible, but not necessary to prefix keywords with "unit" (ex.

unit.phone). Things you can set (the keys):
phone → Will set the phone
email → Will set the Email
metadata.test1 → Will set the metadata with name "test1"
tags → A key that starts with tags will set a tag ex.
tags1 → Will set a tag
tags2 → Will set another tag
Setting the result as incoming unit is optional via the setting with the same name.

Notes:
Can set IncomingUnit
Will not update, only create new unit.



How to:
Example: Setting the "Resource to unit map" like the example on the right, will result in a Unit with
Phone = +46701234567
Metadata.firstname = Myname
Tags = [test1] (if the WFC-metadata "current-tag" has the value "test1")

DataLogItemOperations

This job will let you work with the data log items.

This job is divided into four steps, find, filter, operate and store. First you should find the data log item that you seek to work with, use one or more find steps, and use filter-steps to narrow and specify which of the items you want to work with.
Set one or more operations to for example, update the metadata.
Finally select where to store or delete in the store step.

Notes:
Divided into four steps, find->filter->operate->store.
Will not change the data-log but can operate on the log items for a particular data-log.

How to:
Configure your operation with the four steps.

DataLogSetup

Creates or updates the setup/configuration for a data-log. This job can configure the metadata for a given log(-key). Metadata for the log can be used to make sense of the data or for example, indicate what type of content is in the datalog.

The only thing this job can do is the setup of the DataLog. It cannot store values for the datalog, for that either use DataOperations or StoreData-job.

Notes:
Related to DataOperation (LogData) and the StoreData-job

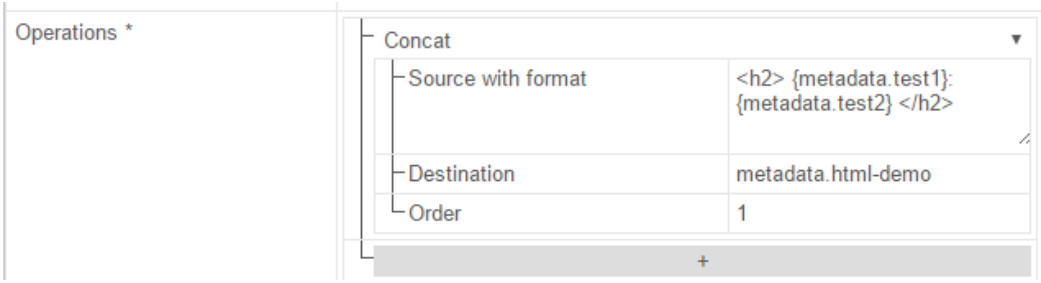
How to:
Set the log key and enter metadata in the table. Use the “Tab” key to tell the workflow-builder that you need another row.

DataOperations

The most flexible job is the DataOperations, since it represents a series of data operations that are bundled together to create a new job assembled from functionality sometimes found in other jobs.

The order of operations to execute can be set with the Order-property in each operation.
What the different data-operations does:

- Get calculated log data
  - Calculates a value based on log items in a datalog.
  - Can be configured to only use values within a given time frame.
  - Sets the result to a WFC resource/destination
  - Can only calculate values from numeric datalog.
- Concat
  - Can concatenate WFC resources.
  - Example:



    This example is from a demo where the input test1 and test2 were two texts in WFC resources and the result of the concat-operation was used as body in a HTML-email.
    With indata: *test1 = Workflow, test2 = Hello!* **The result was:** “*<h2>Workflow: Hello!</h2>*”
- Calculate data
  - Works like the CalculateValueFromMetadata job does.
- Extract value regex
  - Uses the pattern to find a text in the source and puts the result in destination
- Get last log data
  - Gets the most recent value from the datalog with the given key.
- Insert regex
  - Uses the pattern to find a text in the source, and with the options; insert the given value in the source and place the result in destination.
- Concat group member data
  - Data sources with format can be used in the same way as the Concat-operation does with the “Source with format”-property.

    - Separator can be a text or a single character that separates each “row” (each group-member)
    - An example could be: {phone}: {firstname}, {lastname}
      And would result in something like this:
      +46701234567: John, Doe
      +46701234568: Jane, Doe
    - Reserved words are: {id}/{unitid},{email},{phone},{createdon} everything else will be interpreted as metadata (ex. {test1} would refer to groupmember.metadata.test1)
- Log data
    - Will log data to the datalog with the given key.
- Remove regex
    - Removes data from the source, based on the pattern and options and put the result in destination.
- Replace regex
    - Replaces data in the source, based on the pattern and options and put the result in destination.
- Substring
    - Extracts a part of the data in source based on the zero-based index, with a max-length.
    - **Ex: indata:** *This is the text in metadata1* with substring *startIndex: 0* **Will result in:** “*Test I*”
- Set data
    - Will set the destination to the result of the WFC-resource in source
    - Ex. source **[datetime()]** will result in the current date and time
    - More examples in the ***WFC-resources tips&trix***-section

Notes:
This job is more useful once you’ve read about it in tip&trix-section.

How to:
Press the + button and select what operation to add.
Set the order of operations with the Order-field.
Read about what operation to use in the description above.

DeleteOperations

This job can delete resources, such as groups.

The job can delete resources, such as groups.

Notes:
Deleted resources may be impossible to retrieve.

How to:
Set the delete operation.

DeleteUnitsFromAccount

This job will delete units from the account.

The job can be used to delete units form the account.

Notes:
Deleted resources may be impossible to retrieve.

How to:
Delete units based on what is in workflow context right now.

ExecuteOrPostPone

This job either executes the next job, or if the allowed days and hours doesn't match the current time, then this job will postpone the execution of the next job until days and hours are allowed again.

Set the allowed days and set between what hours it is allowed for the next jobs to be executed.
Note that you cannot have different times on different days.
Postpones next job with the Delay-time-feature that is present on every job.

Notes:
You cannot have different times on different days.
For now, this job uses the Delay-time-property, and it may change between each time you view the workflow.

How to:
Set the allowed days and set between what hours it is allowed for the next jobs to be executed.

ExecuteWorkflow

Allow one workflow to execute another workflow.

Sometimes it makes sense to have a part of a workflow reused in different situations. For example, the workflows behind the services are shared as a public workflow that may be executed by any administrator in the system.
The benefit from this is that we can make sure that we have only one place to update the functionality and all the services will automatically use the new functionality.
The job will let you search for workflows on your account or try to match publicly available workflows and will execute the found workflow as the current administrator.

Notes:
You can specify what trigger to execute in the workflow to execute.
Metadata are automatically passed to the new workflow but may be ignored with setting.

Be careful when allowing recursive executions, since this may cause loops that execute many times before you're able to stop it (by inactivating the workflow).

How to:
Configure the workflow to execute by providing id or name, optionally set trigger names to execute in that workflow (only ExecuteWorkflow-triggers will be executed)

ExportDataLog

Exports the datalog(s) to WFC resource with a custom set of rules for the export.

The job will export datalog to the workflow context.In order to get an export, you need to configure the settings. And sample settings can be found in the Shortcuts-menu in the workflow-builder.

Notes:
This job needs data in the data-log.

How to:
Connect the job and configure what keys in the data-log to export.

ExportFormAnswersToWorkflowContext

Exports form answers to WFC resource with a custom set of rules for the export.

This job exports form answers in a customizable way. In order to get a custom export, you need to configure with settings for the columns.
There are different ways to configure the settings and listed below are a few variations:

- ColumnFormatTemplate can be set to @**(answer)** which will configures that column be just the answer.

Notes:
This job needs form answers to export

How to:

ExtractData

This job is built to extract data from events or complex resources, for example when import is finished, when a dynamic group creation is finished. Another example is to move group members from the workflow groups to the WFC temporary group.

There are three different features available at this moment (and you can set them either by the exact name or the number):

1. `GetGroupFromImportFinishedProcessEvent`
    a. Gets the group(s) from the ImportFinished-event and adds the group to the workflow groups (NOTE: not to the temporary group)
2. `MoveWorkflowGroupMembersToTemporaryGroup`

   a. Reads the group-members from the workflow groups, removes those groups from the current execution and adds all members from the workflow groups to the WFC temporary group instead. (This is done so that you may apply WFC temporary group-filters)
3. `GetGroupFromDynamicGroupCreationFinishedEvent`
   a. Just like (1), this extraction finds out what group was just compiled and adds that group(s) to the workflow groups (NOTE: not to the temporary group)

Notes:
Must manually enter one of the "extractions".
For some of the exctactions (1 & 3) it is required that the exection is triggered by a process-trigger.

How to:
Enter one of the actions/extractions and the job will do that.

FilterReceiversFromWorkFlowContext

This jobs filters receiver(s) in the WFC temporary group.

You may choose to keep or remove the receivers that match the filter with the **Keep filtered** property.
Explanation of what the filters does

- ShouldRemoveDeliveredAppMessageReceivers
  - An app-message is delivered when the app-users opens the Bosbec app (more precise it is when the app calls the What-is-new function in the app-API)
- ShouldRemoveMessageReceiversWithoutAppUser
  - Removes every unit that doesn't have the AttachedAppUser set
- ShouldRemoveReceiversWhoCompletedFromInCurrentProcess
  - Removes the receivers that has submitted a form answer.
  - "in current process" means that the form must be generated in the same process as currently executing.
- ShouldRemoveReceiversWithMetadata
  - Will remove receivers that match the metadata defined in the metadata-key and -value properties.
- ShouldRemoveReceiversWhoMatchIncomingSender
  - Requires an incoming message.
  - Removes receivers that matches the incoming message's sender

Notes:
This job is very old and will most likely be replaced in the future. Signs of this is the inconsistent naming ShouldRemove… KeepFiltered.

How to:

Just set the properties and remember to set KeepFiltered dependeing on whether or not you want to keep the receivers that match the filters.

FilterTemporaryGroupWithGroups

Use groups to keep or to remove members from the temporary group

Select one or more groups to use as a filter, either to match and keep or match and remove units in the workflow context temporary group.

Notes:
See information about the TemporaryGroup.

How to:
Drag and connect group(s) to use as filter.

FilterWorkFlowContext

The purpose of this job is to filter contents of the workflow context. There is only one implemented filter and that is the TemporaryGroupMetaDataFilter.

Since Temporary Group Metadata filter is the only implementation for the workflow context filters in this job, this is an explanation of what that filter does.
The metadata key must be the actual key (may not include metadata-part "**metadata.**my-key")
The CompareValueSource will try to find a value in WFC-resource and make the comparison with the resource instead of a fixes value (as in the case with CompareValue). Will default to comparing with the CompareValue if no resource value was found to compare with.

The comparison will first try to compare as number, if it isn't numbers the job will try to compare as DateTime, and finally if it isn't DateTime either the comparison will interpret values as string/text and make the comparison.

To understand how the comparison works check out the section on **MetaData-comparison**.

Notes:
See Metadata-comparison section for more details.

How to:
Select the filter (only TemporaryGroupMetadataFilter available).
Set the properties to compare the metadata.

FindCounters

This job finds synchronized counters and add as resource to the workflow context.

Finds a counter matching the **CounterKey** and **CounterName** and add that counter as a resource in the current workflow context.

Notes:
See synchronized counters for more information.

How to:
Set the key and name and set a resource-name for the found resource.

FindGeneratedFormAsResource

This job finds a form that has already been generated and add it as resource to the workflow context.

A form template can be used to generate an actual instance of a form that a user may respond to. A form can be generated with the GenerateFormAsResource-job.
This job will find a generated form, not a template, and add the found generated form(s) to the workflow context.

Notes:
See information about forms for more information.
See GenerateFormAsResource -job description

How to:
Set the search phrase and resource name.

FindGroups

This job finds groups and add the found groups to workflow context groups.

The search text source is used as search-phrase when finding groups. It may be a list of group-ids or a name of groups. Can use RegEx as search. Case-insensitive search.
Use the AllowedMaximumGroupsFound to control how many groups that may match the search. For example, specify that if should max find 1 group, and the finding resulted in 2 groups, then this job will NOT add groups to WFC groups and the result will be **false**.

Notes:
Can add groups to WFC groups.
Can use RegEx as search.
Case-insensitive search

How to:
Set the amount of groups that the job may find (or set to < 1 to allow any number of found groups)
Use WFC-resource or a specified text to search for groups in the SearchTextSource.

FindMessageTemplateAsResource

This job finds a message-template and add it as resource to the workflow context.

This job finds message-templates and adds as resource to the workflow context. This can be useful when you make different message-templates for different languages and want the workflow to dynamically load the correct message template for the given language. (In such a case search using the metadata from the message-template).

Notes:
See information about message-templates and resources.

How to:
Set the search phrase and resource name.

FindOrCreateAppUserResource

This job will try to find an app-users from the data provided. If it cannot be found the job will create a new one with the provided data.

The job will find app users or create a new one with the data provided.
When the job successfully creates an app-user, the user will be stored as resource in the current workflow context.

Notes:
See information about app-users and resources.

How to:
Set the find phrase to find app users and set the other properties to provide data used to create app users with.

FindOrCreateUnits

This job will first try to find units, if it cannot find units, it will create a unit matching what was searched for.

Will first try to get the value from WFC-resource in SearchSource.
The value found in SearchSource will be used to try to find units.
Depending on the property PreventCreatingUnits this job can be configured to create new unit if finding existing units isn't successful.
If units are found, then the FindDistinctForX.. comes into play. With these properties set it is possible to make sure that only one unit with each phone number/email is returned as the find-result.
If SingleFoundUnitAsIncomingUnit is set to true AND the find-part of this job resulted in a single unit found, then this job will set the IncomingUnit resource.
It is possible to connect a Group to this job. This will make the found or created unit be saved to that group, instead of to temporary-group in the workflow context.

Notes:
Can set Incoming Unit.

How to:
Set FindDistinctForX.. properties if you need to find only one unit.

FindUnitAsResource

This job will find units and add the found unit(s) as resource.

By default this job will find units using the same search syntax as you would use in the graphical user interface at www.bosbec.io but it is possible to turn off this search using the DenyBetaFeature-property.
Setting the pagination properties will allow this job to retrieve just a part of the result, for example the 2:nd page where each page has a size of 10 items.

Notes:
See resources for more information.

How to:
Set search phrase and resource name.

FindUnitsById

This job will find units using the unit-id.

Set a value for the Id source (you can get the ids from within the workflow context). Set the resource name, and the job will put the found units as a resource with the given name in the current workflow context.

Notes:
See resources.

How to:
Set idSource and resourceName.

ForEachResourceStart & ForEachResourceStop

This is two jobs that will help you when you need to repeat a series of jobs for each item in a resource. For example, you may use this to iterate through every unit in a unit resource.

These jobs work together, the ForEachResourceStart-job require you to specify what resource to repeat over and define a temporary name for the single-resource that was picked out of the multi-resource.
Connect the jobs to repeat over from the ForEachJobs, and then connect the JobsAfterComplete (the jobs to continue with after the repeating is completed).

At the end of the series of jobs to repeat you need to connect the ForEachResourceStop, so that the workflow can understand that it is time to go back and repeat from the ForEachResourceStart-job.

Notes:
Lookup information about resources and how they can be used and addressed.

How to:
Connect jobs to repeat to the ForEachJobs connection, and the jobs after looping/repeating is done to the JobsAfterComplete.

FormAnswerRemoteHttpRequest

This job uses the form answer as data when posting HTTP-requests.

For more detailed description about how to configure the http-request look at the RemoteHttpRequest-job.
When using GET http-method the form-answer can be used in the URL.
When using POST http-method the form-answer can be used in the PostTemplate.
To use an answer in the request user brackets with the question name within ex. **[question1]**
If you use Approve answer type you may return the alternative id, value or comment ex. **[question1.alternative-id]**, **[question1.alternative-value]** or **[question1.comment]**
Also **[unit.id]** will be replaced with the user-id of the form-answer (which usually is a unit id, but may be app-user id in some cases.

Notes:
Will execute and can continue after remote http-request is completed.
Similar to RemoteHttpRequest-job.
Need WFC-resource FormAnswer to work.

How to:
Use in same way as RemoteHttpRequest, but can make use of form-answer-data with question-names in brackets [question2]

ForwardMessageToEveryone

Use this job to forward contents of incoming message to everyone in WFC groups and WFC temporary group.

Can be used without setting message template, but will in that case set default values to all message types (that should be Normal prio and settings defined in Account+Admin).
It is however possible to set a custom message template and configure more details about what to send. The job replaces the message body of the parts in the message template with the information to forward from the incoming message.
It is possible to configure a “Predefined Header”, that would result in AppSenderId (for app if valid id) SMS sender name for SMS text messages and the from-address when sending email parts. It has no effect on Iot-messages.

ShouldIncludeHeaderInMessage has effect when incoming messasge is email, and means that the email subject will be included in the beginning of the forwarded text body.

Notes:
Need IncomingMessage.
No need to set message template, but will then set default values for every message type.

How to:
In the most simple way, just connect the job somewhere after an incoming message trigger and make sure that there are members in WFC-groups or temporary group.

ForwardMessageToGroup

Use this job to forward contents of incoming message to a group.

Most of the information about ForwardMessageToEveryone applies to this job as well, but this job need a group to send to and will not default to everyone in WFC.

Notes:
Need IncomingMessage.
Need a group to send to, will not default to WFC-groups.

How to:
Using this job is pretty much the same as ForwardMessageToEveryone, except you need a specified group in this case.

GenerateApproveFileForm

Use this job to generate a predefined form to collect approval for a file (ex. invoice pdf)

The job creates a temporary form template and adds it to the workflow context, so that a SendMessage-job may generate a form-link for each receiver.
It is possible to set the different texts in the form that presents a file, some information and two accept/reject alternatives to the respondent.
It is also possible to add hidden data to the form and make use of that hidden data when the form-answer is incoming (via the form-answer-trigger)
Most properties can get data from WFC-resources.

Notes:
Need an incoming file in WFC.
See files in workflow context.

How to:
Set the information for the form and need files in **workflow context files.**

GenerateFormAsResource

Use this job to generate a form during the workflow execution, either generate one form without connecting it to a unit or generate one per unit and set the form-id as metadata on that unit.

The job can generate a form and set the generated form as a resource.
To do this, set the resource name and connect to a form template.

To generate a form for a unit you also need to set the Settings to a GenerateForUnitSetting where the unit-source is defined (for example a unit resource).

UnitSource can be **incomingunit**, **temporarygroup**, or the name of a unit resource.

GeneratedFormIdDestination will be a metadata-destination on the unit if the configured setting is a GenerateFormForUnitSetting. Otherwise it will be stored in a workflow context destination.

Most often this job is used in a loop as a way to generated form-ids for units while at the same time performing other operations on the same units one-at-a-time.

Notes:
See resources, and forms.

How to:
Set the configuration for the job.

GetContentFromFile

This job can read a file from the file service.

Set the **encoding** of the file, set the **identifier** (the file-id or a source within workflow context). Set the **destination** where to put the result. And connect next jobs to the **JobsAfterDoneReading** in order to continue the workflow after the file content is read. It is also possible to connect to the **Jobs**-connection, but then the file-contents cannot be expected to be found in the workflow context.

Notes:
This job has a special Job-connection "JobsAfterDoneReading" that is expected to be used in order to get expected result from the workflow.
Using FileService.

How to:
Set the encoding, identifier and destination and connect JobsAfterDoneReading.

GetFileFromFileService

This job gets a file from file service and put it into workflow context files.

Set the file identifier and the file will be put into the workflow context.
To read data from file see GetContenFromFile.
After getting the file into WFC, it is possible to send the file with a email message.

Notes:
Using FileService.
Will not do anything with the file, just retrieve it from file service.

How to:
Set the file identifier.

GetMetaData

This job manipulates data on the WFC. It has a few more features than SetDataOperation in DataOperations but can basically do the same thing. This is the only job that can get app-user properties

Have a look at the section that describes what is possible to do with WFC resources and to that list the GetMetaData-job can also do this with app-message "*incomingmessage.sender*":

- **.displayname** – gets the app-user display name
- **.avatar** – gets the avatar of the app-user.
- **.email** – gets the email.
- **.phone** – gets the phone number.
- **.firstname** – gets the first name.
- **.lastname** – gets the last name.
- **.app-user-id** – gets the app users id.

The GetMetadata job can also get the FormAnswer-respondent, that is the unit that answered the form answer. Get this by using **formanswer.answerer** or **formanswer.respondent**
If GetMetaData gets the FormAnswer-respondent it will be added to the WFC temporary group

Finally GetMetaData can also get results from FormAnswers, by using the name of the question (ex. question1) we can write: **formanswer.question.question1** and get the result into a WFC-resource.

Notes:
Need an incoming app message to get properties of incoming app message sender.
Need an incoming form answer to work with the form-answers.
If getting form-answer-respondent, the job will add it to WFC temporary group.
For more details on WFC-resources see the section that describes it in more details.

How to:
Write a source, like the default WFC-resources available (almost) everywhere or something unique to this job (like **formanswer.question.question1**) and set a WFC-resource as destination and the job will put the result in the destination.

GetPropertyOfMetaData

The idea behind this job is to get properties of a WFC-resouce. Example is to find out how many items (separated by comma) is in a string.
That example is currently the only implementation of properties you can use.

Currently the only available operation for this job is the `GetNumberOfStringsInCommaSeparatedString (1)` and what it does is split the given metadata (or other WFC-resource value) by comma ',' and count the result.

Notes:
There is only one operation available for now.

How to:
Set the source and destination to the WFC-resources you like and set Operation to *GetNumberOfStringsInCommaSeparatedString*.

GetUnitsFromAccount

This job will find units from your account, from a number of different find criteria/options. Can be used to set the incoming unit to the result of find operation (if the result is only one unit found).
The found units can be put in a group or in WFC temporary group.

If you set the UseWorkflowContextMetaDataValue to true, it means that phone and email criteria can get their values from WFC-resources, Ex. **metadata.phone-number** otherwise it will simply try to find a phone with the number metadata.phone-number and that won't happen, since it is not a valid phone number.
The same goes for metadata criteria; if using values from WFC the job will try to find values first, otherwise it will simply try to find what was put in as text.
The with- and without-tags will treat every comma, semi-colon and space as a separator between tags that add to the find criteria.

If a group is set, then the job will put found units into that group instead of adding them to WFC temporary group.

If the property SingleFoundUnitAsIncomingUnit is set to true, and the job found exactly one (1) unit. Then the job will set the found unit as the incoming unit.

Notes:
Can set the IncomingUnit.
Will not create units, only find existing units.

How to:

Set the criteria for phone/email, tags and metadata.
Toggle if a single found unit will set the incoming unit.
Toggle whether or not to use WFC-resources / metadata to replace the critera-parts.

GetUsersFromIncomingMessage

This job gets Units (users is a legacy terminology for unit-like concept) from the incoming message.

This job has settings to define where to look for units in an incoming message (called the FindUserLocation).
The job was originally designed to find units from incoming emails but will use the same FindUserLocation alternatives with similar locations in the different message types.

- None - Will not find units
- Receiver fiends - "To/receiver" for app and sms messages, to + cc + bcc for emails
- Sender - From in every message type
- Header - Subject in emails, and none in the other types
- Body - The message body in all types
- Attachments - Not implemented

Notes:
Requires an incoming message.
Will create units if no match is found.
Will set the incoming unit if a single unit is the result of the create/find units.
FindUserLocation alternative "Attachment" is not implemented.

How to:
Connect after an incoming message trigger (any kind of incoming message trigger).
Set the alternatives for where to look for units and whether or not to ignore setting a single found unit as incoming unit.

GroupAverageCalculations

The job calculates average time until "now" (current time) based on a metadata key on group members in a given group.

There is a story behind this job and that story is that it was designed to help out engineering students at the local university in their experiment where BosbecWE calculated the average time that group members were sitting down. Their case where inspired by the possibilities of new devices and sensors in combination with the health-impact of modern workplaces with a lot of workers sitting down most of their time.

The job can, for now, only calculates the average time between system current time and a time found in group members metadata.

As example the engineering students' case;
A group of 4 chairs have sensors that indicate if someone is sitting on the chair or not.

The sensors (or chairs) are represented by units in a group.
Each time a sensor on a chair, registers that someone is sitting on that chair it will report this to the workflow and each time that person stand up, the sensor will report that to the workflow.
When the workflow registers that one of the sensors is in sit-state it will set the current time in a metadata for that group member.
With that information, this job can calculate an average sit-time for an entire set of chair-sensors.

The result from the job is in milliseconds.

Notes:
The result from the job is the average milliseconds.
Can only use the same metadata from every group member.

How to:
Set the CalcAgerageTimeUntilNow to true.
Set UsingMetadataKeyFromMembers to the metadata key that holds a date time for each group member.
Set AverageOnlyForMembersWithKey to true, to make sure that only group members that actually has the given key will be used when calculating average.
Connect the group to find group members in.
Set the ResultDestination to where you want the result.

GroupOperations

This is the job that should be used when operating on a group.

The job has four parts, Find, Filter, Operate and Store.

The idea is to get groups, optionally filter out and select a few of them, perform an operation on each of the groups after the filter.
And finally perform a storage-operations such as saving to account / workflow or deleting the groups.

Operations can be: RemoveMembers, UpdateGroupData

Notes:
See groups.

How to:
Configure find-step, optionally add a filter and operation then select what store-step to take.

HandleIncomingQuotaEmail

This is a specific job developed for one customer, but available for other customers as well. However, to make the most of this job you should get in touch with the support or developers. The job is a predecessor to the concepts in SynchronizedCounter-job.

The job is designed to be put into the standard email-to-sms workflow, and when an incoming email arrives, this job will count/calculate how many sms-messages the email would result in. Taking into account how long the text is and how many receivers would get the message.
Each email-address gets an amount (let's say 100 sms messages to send) and each time the workflow executed this job it will increase a counter for each email with the number of sms-messages used. For example, the email has a text that would result in 3 sms-parts (further reading about sms-parts see customer-wiki-pages on mobilresponse freshdesk), and 10 receivers to forward, would result in counter for that sender-email increases with 30.

The job can send warning-emails when counter exceeds certain numbers.

Notes:
Get in touch with support or dev-team for further instructions with this job.

How to:
Get in touch with support or dev-team for further instructions with this job.

InactivateTrigger

This job inactivates the connected trigger.

The only thing this job does is inactivates the trigger that it connects to.

Notes:
Will inactivate the connected trigger.

How to:
Connect to a trigger.

InactivateWorkflow

This job inactivates the current workflow.

This job will inactivate the current workflow.

Notes:
Inactivates the workflow, and no other job can at this time activate a workflow again, it has to be manually activated after the execution of this job.

How to:
Just add and connect this job.

IncrementalRouteEvaluator

This job is considered obsolete and will be removed in the future.

Each time the job executes it will update itself and increase the number of executions. When a certain number of executions matches the evaluation operator it will continue with the jobs called “JobsWhenEvaluationMatch” otherwise it will continue with the “JobsWhenEvaluationDoNotMatch”.

Notes:
Obsolete, do not use this job.
Same functionality can be built in better ways using SynchronizedCounter.

How to:
-

Intersection

This job is considered obsolete and will be removed in the future. It will work, but it is not recommended to use this job. Use combination of ExecutionRules, Routing-jobs and RegEx-jobs instead.

The job will evaluate a condition, and then execute the job in “MatchJobId” or “NoMatchJobId”.

Notes:
Obsolete, do not use this job.
Cannot connect to next jobs without entering their Id manually.

How to:
Set a condition.
Copy the Id from a you want to continue with and paste in the textbox for “MatchJobId” or “NoMatchJobId”.

LogToProcess

This job adds a log message to the process. (The process that is currently executing, can be found in admin → workflows → workflow execution contexts)

Each time a process executes, for example a workflow is executed we have a process running. And a process can be considered a log of events that most of the time will be human readable and understandable information.

This job can add an extra log-message (great for debugging purposes).
You may use a WFC-resource / metadata as Log text source.

Notes:
Will only log to the current processes process-event.

How to:
Set a source or a fixed text to log.
Read the log in admin → workflows → workflow execution contexts.

NumberLookup

This will do a simple Lookup for each receiver in WFC temporary group that is a SMS-receiver and add the `sys-operator-alias` *metadata to those units.*

The reason for this job is to determine what is the mobile operator for a receiver so that it can be routed on in the following jobs.

To use the lookup-service you should get in touch with our support-team to make sure that your need for lookups will work out with this implementation.

Cases when this is useful can be when you have a special request to route some SMS-traffic via a certain provider and let other traffic go to what the system decides is the best match for the moment.

Notes:
Will only set a metadata on the WFC-group members (or on the units if persisted)
Can persist the lookup-data to the unit on the account.

How to:
Set PersistLookupResults to true if you want to save the results to units on the account, and false to make the workflow keep the metadata during the current execution and then discard it.

ParseAndFormat

This job will be rebuilt or marked obsolete.

The original case for this job is to take a WFC-resource and be very generous with parsing and output a result that is more strictly defined, ex. to facilitate getting date-values from one format to another.

Notes:
Obsolete, it's recommended not to use.

How to:
Get in touch with support- or dev-team if you need this functionality and they can help you with other ways to do this.

ParseCsvToResource

This job will parse CSV-data and put the result into a workflow context resource.

The **CsvSource** should point to a source within the WFC where the CSV-data is stored, could be a metadata after a file was read by GetContentFromFile-job.

FirstRowIsHeaders indicate whether or not the first row in the CSV-data is a header-row or not.
DestinationResourceName is what to call the Csv-resource.

Notes:
See CSV-resources.
Job will set metadata with success (true/false) result.

How to:
Add the job and configure the parameters.

ParseFormDataToResource

This job will parse HTTP-FORM-data to resource.

Will try to parse form-data from a WFC-source and put the resulting FormData as a resource in the WFC.

Notes:
See form-data-resources.

How to:
Set the **FormDataSource** and **DestinationResourceName**.

ParseJsonToResource

This job will parse JSON -data to resource.

Will try to parse JSON-data from a WFC-source and put the resulting **JsonResource** as a resource in the WFC.

Notes:
See JsonResources.
Reports job result to wfc-metadata.

How to:
Set the **JsonSource** and **DestinationResourceName**.

ParseXmlToResource

This job will parse xml-data to resource.

Will try to parse xml-data from a WFC-source and put the resulting XmlResource as a resource in the WFC.

Notes:
See XmlResources.
Reports job result to wfc-metadata.

How to:
Set the **XmlSource** and **DestinationResourceName**.

PlaceCustomBillingOrder

This job can be used to put custom billing-orders onto the invoice of the current account, this product must be defined together with Bosbec.

The job will create a custom billing-order and if that order is paid you may choose to have a set of jobs for that case and another if the payment was unsuccessful. The payment of a custom billing order will send a message to the Bosbec billing-system, which will determine if the current account can pay for that order and charge it to the account's payment method.

Notes:
Custom products and specifications must be a collaboration with Bosbec, contact support for more information.

How to:
First get in touch with support- or dev-team and then set the order specification and configure the rest of the job.

Placeholder

This job doesn't do anything but act as a placeholder so that you may build a workflow and later on replace this job with another.

The job just acts as a placeholder and will not do anything but let the next job(s) be executed.

Notes:
Just a placeholder.

How to:
-

RegExFilter

The regex-job is also available in DataOperations as separate operations for the different types of regex-filters.
What it does is simply execute a regex on a text and make use of options to decide on whether or not to insert, extract, replace… before, after…

There are 4 types of filter operations:

- Extract value - Will get a value based on the settings for that filter and put result in the destination.
- Insert value – Will insert a value form dynamic source into the source depending on the settings for the filter and put result in destination.
- Remove – Will remove text from the source and put result in the destination.
- Replace – Will replace text in source with text from dynamic source and put result in destination.

Notes:
Most of the time it is easier to use the regex-operations in DataOperation-job instead of this.
Dynamic source must be used when inserting or replacing.

How to:
Select what filter operation and configure it according to information above.
Set source/destination and maybe the dynamic source.

RemoteHttpRequest

This job queues a HTTP-request to MR-remote-http service. It is possible to configure POST or GET requests and chose to continue direct or after response.

Set the url to call (it is recommended to test out the request against a dummy-server to confirm that the request is what you've expected).
If you want to do a Http-request with the POST method, then you should put your data in PostTemplate, otherwise the GET-request will use data from the Url-field.
HttpMethod can be either POST or GET at the moment.
It is possible to set custom HTTP-headers.

If you want to wait and continue the execution after the request is done, then connect next job(s) from the "Jobs to execute after response" instead of the regular "jobs"-connector.
In this case, were you continue execution after response you can make use of the ResponseSettings, and set a WFC-resource to put the status-code, response body and headers.

Notes:
Inactivates the workflow, and no other job can at this time activate a workflow again, it has to be manually activated after the execution of this job.

How to:
Just

RemoveFormAnswer

Removes form answers for a form within a given period.

Will remove every form answer within the given time period (which defaults to before MR was created until a very distant future)

It is not possible to only delete answers that are completed or not completed.

Notes:
Removes form answers for a FormTemplate and there is no way to get the results back.
The times, from and to can use WFC-resources.

How to:
Connect to the FormTemplate
(Optionally) set the start and end dates.

RemoveGroupMembers

This job removes group members from a connected group or from found groups.

If UseGroupFromGroupId is true the job will use the connected group.
If UseGroupFromFindGroupSource is true then the job will try to find one or more groups by id or name. To search for multiple groups use semi-colon as separator ex. **Group1;group_2**

The different alternatives that starts with “Remove using” will get unit-ids from either a WFC-resource (Metadata source) or the temporary group or the incoming unit.
And then remove just those units from the groups.

If ClearGroups is set to true, then all members will be cleared in the given groups.

Notes:
Removes group members by unit id, so two different units with ex. same phone could still result in just removing one of them.

How to:
Either connect one specific group or search for groups to remove from.
Set what to use as template when removing.
Set clear group if you want to remove all members from the given groups.

RequestExport

This job can send message to the exporter to export other things than the formAnswer (to export FormAnswers use the RequestExportFromAnswers-job)

This job will result in an export to a (csv-)file sent via email.

Notes:
Only need you to enter email to export to, other values has defaults.

How to:
Set the email to export to.
Set column-separator or it will default to semi-colon. ;
Set text-wrapper or it will default to double quotes. “
(Optionally) set to and from times.
Select what you want to export, units, message history, group members (require that you’ve connected a group)

RequestExportFormAnswers

This job can export form-answers, actually it sends a message to the MR-export service.

For more details and in-depth explanation of what is possible with this job have a look at the section on exporting with the exporter.

Notes:
Save the job once to let the system set it up with default settings.

How to:
Drag the job to the canvas and save it once, reload the workflow.
This causes the system to set default values for this job, and it is easier to work from that.
Connect to the form you want to export and change settings (see exporting with exporter section)

RequestFileServiceImport

This job will start an import-process for a file from the file service.

This import is a simplified version that will allow the “happy-path”-version of imports. Just set the file-id from the file-service.

Notes:
See section on Importing units.

How to:
Not many options and ways to configure this job, set the file id.

RequestForEach

This job will create a HTTP-request for each of the units in the given resource.

The request template can be configured as a HTTP-POST or a HTTP-GET request.
This job has been used when a set of units should be posted to another system with an API accepting HTTP-requests.

Expressions in this job uses the @{…} syntax.

Notes:
The syntax for reaching metadata and such resources may differ from standard.
Http-messages will show up in message history.

How to:

Configure a RequestTemplate and make sure there are units in the resource.

RequestFormRespondentsInBatch

Will get the form-respondents from a sent batch.

Need a batch-id from when the message with form was sent.
Can get both the units that did respond to the form and the ones that didn't.

Notes:
Need a batch-id, and a batch with a form.

How to:
Configure the settings and connect the next jobs to the JobsAfterRetrieve.

RequestImport

This job sends a message to MR-import service and requests an import with the given settings.

This is a complex yet very useful job. To get the most out of this job refer to the section on importing units.

This job is mostly used in combination with this job being scheduled ex. via a Scheduling trigger. And when the import is finished a ProcessTrigger can continue execution.

Notes:
See section on Importing units with importer.
Often used together with Scheduling triggers and Process triggers.

How to:
Set settings based on what you need to import units.
More detailed information can be found in the section on importing units with the importer

RequestRetrieveFile

Will get a file from a FTP-server and put in FileService.

Configure the FTP-settings for the job, and the job will retrieve file from the FTP-server.
Use JobsAfterRetrieve to continue execution after the file was retrieved.

Notes:
Will add file to file service.

Will add file to WFC.

How to:
Configure the FTP-settings and the file will be brought to the file service. Make sure you connect jobs to JobsAfterRetrieve if you need to use the file within the same WFC.

RequestRetrieveUnitsFromBatch

Will get units from a previously sent batch.

Provide a batch-id in the source and the job will try to get units from the batch that was sent.
The result will be put in a unit resource.
Use the JobsAfterRetrieveUnitsFromBatch to connect to subsequent jobs.

Notes:
Need a batch-id, which can be found I WFC-metadata after a SendMessage-job has executed.

How to:
Configure the settings and connect the next jobs to the JobsAfterRetrieveUnitsFromBatch.

RequestUploadFile

This job will try to upload a file to an FTP-server.

Set the FTP-settings and a name for the uploaded file, and this job will get a file from file-service and upload a file and then continue the workflow-execution with the JobsAfterUpload

Notes:
The job has a special jobs-connector to use when continue executing after upload is done.

How to:
Configure the job with FTP- and file information and make sure to set the JobsAfterUpload.

ReSendMessageToNewInGroup

Works like SendMessageToGroups, but keeps track of what units have received this message before. Will probably be marked obsolete in the near future, since there are other ways to do this.

The use case that this job were designed for was when you want to add members to a group, like registering customers and send a welcome-message, but only to those in the group that were added since last execution.

Notes:

High risk of being obsolete, get in touch with support or dev-team to know how to re-design solutions with this job.

How to:
Get in touch with support or dev-team for more information.

ResetSynchronizedCounter

This job is used together with SynchronizedCounter

This job is a complement to the SynchronizedCounter and can reset the counter or change the limit dynamically.
Can be used to initialize the counter to a given value.

Notes:
This job is used together with SynchronizedCounter

How to:
Set the counter name and key to the ones you want to reset synchronized counter for.
Set the reset value (note can be other than zero, 0)
Set the counter limit to change the limit of the counter.

RouteFromGroupMemberMetaData

This job is similar to RouteFromMetaData, but this job is no longer supported and treated as Obsolete. Use RouteFromMetaData or ExecutionRules instead.

Use RouteFromMetaData or Execution rules instead.
This job would route based on the IncomingGroupMember's metadata.

Notes:
Use RouteFromMetaData or Execution rules instead
Need IncomingGroupMember in WFC.

How to:
Use RouteFromMetaData or Execution rules instead

RouteFromMetaData

This job choses what jobs to continue executing next. Each metadata-route creates a new evaluation and a way for the workflow to continue execution with the jobs connected to that evaluation.

This job makes a decision on what jobs to execute next.
See how to compare metadata (actually WFC-resources) to understand how to configure the evaluations.

Compare value is used when you have a fixed value to compare to, and compare value source when you need a WFC-resource to compare to.
Name is just a help with the visual in WF-builder.
Meta data source is what metadata (WFC-resource) you want to evaluate.

Notes:
See MetaData-comparison for more details on how to set up the evaulations.

How to:
Create as many metadata routes as you need and set up the evaluation for each of them. Connect one or more jobs to each evaluation and they will be executed after evaluation matches. Connect to the RouteDestinationOnNoMatch to catch every other case when none of the evaluation matches.

RouteFromWorkflowMetaData

Obsolete, do not use.

This job does nothing that

Notes:
Use RouteFromMetaData or Execution rules instead

How to:
See RouteFroMetaData or see ExecutionRules-section.

RouteGuard

This job is about to be obsolete and it is not recommended to use. What it does is acts as a guard and only let one execution pass the place where the guard operation is Lock, until that has passed another RouteGuard-job with the operation Unlock, no other execution will continue, just keep looping and wait for the next possible time to execute.

Blocks every execution of the workflow until first execution has passed through the Unload-operation or max locking time has passed.

Notes:
This job is known to cause problems and mostly a well-planned workflow will be better designed with clever usage of Synchronized counters.

How to:
Get in touch with the support or dev-team if you think you need this job.

SaveToGroup

This job can create groups and save group members to the given group.

If you want to create a new group set the "CreateNewGroupAndIgnoreGroupId" to true. This will ignore what is connected in WF-builder.
Set NewDynamicGroupTagsSource to a WFC-resource that contains a comma-separated list of tags to include if you want to create a dynamic group with the given tags.

If you do not want to create a new group, then connect a group in WF-builder.

If you want to save members from WF-groups set "Use workflow context groups" to true.
If you want to save members from WF-temporary group set "Use workflow context temporary group" to true.

Notes:
Can both create new groups/dynamic groups and save to an existing group.

How to:
Chose if to create a new group or connect to an existing.
Select what members to save to group.

ScheduleNextJobs

This job can both execute jobs directly after being executed or start next jobs with a scheduling rule.

If you would like to schedule the execution of the next job you could use this job. Set the date time source to use a WFC value and schedule the execution of the next job.
Allowing scheduled execution to start in the past can be a good idea if you need the next job to be executed even if the expected scheduling time has passed when executing this job.
You can set custom date time parse formats (see parsing formats for Microsoft dotnet, or the section in this document on dates in MR).
Remember to set the destination for scheduling id if you want to be able to stop the scheduled execution.
In case of error and you cannot stop an execution, you can set the workflow or next job to inactive and it will not execute.

Notes:
Remember to set destination for scheduling id.

How to:
Set the source and/or rule for execution (see section on scheduling for more information).
Connect next job(s) from the "Scheduled jobs" if you want them to

execute on schedule.
Connect next job(s) from the regular “jobs” if you want to execute without schedule.

SendMessageToGroups

This job sends the message (connected message template) to the connected groups or defaults to WFC-groups and WFC-temporary groups.

If there are no groups connected, the job will find receivers in WFC-temporary group and WFC-groups.
Fixes some default-settings for app-messages, such as setting the inbox and sender id to a default value from merged settings (see section on merged settings for more information).

Notes:
Sets defaults for MessageTemplates (only app-message in this case)
Defaults to WFC-groups and WFC-temporary group.

How to:
Set the message tempalate.
(Optionally) Set the groups.

SendMessageToGroupsExceptInGroup

This job works like SendMessageToGroups but will remove units that are present in the “ExceptGroup”. Think of this as a SendMessageToGroups with a stop-group that stops some units from receiving messages (will not stop if a new unit with same phone/email… is created)

Sends to groups, but not to the members of the Except-group.
See SendMessageToGroups for more details.

Notes:
Sets defaults for MessageTemplates (only app-message in this case)
Defaults to WFC-groups and WFC-temporary group.

How to:
Set the except-group.
Set the message tempalate.
(Optionally) Set the groups.

SendMessageToUnits

This job works as SendMessageToGroups, but is only sends message to the units connected in WF-builder.

Sends message to units, not group members in the given groups.

See SendMessageToGroups for more information.

Notes:
Sets defaults for MessageTemplates (only app-message in this case)

How to:
Connect units to send message to.
Connect the message template to send.

SetDebugMode

This job can set or unset the debug-mode for the workflow.

This job can be used as a “break-point”. You may with this job change the workflow state to be in debugging mode. The debugging mode will stop the execution after executing this job (if the setting is to turn on debug mode).

The debugging workflow context must be activated again by commands to the API or from the workflow builder.

Notes:
See information about debugging workflow contexts.

How to:
Set the debug mode to be turned on or off after executing the job.

SetGroupMemberMetaData

This job will set the group member metadata.

Gets the value to set from ValueSource (which gets a value from a WFC-resource) or defaults to what is written in Value.
The Key is a fixed value and cannot find it’s value from WFC-resources.

Notes:
If no incoming unit is present it will set metadata for all members in the selected group.
If Incoming unit is set the job will only set metadata for that if the incoming units is a group member in the given group.
Also note that this job sets the GroupMember’s metadata, not the units metadata, but the overridden data for the unit when the unit is treated as member of a group.

How to:
Connect a group where to set group-member metadata.
Make sure to clear IncomingUnit if you want to set the metadata for all group members.
Set the Key and the Value/ValueSource

SetIncomingUnit

The goal of this job is to set the IncomingUnit resource on the current WFC.

This job sets the Incoming Unit resource on the WFC to the unit connected in workflow builder.

Notes:
Can only set the connected unit as incoming unit.

How to:
Drag-connect the unit to set as incoming unit.

SetUnitsMetaData

This job will set a given metadata on each unit in WFC-temporary group + WFC-groups + those connected in workflow builder.

The job will get all members from WFC-groups and add to WFC-temporary group.
Then try to find the connected units from the account.
Finally the job will set the given metadata on each of the units.

Notes:
Will put WFC-group members in temporary group before adding metadata.

How to:
Set the property to allow getting metadata values from workflow context.
(Optionally) connect certain given units to the job to set their metadata as well as the ones in WFC-groups and WFC-temporary group.

ShareListOperations

This job can perform operations on a share list.

This job can be used to perform operations on a share-list.

Notes:
See share-lists.

How to:
Configure the ShareListOperations

SplitWorkflowContext

This job splits the current executing workflow context into different copies for each connected job.

When this job executes it will see how many jobs are connected as next jobs and create a new copy of the current workflow context for each of the next job and then execute them.

This is very useful if you've loaded a number of units into the WFC-temporary group and would like to have some receive one message and another part of them receiving another message. Once you have a copy of the WFC you don't have to worry about both processes removing each other's units.

Notes:
Use this job to make sure that parallel job-executions don't touch each other's WFC-resources.

How to:
Just connect as many next jobs to this as you like, and the job will create a fresh copy of the current executing workflow context for each of the next jobs.

StopListOperations

This job can perform operations on a stop list.

This job can be used to perform operations on a stop-list.

Notes:
See stop-lists.

How to:
Configure the StopListOperations

StopScheduledNextJobs

This job should be used together with ScheduleNextJobs, and the responsibility for this job is to abort a scheduled execution of next jobs.

Use the schedule-id (the result from ScheduleNextJobs) to abort the scheduled execution of a next job.

Notes:
Cannot abort the execution of a next job if it already has started or executed.

How to:
Set the SchedulingIdSource to the metadata or WFC-resource that contains a scheduling-id to stop.

StoreData

This job is almost the same as LogData-operation in DataOperations. It will add another log item to a given datalog.

This job will create DataLog-Items for a DataLog (with the DataLog-key *store data with key*).
This job will create a new DataLog document if none present.

Notes:
-

How to:
Set the data to store in the log item.
Remember to set a key.

SynchronizedCounter

This job is a counter, and the synchronized-part of the name indicates that it will synchronize the counting even if there are many workflows that simultaneously want to increase the counter.

The counter name is just the name of a counter and in combination with the counter key it creates a unique counter with a counter limit and current value and so on.
For example, if I want my counter to limit the executions per incoming email address, I would set up a counter with the name **emailcounter** and dynamically use the different email sender addresses as a key. So if sender1@bosbec.io sends email to the workflow the counter **emailcounter + sender1@bsobec.io** will increase its current value, but the counter **emailcounter + another_sender@bosbec.io** will not increase.

CounterName, **CounterKey**, **CounterLimit**, **RequestedIncrement**, **MinIncrementChunkSize** can all be set either as the value direct or set as a WFC-resource containing the value.
RequestedIncrement and MinIncrementChunkSize will default to 1 if no other valid value is set.

When the counter is executed, it will produce a result and a current value that can be put in any available workflow metadata or equivalent resource.
The results are:

- **OK** – which means that increasing the current value for the counter was OK, no limit was reached or already passed. Ex. the limit is 10 and the counter is now 3.
- **Limit reached** – which means that there are no more room to continue counting. Ex. the limit was 10 and the counter is now 10.
- **Fail** – means that it is not possible to increase the counter, and no increment has been made. Ex. limit was 10, counter value was 10 before trying to increase, counter value is 10 after job execution as well.

Another thing to mention here is that the RequestedIncrement and MinIncrementChunkSize can be for some cases when counting. The RequestedIncrement is how much we want to increase the counter with at the time of execution. (It can be a dynamic value from a WFC resource or a fixes value configured on the job). For example RequestedIncrement can ba 2 and each time the counter is

executed the value will increase with 2. Ex. **First execution: 2**, **Second execution 4** …
MinIncrementChunkSize, on the other hand is used together with the RequestedIncrement. It tells the job if it is allowed to increase the counter-value by a part of the RequestedIncrement. So if the RequestedIncrement is 2, but the MinIncrementChunkSize is 1 it means that such a counter can have an odd number (ex. 3) as limit, even though we increase it with 2 (most of the time). Ex. ***First execution: 2***, **Sencond execution:** *cannot increase with RequestedIncrement (2), will attempt a smaller chunk (1), and that works result:* ***3***

Notes:
The job will execute next jobs even if the result is Limit reached or Fail, and you need to use the result in ExeccutionRules or MetadataRoutes.
For now this job is designed to be used with counting "up", that is +, adding and increasing the numbers.

Tip: Use in combination with the ResetSynchronizedCounter job!

How to:
Set a counter name (usually you will just need one counter name for a workflow, since you have counter key to make it unique within a workflow)
Set the counter key, counter limit... and the other properties as described above.
Remember to set the output value and result to make use of it when routing after the job is executed.

TagUnits

This job tags units in WFC temporary group and WFC groups.

The job will read members from all WFC groups and put in WFC-temporary group before adding tags.
Can remove many tags.

Notes:
Adds tags to units on account, not just during the execution.
Will read WFC-groups into WFC temporary group before adding tags.

How to:
Set a single tag in the Tag-text field to add that tag.
Set TagsFromWorkflowMetaData to a WFC-resource and it will split the content of the resources value with comma, semi-colon and space and then add all of them as tags to each unit.

UniqueSenderRoute

Will make sure that the sender of the incoming message only passes through to the "Jobs when evaluation match" one time.

This job can be configured with a manually created group or it will automatically generate a group upon first execution and update itself.
The job stores units in a group to make sure that the incoming sender only can pass through to the "JobsWhenEvaluationMatch" one time, and the rest of the times it will continue with "JobsWhenEvaluationDoNotMatch".

Notes:
Requires incoming message sender.

How to:
Create a group or let the job create it for you once executed the first time.
Connect jobs to the "JobsWhenEvaluationMatch" if you just want those jobs to be executed one time per unit/sender.

UntagUnits

This job removes tags from units in WFC temporary group and WFC groups.

The job will read members from all WFC groups and put in WFC-temporary group before removing tags.
Can remove many tags.

Notes:
Removes tags from units on account, not just during the execution.
Will read WFC-groups into WFC temporary group before removing tags.

How to:
Set a single tag in the Tag-text field to remove a certain fixed tag.
Set TagsFromWorkflowMetaData to a WFC-resource and it will split the content of the resources value with comma, semi-colon and space and then remove all of them as tags to remove.

ZipFile

This job can zip a file from the file service.

Need a batch-id from when the message with form was sent.
Can get both the units that did respond to the form and the ones that didn't.

Notes:
Need a batch-id, and a batch with a form.

How to:
Configure the settings and connect the next jobs to the JobsAfterRetrieve.

What WFC resources are available and how to use them:

From many places in the workflow it is possible to access the data and resources that are available. This section describes most of what is to know about it.

Nested Metadata-expressions can help in some cases. But what is a nested Metadata-expression and how do I use it?

A simple case of when it is possible to make use of nested metadata-expressions is when I have 3 metadata properties on my workflow context;

metadata.test1 = "Thank you!"
metadata.test2 = "Thanks for your feedback!"
metadata.test3 = "Your opinion is valuable for us, thank you!"

And I would like the text body of my message template to end with a random "thank you"-message. Then I could use the random-number helper together with the known part of my metadata-key. Like this:

Source: metadata.test{[rndnum(1,3)]}
Destination: metadata.message-ending

And in my message template I would now end the message with:
[message-ending]
Then the result would be one of the three alternatives that would end the message from the message template when sending it to the receiver.

Metadata-comparison:

The compare operator can be different depending on what type of value to compare.

- **between** – define lower and upper values like this **1;10 | A;Z** and compared value must be between the two values.
- **!between | not between** – the inversion of between, when the value is not between the two given numbers.
- ***(NUMBER COMPARISON ONLY)*** **is in | isin** – a set of values separated by semi-colon or comma **1;3,5** and the job tests if the value to compare matches any of the given values.
- ***(NUMBER COMPARISON ONLY)*** **Is not in | notin** – the inversion of **is in** comparison above. Tests that value is not a match to the numbers given in the set.
- **= | ==** - Compare if the values are equal
- **<** - Compare if the value to evaluate is less than the value to compare to.
- **<= | =<** - Compare if the value to evaluate is less than or equal to the value to compare to.
- **>** - Compare if the value to evaluate is greater than the value to compare to.
- **>= | =>** - Compare if the value to evaluate is greater than or equal to the value to compare to.
- ***(STRING/TEXT COMPARISON ONLY)*** **startswith** – Tests if the text starts with the given value, ignores upper/lower-case.
- ***(STRING/TEXT COMPARISON ONLY)*** **startswithcase** – Tests if the text starts with the given value, takes upper/lower-case into consideration.
- ***(STRING/TEXT COMPARISON ONLY)*** **endswith** – Tests if the text ends with the given value, ignores upper/lower-case.
- ***(STRING/TEXT COMPARISON ONLY)*** **endswithcase** – Tests if the text ends with the given value, takes upper/lower-case into consideration.
- ***(STRING/TEXT COMPARISON ONLY)*** **contains** – Tests if the text contains the given value, ignores upper/lower-case.
- ***(STRING/TEXT COMPARISON ONLY)*** **!contains | not contains** – Tests that the text doesn't contain the given value, ignores upper/lower-case.
- ***(STRING/TEXT COMPARISON ONLY)*** **containscase** – Tests if the text contains the given value, takes upper/lower-case into consideration.
- ***(STRING/TEXT COMPARISON ONLY)*** **!containscase | not containscase** – Tests that the text doesn't contain the given value, takes upper/lower-case into consideration.

Coming:

The following is just a short description and more information will come in future versions of this document.

RegEx

Regex is used to express patterns that is found in text.
We use the dotnet-platform behind our regex-jobs.

JSON & XML

JSON and XML are two formats to represent data and in most cases, we make use of JSON-format. To read more about the JavaScript Object Notation (JSON) go to: https://www.json.org/

XML is used in some workflows and if you are planning implementations where you use XML, you should get in touch with the support or development team support@bosbec.com

ExecutionRules

The Execution Rules may be used like RouteFromMetadata-job or as a way to create debugging-breakpoints.

An example of an execution-rule is:

```
[
  {
    "EvaluationTime": "BeforeExecutingJob",
    "Order": 1,
    "Resource": "metadata.test1",
    "ComparedTo": "metadata.test2",
    "Operator": "==",
    "MatchActions": "debug=false",
    "MisMatchActions": "debug=true"
  }
]
```

- The EvaluationTime decides if the rule should be evaluated before or after the job is executed. (settings are: BeforeExecutingJob | AfterExecutingJob)
- Resource may be any metadata or similar resource that is available at the current time in the workflow context.
- ComparedTo is, just as Resouce something from workflow context that is compared to the Resource.
- The operator is the same type as can be used in RouteFromMetaData
- MatchActions and MisMatchActions is the two different scenarios.
    - If the evaluation results in a true expression, then the MatchActions will happened.
    - Else the MisMatchActions will happened.
    - The available actions are
        - debug
        - executenextjobs
        - executecurrentjob
    - And the action can be turned on or off.
        - debug=true
    - The actions can also be separated by semi-kolon.
        - debug=true; executenextjobs=false