Introduction & Reference
Overview

In this document, we have gathered information about the resources and concepts related to the platform previously known as MobileResponse and its Workflows.

OBS!
As of November 2017, we are changing the name of MobileResponse in order to provide a better and more unified naming and branding throughout the platform.
The platform continue operations as before, but under the name:

Bosbec WE

Also note that the domain-names will change from(mobileresponse.io or mobileresponse.com to bosbec.io. The old domain-names and URLs will still be active for a while, but 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, BosbecWE or the platform. There are still smartphone-apps available under the name MobileResponse and URLs / domain-names with mobileresponse.com or mobileresponse.io are still valid and may be referred to in this document. If you find errors or have questions about the documentation, please get in touch with us: support@mobileresponse.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 is 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 are used for a wide range of purposes. Here is 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.
- As a way 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 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 and start the execution of a series of jobs. Triggers can still be configured and impact the way the workflow works.

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.

Jobs

Jobs are the small building-blocks that work together to create the solution 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 make use of. Some of the jobs does however make sense only in certain cases. Some jobs that are visible today may be hidden or removed in the future.

How we work with jobs over time

Our vision is that we provide as much tools as possible, and every day try to empower our customers with better building-blocks (jobs) or improve functionality and performance of existing jobs. But sometimes a
job has drifted too far away from the rest of the concepts or they become obsolete. New jobs are from
time to time implemented with improvements that replaces old functionality or can do a better job in a
slight different way. When this happen, we try to make sure that no workflows are using obsolete jobs.

If we want to retire jobs, we will first make sure that existing jobs can be viewed but not modified, users
will be informed on how to take actions to update their workflows and new jobs of the retired type can no
longer be created.

Resources
Since the first version of this document the concept of resources have expanded. Before the word
resource would indicate the kind of resources that can be listed and operated on from within admin-GUI
or inside the workflow-builder. But resources also refer to an object that is created during the execution
of a workflow. For example such a resource can be JSON-resource or XML-resource. And they are a bit
different from the resources presented below. *Note that more information on such resources will be
included in this documentation in coming versions of this introduction/reference.*

In workflows, we make use of different kind of resources. We manipulate data on our units, we select
receivers (Units) from a group, we need a message (MessageTemplate) to send a message with a
questionnaire (Form) to the recipients.

Units
A Unit is our most generic model. The idea is to let our customers decide; what do I need to model this
time or in this case.

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

- A *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 “incident-case” or “ticket”.
- IOT (Internet-Of-Things) – devices communicate with a workflow, a unit represents the current
  state of the device, current battery-status, current temperature in the room. One unit is used per
  IOT-device.
- Customer SMS survey – units model the customer

Units have phone, email and IOT (which is actually MQTT) endpoints. 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. We cannot group anything other than units. A group has metadata, and
the metadata of the group are for now primarily used by Admin-GUI to display custom properties in the
group-member listing.

In the future, the metadata of the group will be used in more cases.

\(^1\) Typical workflow that solves a case where the employer need 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.
Messages
The messages (MessageTemplates) are just as the name suggests a template. The template contains up to 4 different message parts, but the parts must be of different type.

- **App-message** to send messages to MobileResponse/Bosbec - iOS and Android Apps.
  - Unique properties are SenderId and InboxId can be configured to send app-messages from different app-users.
- **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 is what we are creating with Workflow-Builder. The 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 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 BosbecWE 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

\(^2\) 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
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.

There are some exciting updates planned to go live later this year in the area of forms:

- One thing we are working on is simplification, forms will be more simple and easy to link to each other with workflows, and at the same time merge exports of many forms into one CSV-file.
- Another thing is how we are improving the ability for workflows to control the “flow” or order for questions in forms.
- Triggering upon respondents visiting the form/link

DataLog

If you want to store more than just the current/present value, then the DataLog is a better option than for example a unit. The DataLog is design just to be a container for data. You may specify metadata for the DataLog and use it when visualizing or getting data from the API.

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.

The DataLog consists of two parts, the DataLog -setup (DataLogDocument) and the DataLog -Items. The Items is the actual value + time of creation.

Services

Services is an extension of what workflows can do. Services is for now only implemented by BosbecWE dev- and support-team. This is because there are no 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 need 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 MobileResponse/Bosbec app (web-app or smartphone-app).

AppUsers will soon (around new-year 2017-2018) be able to view and execute the “Services”, which will reduce the need for many administrators on an account.

**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.

**Underneath the hood**

For a more complete understanding of how to design solutions with Workflows or debug Workflows we have to introduce the concepts of WorkflowContext and Processes.

**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 where most workflows put units before processing them or sending messages to. Most jobs that can create or find units will also put the result in the TemporaryGroup.

- **The Workflow Groups** (in some cases referred to as Workflow Context Groups) – These are groups that you drag out to the workflow canvas and leave them without connection to any job. They are then “lazy”-loaded into the workflow context, 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 number of 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** – This is one of the most recent concepts that are being used more and more throughout solutions. A Resource can only be set with jobs that can create resources. Resources can then be accessed in a way similar to how we use the metadata in workflow contexts.

**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
We have two primary graphical tools to our assistance when developing workflows. One of the tools is our Admin-GUI, where you can view things like who is in a group, what is my message-history and so on. But you can also view the DataLog and open workflows in the workflow builder.

In the workflows-area you should be able to list executions for each workflow and see the workflow context and the process events.

In workflow-builder you can build your workflows and forms, more detailed description could be found under each section above. It is also possible to create some of the workflows resources both in Admin-GUI and workflow-builder (ex. Groups or units).
The references

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 Jobs

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

AbandonKeyword

OBSOLETE!

This job will either change or be removed later this year.

How to:
Do not use this job in new workflows and do not change settings for this kind of jobs in old/legacy workflows.

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 continue 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 continue 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.

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). Set 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

**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 \(\rightarrow\) will map to unit property):
- phone, phonenumber \(\rightarrow\) Unit.PhoneNumber
- email, emailaddress \(\rightarrow\) Unit.EmailAddress
- tags \(\rightarrow\) 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.

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 \(\rightarrow\) Will set the phone
- email \(\rightarrow\) Will set the Email
- metadata.test1 \(\rightarrow\) Will set the metadata with name “test1”
- tags \(\rightarrow\) A key that starts with tags will set a tag ex.
  - tags1 \(\rightarrow\) Will set a tag
  - tags2 \(\rightarrow\) 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”)

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

  ![Concat Example](image)

  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.

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. Postpone 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.

ExportDataLog

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

TODO: add description and example of rules here.

Notes:
This job needs data in the datalog.

How to:
ExportFormAnswersToWorkflowContext

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

TODO: add description and example of rules here.

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 extractions (1 & 3) it is required that the execution is triggered by a process-trigger.

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

Filter Receivers From Workflow Context

This job 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 do

- Should Remove Delivered App Message Receivers
  - An app-message is delivered when the app-users opens the MobileResponse app (more precise it is when the app calls the What-is-new function in the app-API)
- Should Remove Message Receivers Without App User
- Should Remove Receivers Who Completed From In Current Process
  - 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.
- Should Remove Receivers With Metadata
  - Will remove receivers that match the metadata defined in the metadata-key and -value properties.
- Should Remove Receivers Who Match Incoming Sender
  - 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 depending on whether or not you want to keep the receivers that match the filters.

Filter Workflow Context

The purpose of this job is to filter contents of the workflow context. There is only one implemented filter and that is the Temporary Group Metadata Filter.

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.

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.

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.

Notes:
Can set Incoming Unit.

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

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.
Do not 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.**

**How to:**

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

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-resource. 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 criteria-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 friends** - “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.

So 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 was 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. (TODO: add link to their report?)

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 CalcAverageTimeUntilNow 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.
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 amount 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. It will work, but it is not recommended to use this job. One reason for this job being obsolete is that it is not synchronized.*
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. Don’t use it!*

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:**
Don’t use.

**How to:**
Don’t use, get in touch with support- or dev-team if you need this functionality and they can help you with other ways to do this.

### 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-wrappper 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)

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.

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 chooses 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 RouteFromMetaData 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 template.
(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 template.
(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.
SendMessageWithFileInformationToGroups

This job sends file-information as AppMessageMetadata to group members.

The job was designed as a Proof of concept when workflows began to handle files and are focused on forwarding file-information to app-messages. Adds AppMessageMetadata with the file information (the files in WFC-files resources)

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

How to:
Basically use as SendMessageToGroups.

SetGroupMemberMetaData

This job inactivates the current workflow.

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.

**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.

**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@bosbec.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 be 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, Second 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 ExecutionRules 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.

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;

\[
\begin{align*}
    metadata.test1 &= \text{“Thank you!”} \\
    metadata.test2 &= \text{“Thanks for your feedback!”} \\
    metadata.test3 &= \text{“Your opinion is valuable for us, thank you!”}
\end{align*}
\]

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:

\[
\begin{align*}
    \text{Source: } &metadata.test\{[\text{rndnum}[1,3]]\} \\
    \text{Destination: } &metadata.message-\text{ending}
\end{align*}
\]

And in my message template I would now end the message with: \texttt{[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.

- \texttt{between} – define lower and upper values like this \texttt{1;10 | A;Z} and compared value must be between the two values.
- \texttt{!between | not between} – the inversion of between, when the value is not between the two given numbers.
- \texttt{(NUMBER COMPARISON ONLY) is in | isin} – a set of values separated by semi-colon or comma \texttt{1;3,5} and the job tests if the value to compare matches any of the given values.
• **(NUMBER COMPARISON ONLY)** is not in | not in – 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 or equal to 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/](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@mobileresponse.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 Resource 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.
  o If the evaluation results in a true expression, then the MatchActions will happened.
  o Else the MisMatchActions will happened.
  o The available actions are
    • debug
    • executenextjobs
    • executecurrentjob
  o And the action can be turned on or off.
    • debug=true
  o The actions can also be separated by semi-kolon.
    • debug=true; executenextjobs=false