
Cuesta manual

September 21, 2022



Contents

Introduction	3
Cuesta and Kwanza	3
Landing page	4
Concurrent edit protection	5
History of changes	5
Keyboard shortcuts	7
Subjects	9
Creating subjects	10
Applications	12
Types of applications	13
Creating and editing applications	14
Fields	16
Creating fields	21
Using a field in a flow	23
Flows	23
State flows	23
Mapping flows	25
Action flows	25
Module flows	27
Triggers	28
Creating a flow	28
Editing a flow	30
Groups	36
Tables	37
Creating and using tables	37
Services	38
Mail	38
Serial port	40
Chat	42
Conductor	42
Filtering Manatees	42
MQL	45
Detailed view	47
Alerting	49
Conditions	49
Actions	51

Tracking alerts	52
Restrictions	53
Deny	53
Prompt the user	54
Log the invocation	55
Check parameters	56
Affected flows	57
Settings	58
Localization	59
Hub	60
Configuring hubs	61
Publish flows	63
Get flows from a hub	68

In order to better understand how the driver platform (Manatee) operates and how the configuration interface (Cuesta) is structured it is helpful to know a bit about the history of the system.

Introduction

The CCOW standard specifies an architecture for keeping multiple applications synchronized within a shared context. Typically this context could be a patient and when one application decides to switch patient all participating applications are made to follow the switch by a context manager (CM).

Originally Manatee was built to support our CCOW context manager (Seacow) since very few applications actually implement the HL7 CCOW protocol needed to interoperate with Seacow. Manatee then provides a layer which operated on the UI or API of an application to make it CCOW compliant. Gradually Manatee outgrew the Seacow and began to support regular UI automation as well, i.e. a sort of virtual machine for scripting steps to be performed on the UI of a given application.

The concepts from the CCOW standard is still prevalent in the configuration interface and in the engine which schedules automation as well as context synchronisation is still a true CCOW context manager.

Cuesta and Kwanza

Cuesta is the name of the configuration interface in which Manatee can be configured and the subject of this manual. All configurations are stored in our Context Management Registry, CMR in shorthand, which we have called Kwanza. Kwanza is a fairly simple infrastructure element - it has a couple of features which are worth noting though.

Landing page

After logging in you'll see the landing page.

Welcome to Cuesta

Use Cuesta to code automations for Manatee and configure when, where and for whom they are run.

Search for flows, fields, apps and tables.

The screenshot shows the Cuesta landing page. At the top is a search bar with the placeholder text "Search". Below the search bar are three columns of flow lists. The first column is titled "Window interacting flows" and contains a list of flows: UpdateSearch, Irs registrering af udtagning, Implantatregistrering, 12A.1, Afslut usignerede, Find nogle bookinger, ENTER, Test Bestil Patienttransport, Basic_Functions_Module, and Login. The second column is titled "Last edited" and contains a list of flows: To befordring (a few seconds ago), sfsdf (a minute ago), and a vertical line. The third column is titled "Los flows" and contains two sections: "WebReq H-Chrome (copied by los)" with flows Opret Rekvisitioner Parallel, prepareBrowser, FixRedirectProblem, Patient Podning - chrome, InvokeExperiments, fubar, and findBadNames; and "los-headless" with flows CclActivityTest and Opret rekvisitioner parallelt. Red lines point from the text "You can configure 3 elements to display here. Here are shown a custom list with all flows that interact with windows on the left, a list of things I've recently edited in the middle and all flows maintained by the 'los' user." to the three columns.

Search

Window interacting flows

Last edited

Los flows

UpdateSearch

Irs registrering af udtagning

Implantatregistrering

12A.1

Afslut usignerede

Find nogle bookinger

ENTER

Test Bestil Patienttransport

Basic_Functions_Module

Login

To befordring
a few seconds ago

sfsdf
a minute ago

WebReq H-Chrome (copied by los)

Opret Rekvisitioner Parallel

prepareBrowser

FixRedirectProblem

Patient Podning - chrome

InvokeExperiments

fubar

findBadNames

los-headless

CclActivityTest

Opret rekvisitioner parallelt

You can configure 3 elements to display here.
Here are shown a custom list with all flows that
interact with windows on the left, a list of things
I've recently edited in the middle and all flows
maintained by the "los" user.

Figure 1: The landing page

The landing page has two primary elements. The first is a global search from which you can search for flows, apps, fields and tables by their name. The second is a user configurable area where 3 elements can be configured to show. This is done by finding your username in the primary left-hand side menu and choosing the elements to display under the "Landing page" heading.

Landing page

Configure what to display on the landing page

You can configure 3 elements shown below the search field on the landing page.

The image shows three separate configuration panels, each with three settings:

- Panel 1:**
 - Type: saved search
 - Saved view: Window interacting flo
 - Number of entries: 10
- Panel 2:**
 - Type: Last edited
 - Number of entries: 10
- Panel 3:**
 - Type: saved search
 - Saved view: Los flows
 - Number of entries: 10

Figure 2: The landing page configuration

Concurrent edit protection

Kwanza uses a lamport clock to ensure that the order of updates to any stored entity is preserved and that you cannot overwrite an entity without having seen the latest version beforehand. This translates to the fact that if two people edit the same thing at the same time one of these may get an error saying that their changes cannot be saved as it would overwrite the changes made by the first person. It is a safety feature and it makes editing an item concurrently safe, but it does not make it very easy to do so.

History of changes

All changes are tracked in Kwanza meaning that its possible in Cuesta to see the history of changes made to e.g. a flow, and it is possible to revert to a previous version. For flows this feature can be accessed through the [Change log](#) tab in the flow editor.

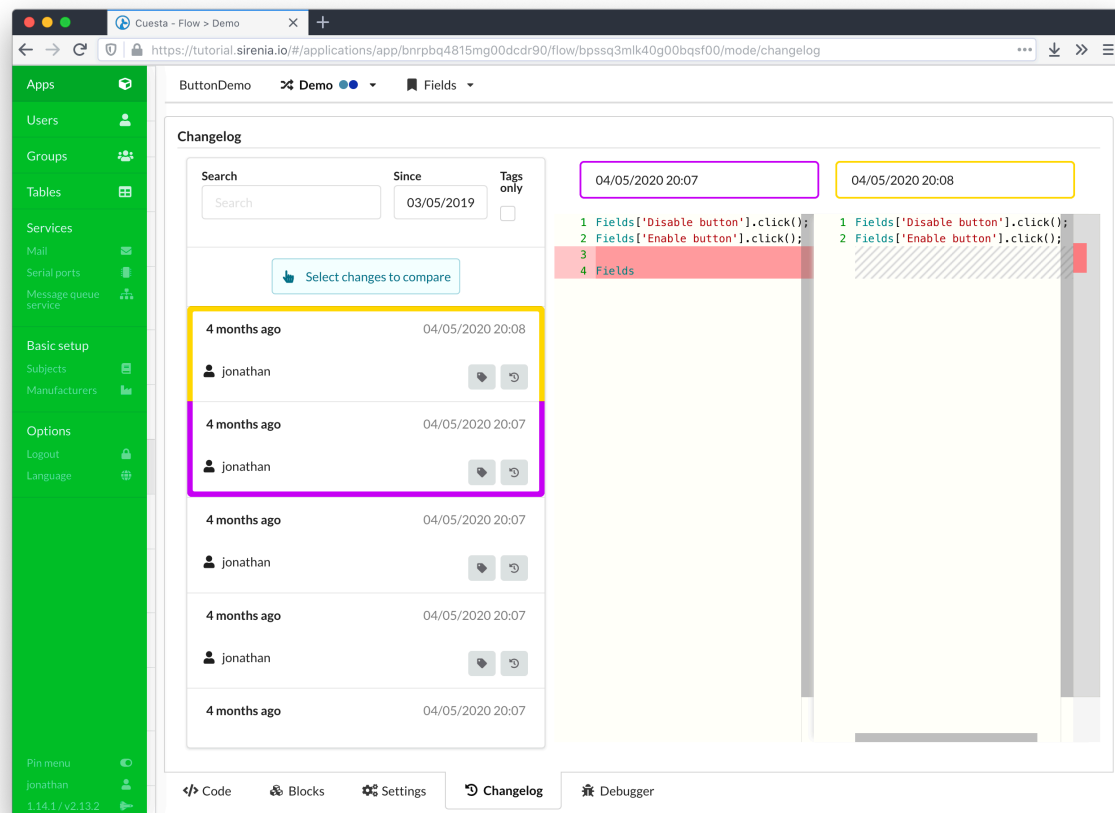


Figure 3: Flow changelog view

For the rest of the items such as users, groups, tables, subject, manufacturers and services the changelog is accessible through a changelog button on the item's edit page. The button provides a list of changes, and after selection of the specific change marks the fields which have been changed.

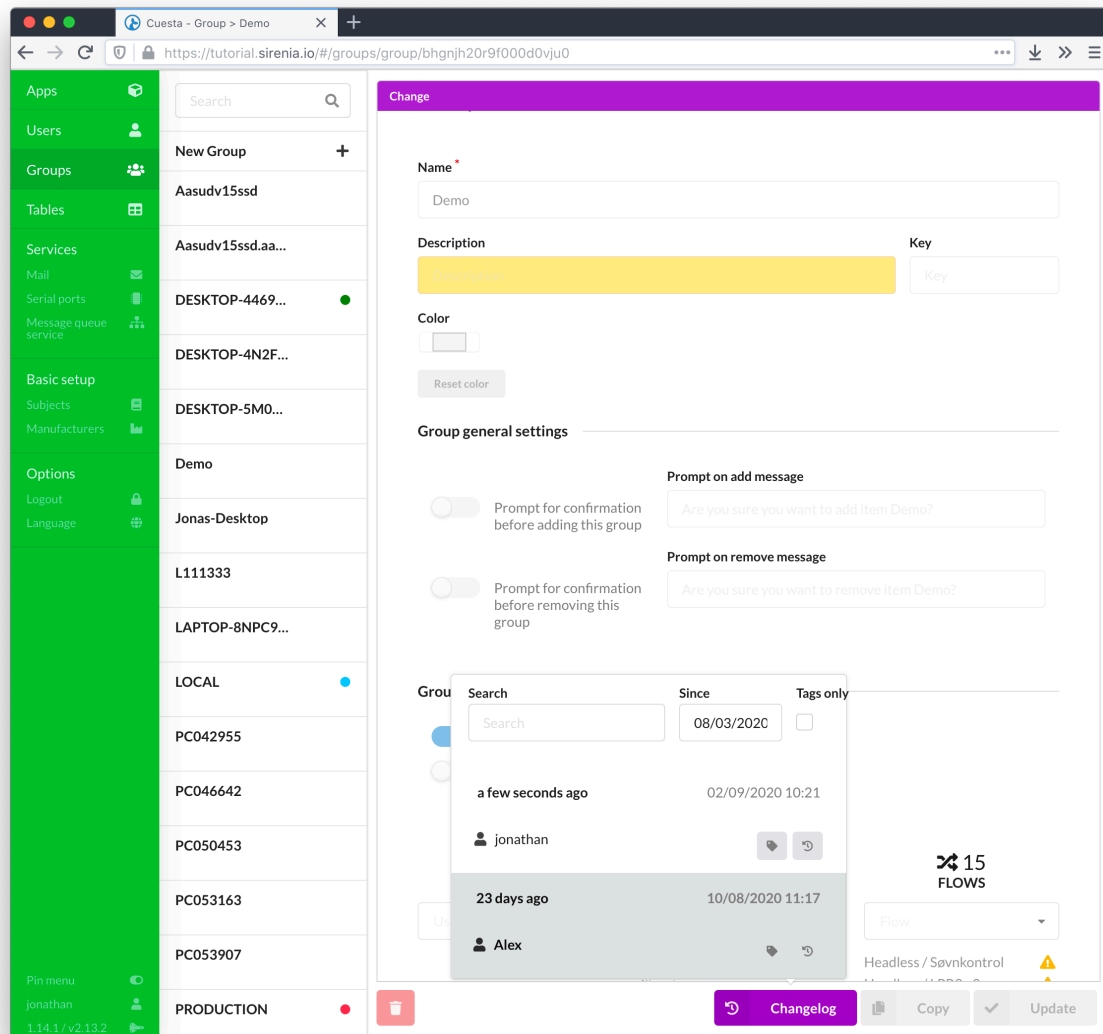


Figure 4: Group changelog view

Keyboard shortcuts

Keyboard shortcuts are available throughout Cuesta. On any screen you can hit the ctrl+h keyboard shortcut to show a window with applicable shortcut keys for this particular screen.

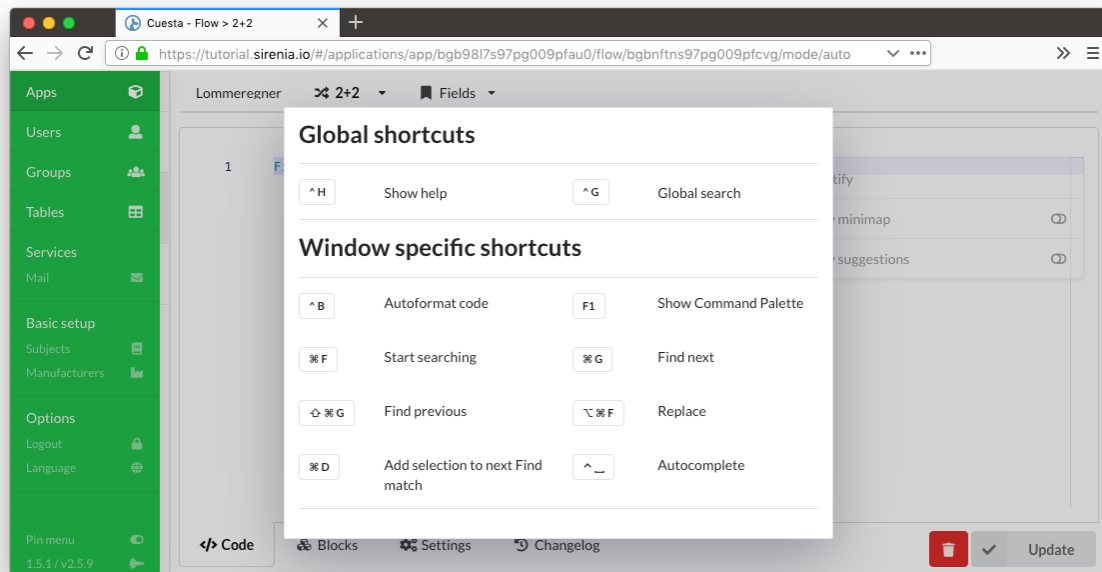


Figure 5: The helpful shortcut display

Global search

Hit ctrl+g on any screen to bring up the *global search* feature. Then begin typing. This functionality lets you quickly jump to a group, application, flow etc. as long as you type all or part of its name.

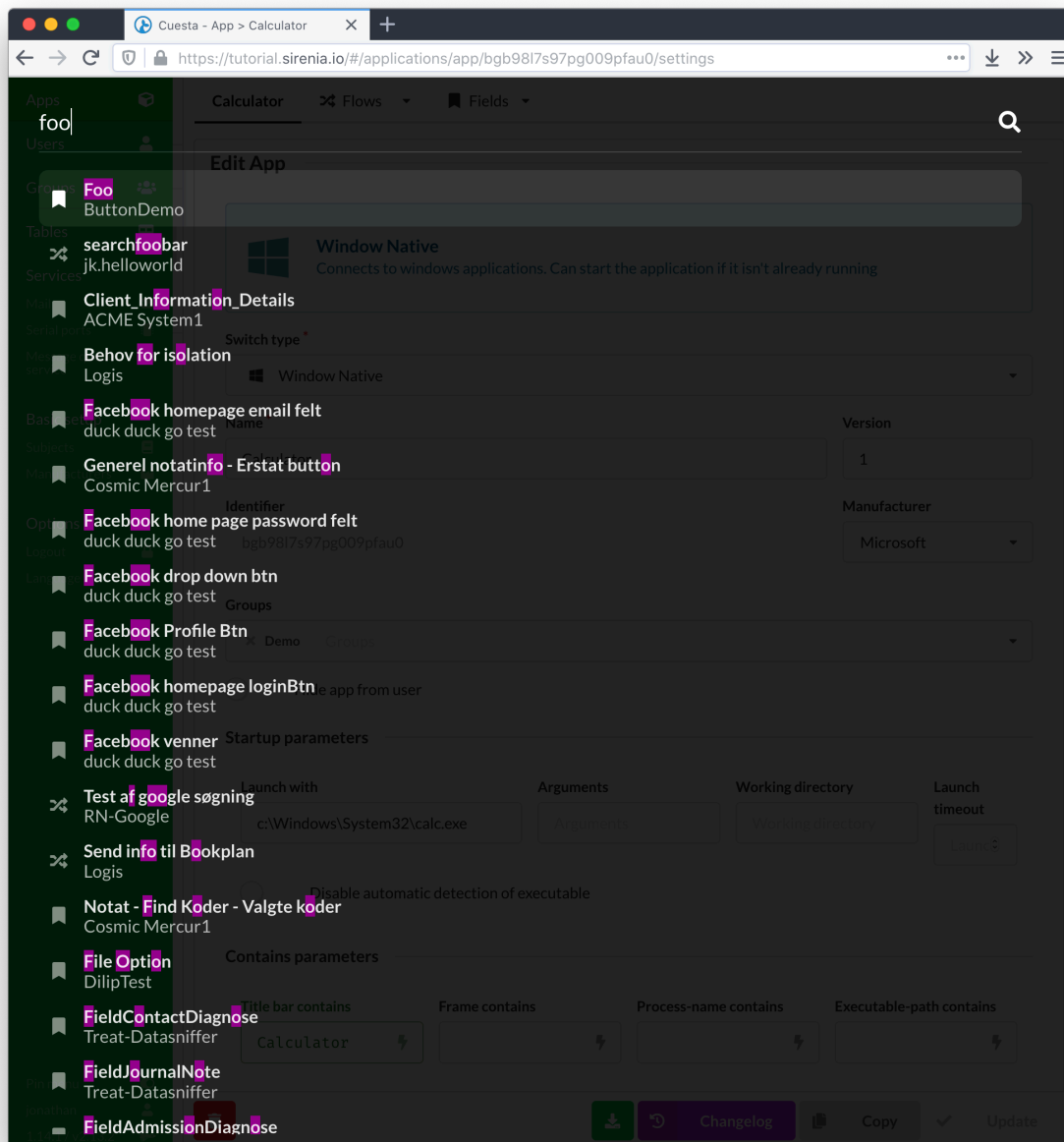


Figure 6: The global search view

Subjects

The shared context in a CCOW session is in essence a key-value store where the keys are termed *subjects* and usually have a specific format indicating the origin, type etc. of the information stored for the key. The concept of a *subject* is re-used in Cuesta as a key for identifying both flows to execute and

extracting structured information from the UI of an application.

Subjects are defined in terms of a name, a key (the actual subject) and a regular expression. The regular expression is used to determine whether the subject is present in a string extracted from e.g. the UI of an application.

Creating subjects

Subjects are created by entering the name and key (use something suitably unique here, preferably a proper CCOW subject) and a regular expression.

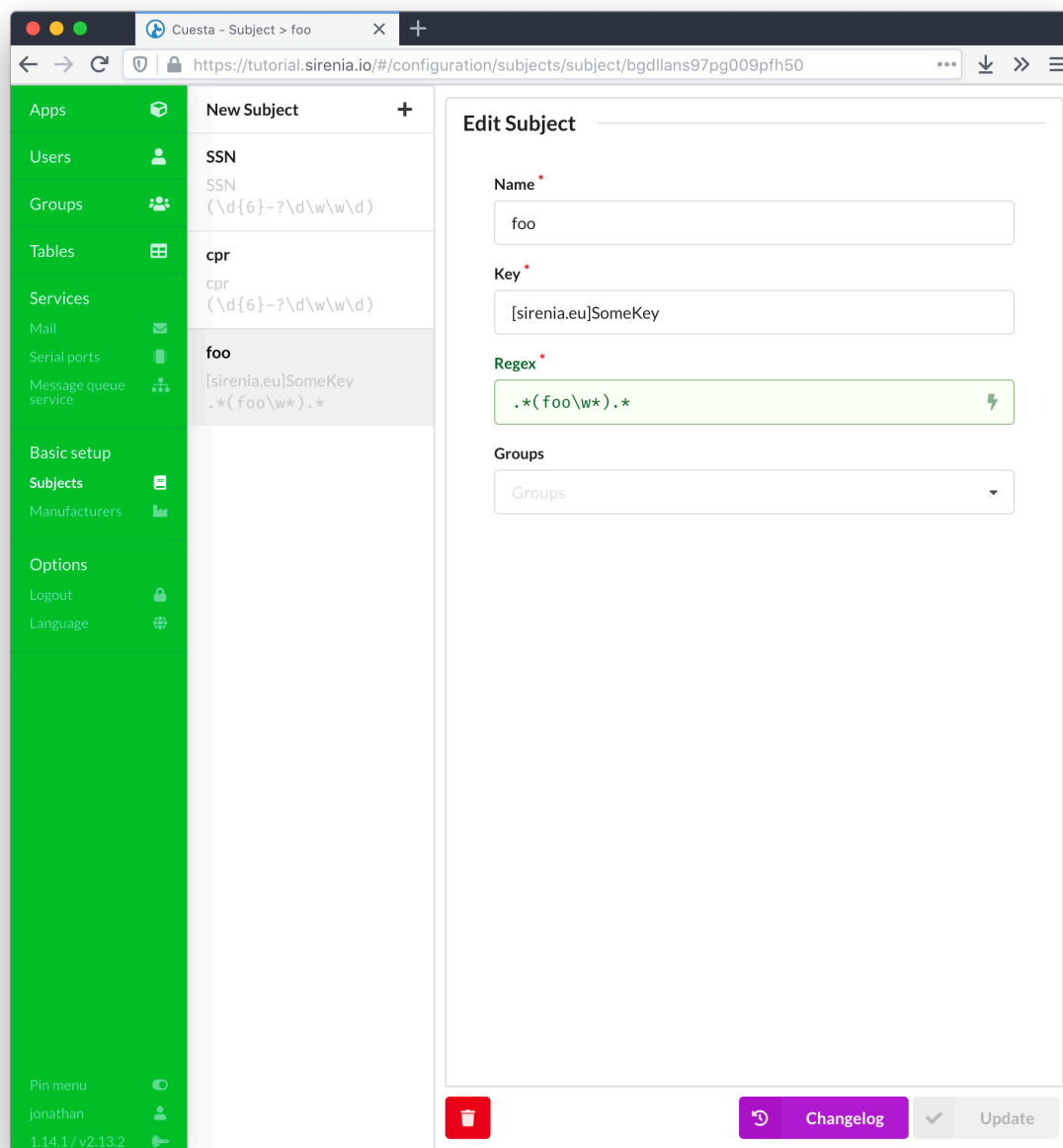


Figure 7: Subject configuration

The regular expression should contain at least one group. The group is used to extract the actual value for the subject. A named group (with name *value*) can designate a single group of multiple groups are required in the regular expression. Multiple named groups can also be used in which case the extracted information is made available to a flow as a stringified json object with group names as key and group values as values.

Applications

An application as defined in Cuesta corresponds to an actual application that the user wishes to include in a session. An application must be defined in Cuesta in order to be automated or take part in context synchronisation.

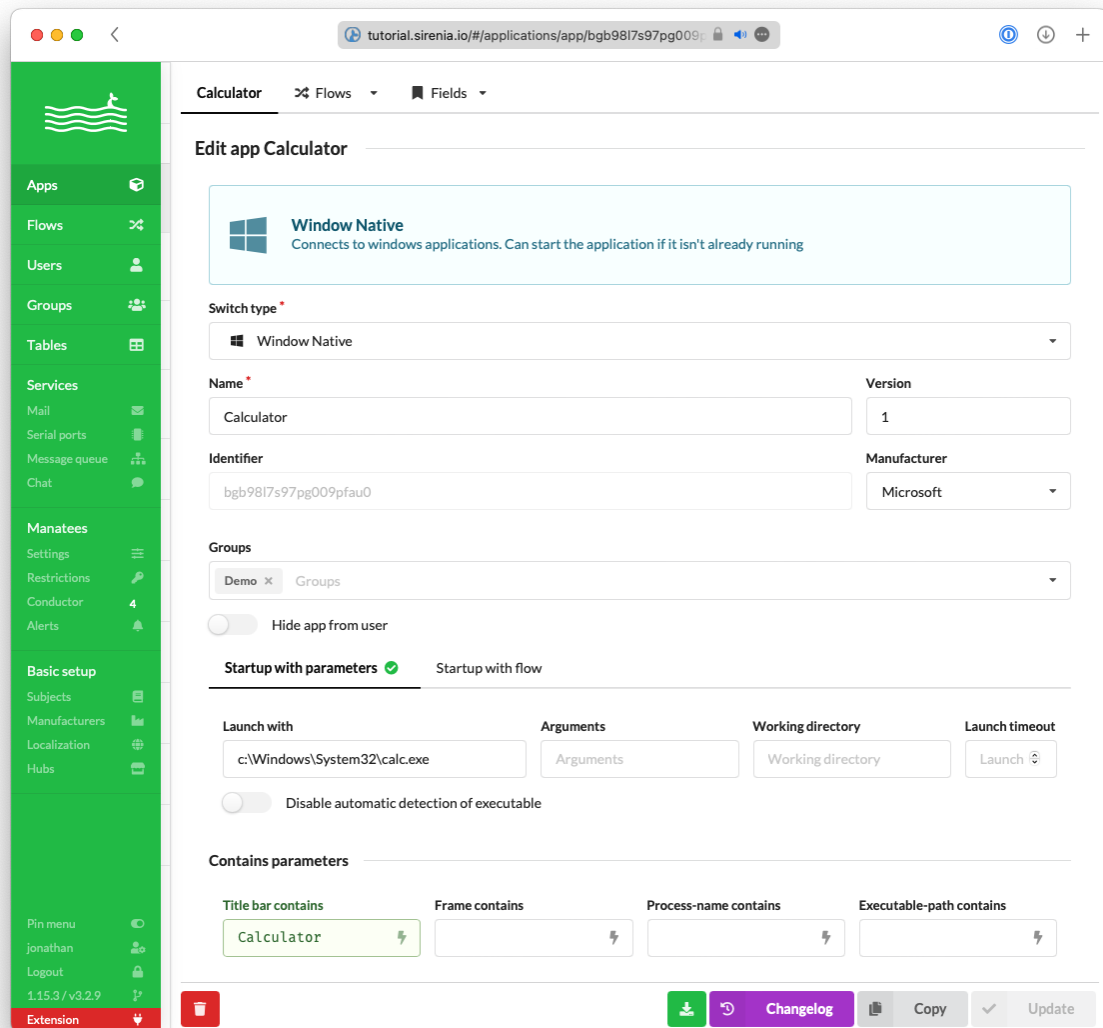


Figure 8: Application configuration

Applications are defined by a name, a type and information about how to recognise and possibly launch the application. The type determines which driver implementation Manatee will use to interact with the application.

The startup and contains parameters are used respectively to launch the application (if needed) and to recognise the application if it is already running.

Types of applications

Manatee can drive both native, web and java-based applications as well as handle proxies for actual context participants. Each type of application is handled by a specific driver and some requires extra components to be installed, e.g. a Chrome extension for the Chrome browser.

Standalone

We have the following drivers available for standalone applications;

- The **native** driver which are able to drive Windows native applications. These are normally applications written specifically for Windows and are accessed using a native accessibility API.
- The **Java** driver uses Java accessibility APIs to interact with Java applications. It has support for Swing/AWT applications and JavaFx embedded in Swing/AWT.

Web-based applications

For web-based applications we have support for the following browsers:

- The **Chrome** driver uses an extension to drive web-pages/applications shown in the Chrome browser. It is available at the Chrome store.
- The **Firefox** driver does the same as the Chrome driver but for Firefox. It can be downloaded from [here](#).
- The **Edge** driver which again is a driver for the Edge browser - available from the Edge Add-ons store.
- The **IE** driver interfaces with Internet Explorer to be able to automate web-pages shown here. It does not require any extensions/add-ons.

Manatee (the driver platform) also has some built-in browsers:

- The **embedded IE** browser runs an Internet Explorer browser-engine in a built-in window. It requires Internet Explorer to be installed on the host-machine and will use the version (depending on registry settings) and settings of IE installed but will display new window chrome with greater support for hiding the location bar etc.
- The **embedded Chrome** browser runs a Chrome (Chromium-based) browser engine. It does not require Chrome to be installed as Manatee includes the Chromium-runtime when installed. It can as with the embedded IE be configured to not display a location bar and will re-use cookies

or other state between launches making it better suited than using a regular to some forms of automation.

Furthermore we also have;

- The **headless** driver which does not attach itself to any running applications and is mostly used for running flows outside of the context of any applications.
- The **ContextParticipant** driver which is used to integrate with CCOW-compatible (via our external APIs) applications.

It is possible to add more drivers by way of Manatee plugins, so if there is a special need we can deploy support for new application types dynamically.

Creating and editing applications

An application is created by entering the required information in the new application input form. Each type of application has a specific set of fields that need to be filled out.

Launching or attaching to an existing application

For a web-application the section containing the parameters for launching is simply the url and (in some cases) whether to open a new tab or a new window. Similarly the information needed to detect an already existing instance is the url (or title in some cases).

The screenshot shows a form for configuring a web application. It has two tabs: 'Startup with parameters' (active, marked with a green check) and 'Startup with flow'. Under the active tab, there are three main sections: 'Launch with', 'Open page in', and 'Arguments'. The 'Launch with' section has a text input field containing 'http://sirenia.eu'. The 'Open page in' section has four radio buttons: 'New tab' (selected), 'New window', 'Incognito window', and 'App window'. The 'Arguments' section has a text input field. Below these is a section titled 'Contains parameters' with two sub-sections: 'URL contains' and 'Process-name contains'. The 'URL contains' section has a text input field with 'sirenia.eu' and a lightning bolt icon. The 'Process-name contains' section has an empty text input field and a lightning bolt icon. Two red annotations with arrows point to the form: one points to the 'Launch with' input field with the text 'The url to use when launching the application', and the other points to the 'URL contains' input field with the text 'The url to match existing web-browser instances against'.

The url to use when launching the application

Startup with parameters ☒ Startup with flow

Launch with

Open page in ☒ New tab ☐ New window ☐ Incognito window ☐ App window

Arguments

Contains parameters

URL contains ⚡

Process-name contains ⚡

The url to match existing web-browser instances against

Figure 9: WebApplication details

For a native- or java-based application more information is needed to launch the application as shown below. Detecting an existing instance can be done by different predicates:

- *Title bar* will match the current title of the application against the regular expression given, the same string as returned by `Window.title`
- *Frame* will match the class of the window frame, the same string as returned by `Processes.current.class`
- *Process-name* will match the name of process and may include arguments given to the process, the same string as returned by `Processes.current.commandLine`
- *Executable-path* will match the full path to the executable, the same string as returned by `Processes.current.fileName`

The predicates are joined in a conjunction meaning that the more predicates / fields that have input the more specific the match is. All fields accept regular expressions.

Launching an application via a flow

You can also choose to start an application with a flow. This flow is run in headless mode meaning that no application is available so you cannot interact with fields etc.

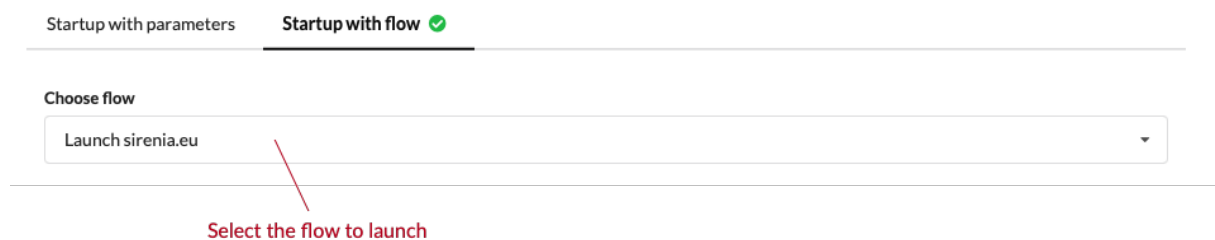


Figure 10: Launch an application via a flow

In the flow you can write some code that produces an application by:

1. Returning a `string` which will be interpreted as an url to launch (for web-applications) or a command to be executed for other types of applications
2. Launch the application in the flow and return an object containing the properties:
 - `hwnd` a window handle for the main window for the process
 - `pid` a process identifier for the process you have launched
3. Or return an object which will be used to launch the application. It should have the following properties:
 - `command` a command to run and
 - `arguments` arguments to a command
4. Return a `ProcessProxy` object designating the process launched.

5. Return an error indicating the launch failed.
6. No return value is interpreted as a successful launch and Manatee will try and attach.

Groups

You can add the application to as many groups as you would like. The more groups you add the greater number of Manatees will be able to use the application. The groups found on an application should be a super-set of the groups of all of its flows - in other words; do not add a group to a flow without its parent application also having this group.

Fields

Fields are used for interacting with the UI of an application. A field represents an element in the UI, a button, a text-field, a panel etc. Each type of field can be interacted with similarly to how a user can interact with the field. A button can be clicked, a text-field can have text written into it and text extracted from it and so on.

Fields are identified either by a *path* or a *screenshot* of the field. The screenshot is simply used to identify the field visually and should only be used if the field cannot be identified by a path. Paths are more flexible and robust with regards to changes in the UI of the application.

A path is a string which identifies the given field by containing instructions on how to traverse the tree that is the structural model of the UI. UIs are normally hierarchical in nature and structurally composed like a tree (e.g. the DOM of a web-page). The window is normally the root of the tree and it will contain a few panels as its children which again will contain other panels or UI elements directly. A path is thus a way to navigate from the root of the application (the outermost window) to the field we are interested in. An example:

```
1 /OuterPanel1/OuterPanel2/OkButton
```

This path specifies that we are ultimately looking for an `OkButton` element which is contained in an `OuterPanel2` which itself is contained in an `OuterPanel1`. The elements along the path are matched on multiple criteria e.g. `OkButton` might be the actual text shown on the button or the class/-type of button used.

The path editor (the text field in which the path to your field can be seen and edited) applies some formatting to make it easier to parse what is going on in the individual path elements.

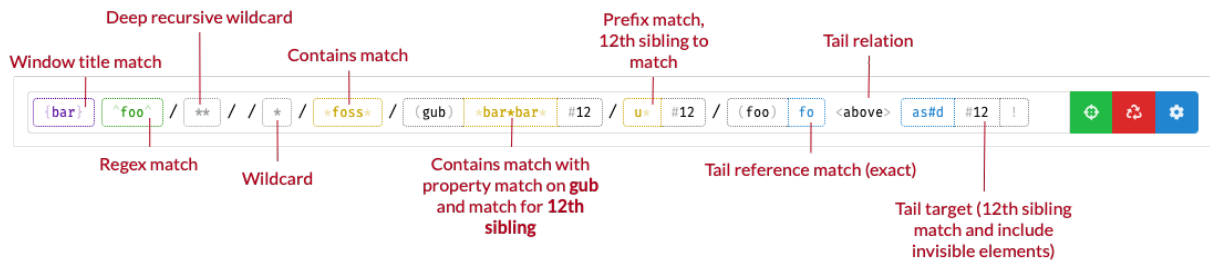


Figure 11: The path editor

Hovering over an element of the path will display some information about how that particular element is matched as well. You can read more about fields and how the individual elements match the nodes in the UI tree in the Manatee manual.

You can edit the path and its elements directly but if you *right-click* on an element you will get a menu in which you can edit an element using predefined controls. There are three distinct editors. The first one here is the editor for the window-match element (the first optional element in a field path).

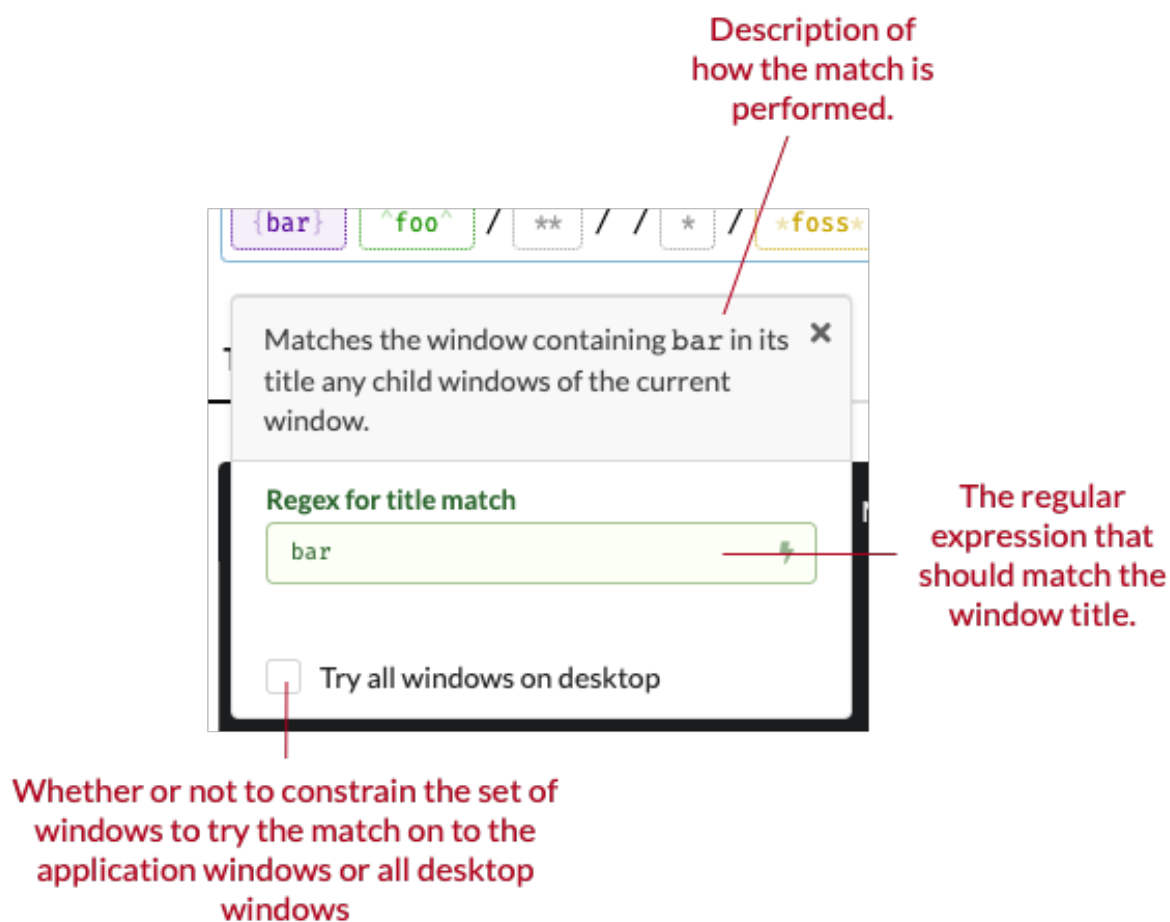


Figure 12: The editor menu for a window element

Secondly there is a menu for all the intermediate elements where you can choose how the match should proceed:

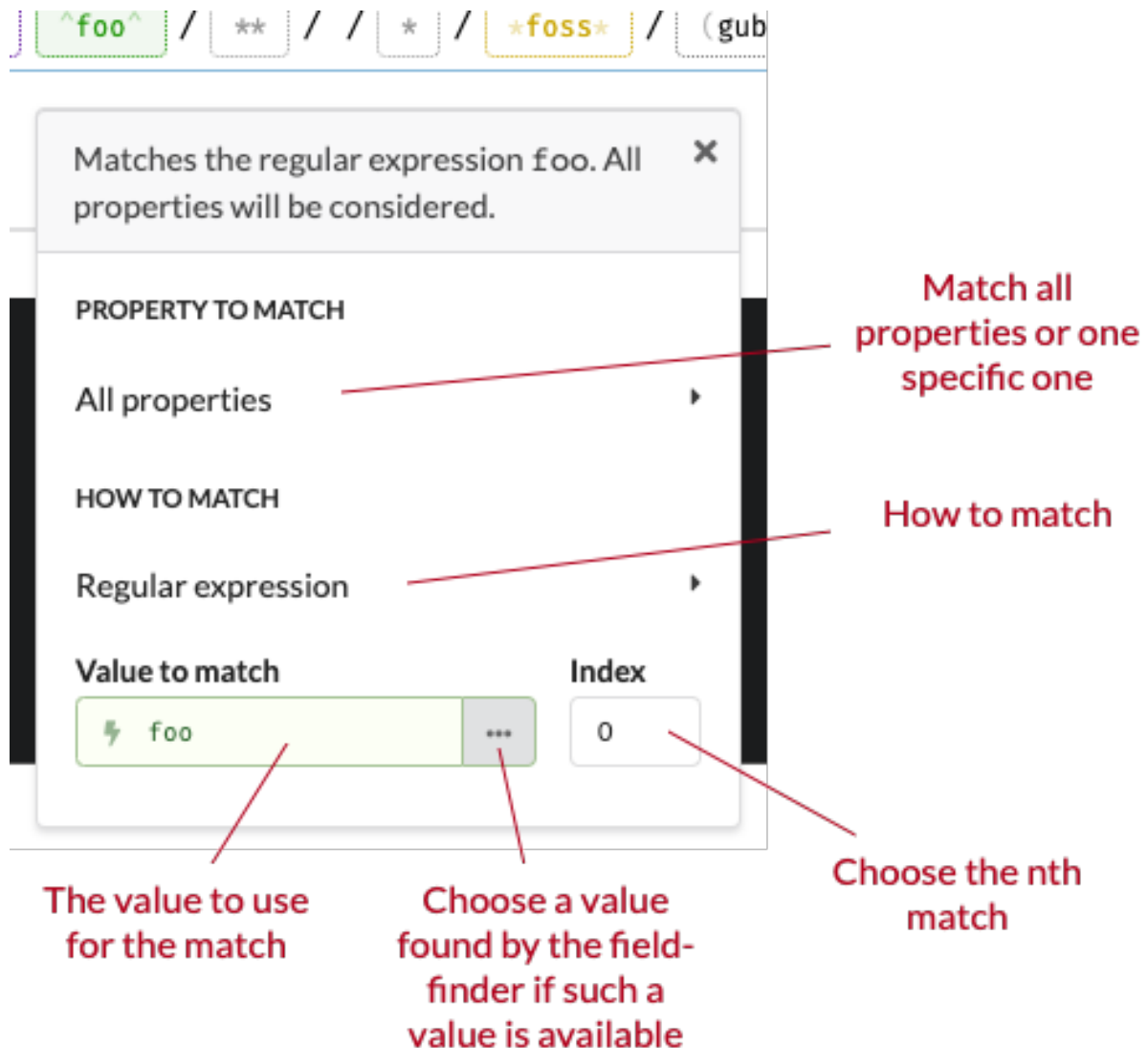
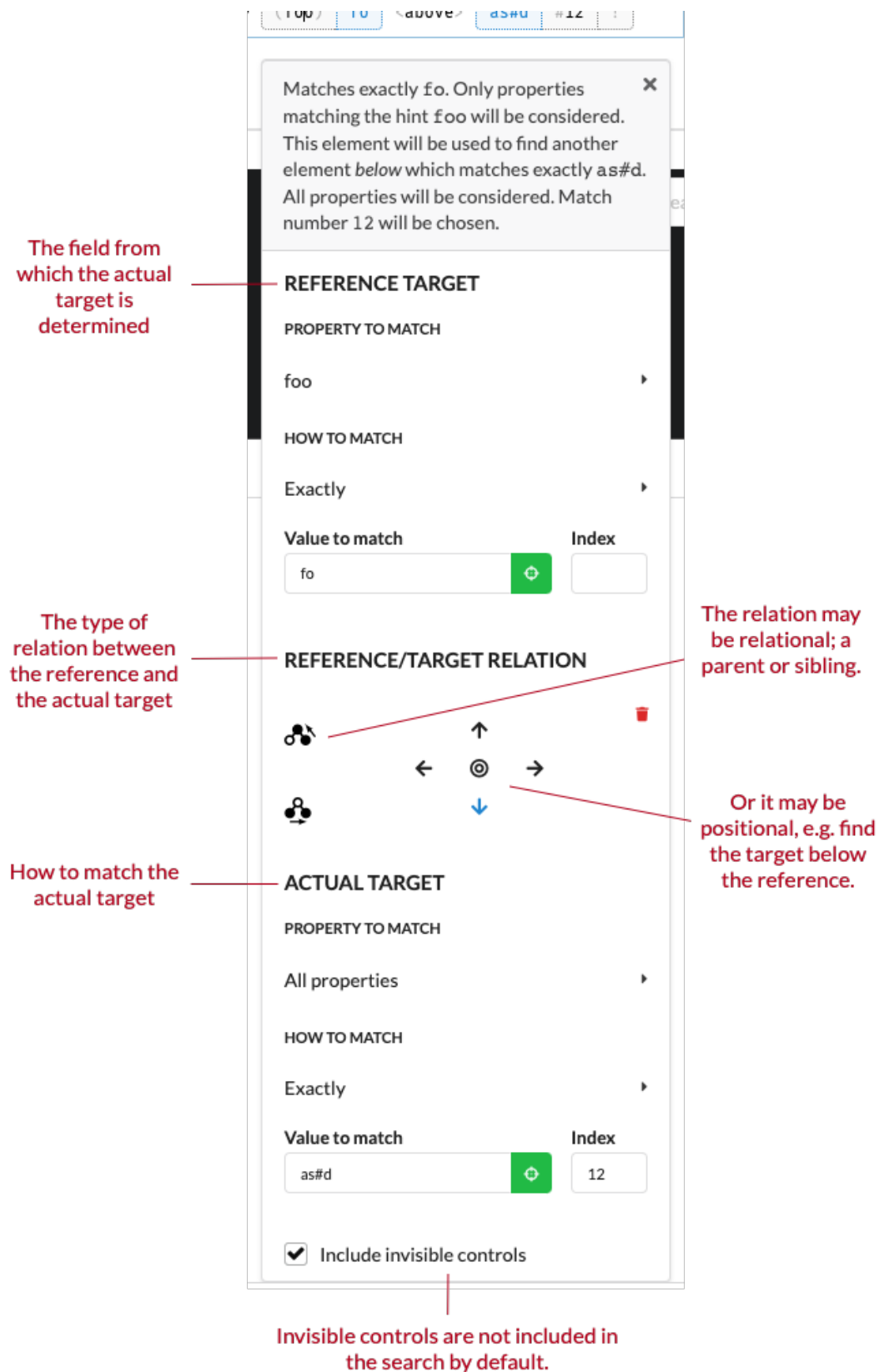


Figure 13: The editor menu for a normal element

And lastly there is a menu for the “tail” path element. The last element may contain a relation between a reference and a target element so it is more complex:

**Figure 14:** The editor menu for the last/tail element

Creating fields

It is not normally necessary to manually construct these paths as Cuesta will connect to a locally running Manatee instance and try to figure out the path to a given field when you use the field-finder functionality in Cuesta.

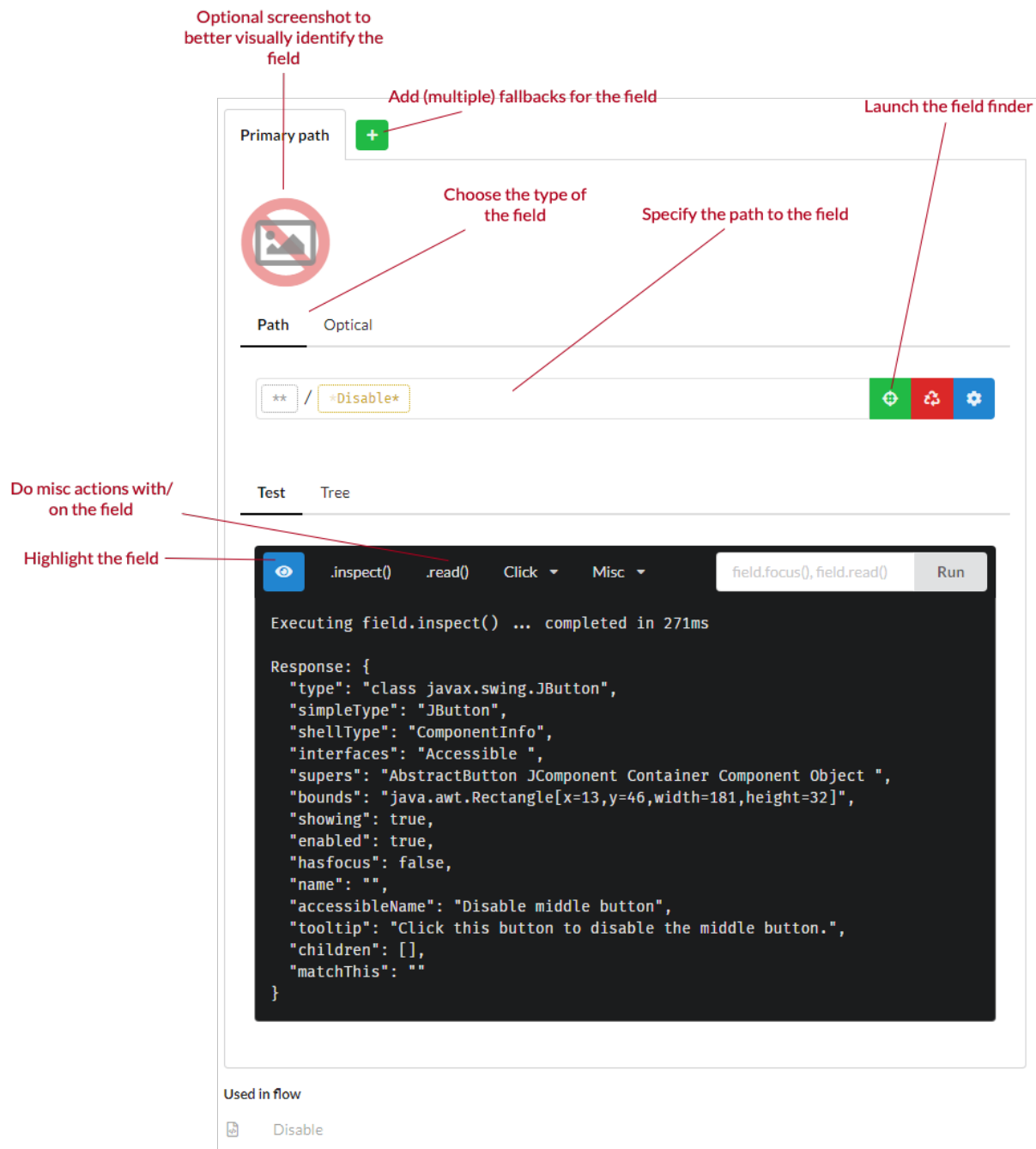


Figure 15: The field creation form

Then you will simply point at the element in question and Cuesta will input the path for you. If the element is in a dialog that needs to be activated after the field finder is activated, you can hold down ctrl, which will pause the field finder while you activate the dialog.

Instead of clicking on the app element, you can also press spacebar to select the currently highlighted element.

Tree view

You can also use the tree view to explore the UI structure if the application and select an appropriate field to get a path for it.

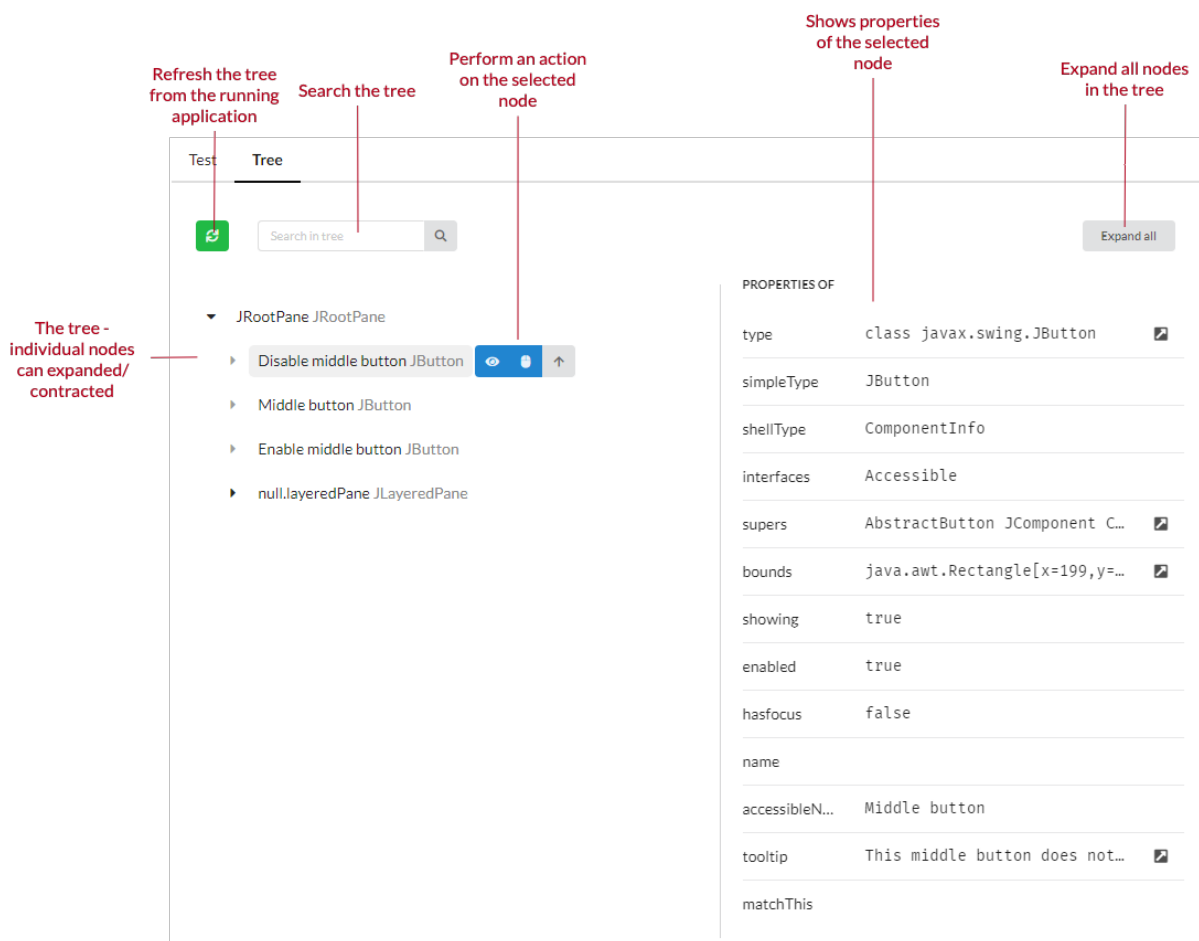


Figure 16: The tree view

Using a field in a flow

Once defined the field may be used in one or more flows for the application. For example the following code snippet clicks the button designated by the field name “OkButton”.

```
1 Fields['OkButton'].click();
```

A more complex example could be a field which represents a table:

```
1 // Inspect (i.e. get a JSON serialized version of) the table
2 var table = Fields['MyTable'].inspect();
3 // The `table` variable is an object serialized according to
4 // the JSON serialization document
5 var rows = table.rows;
6 var someCell = table.rows[10]['Header1'];
7 // There is also functionality for clicking in a certain cell
8 table.clickCell('row1', 'Header1');
```

Flows

A context manager can both facilitate synchronisation and support a form of RPC by letting participants in a context session expose *actions* which other participants can ask the context manager to run. Both *state* synchronisation and *action* invocation are central concepts for Manatee. A **flow** is a single *executable script* which can interact with UI of an application or perform other functionality as described in the modules documentation. A flow can be either a *state*, an *action* or a *module*.

State flows

A *state* flow represents functionality to set or get the state of an application for a given *subject*. Both a *get* and a *set* version for the same subject is required for the state synchronisation to take place.

An example: Getting an application *A* and application *B* to synchronize state using UI integration with the subject *foo* can be accomplished by creating two state flows for each application, one for *getting* the current value of *foo* and one for *setting* it. If the current value of *foo* in application *A* is stored in a textfield accessible with the name *fooA* then we'd perhaps simply write the *get foo* flow as follows:

```
1 Fields['fooA'].read();
```

and the *set foo* flow as:

```
1 Fields['fooA'].input(Value);
```

Similar flows may be applicable for application *B*.

Manatee will then take care of extracting the value and setting it when necessary. The state of *foo* will be extracted and stored in the context manager regardless of whether there are other applications that must be synchronised.

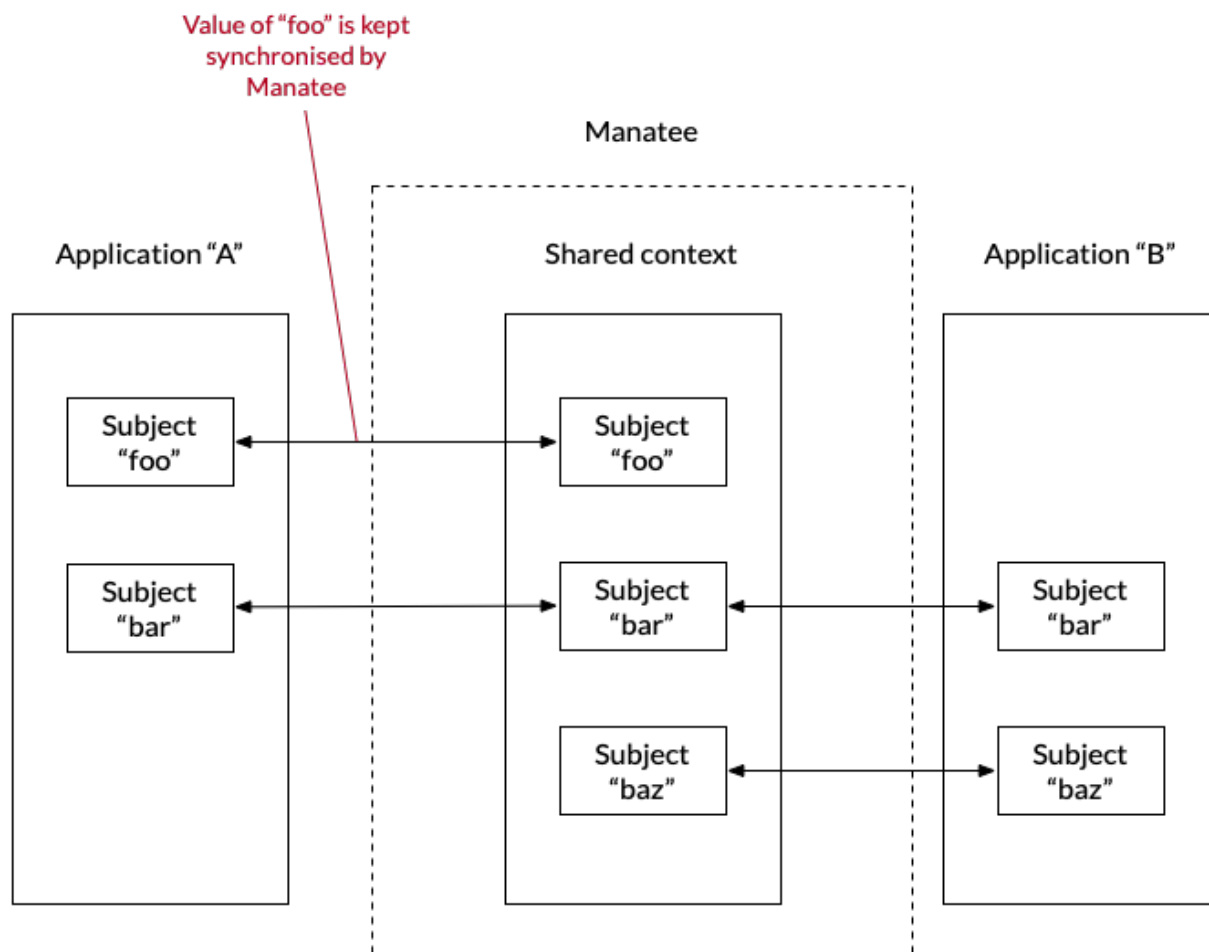


Figure 17: Application synchronisation example

The shared state (the current value of all synchronised) subjects are available to all flows of type *state* and *action* in the *Inputs* map. The current value for the subject of *state* flow is also made available in the special *Value* variable (as seen in the *set foo* flow above).

An application does not need to support all subjects made available in the shared context but may decide on supporting a subset only.

Mapping flows

A mapping flow is used to map the existing values in the shared context to new values through a transformation performed in a flow. An example could be that you have a patient identifier in one format that needs to be translated into another format for another application. In this case you can utilise a mapping flow to do the mapping automatically. This is illustrated below.

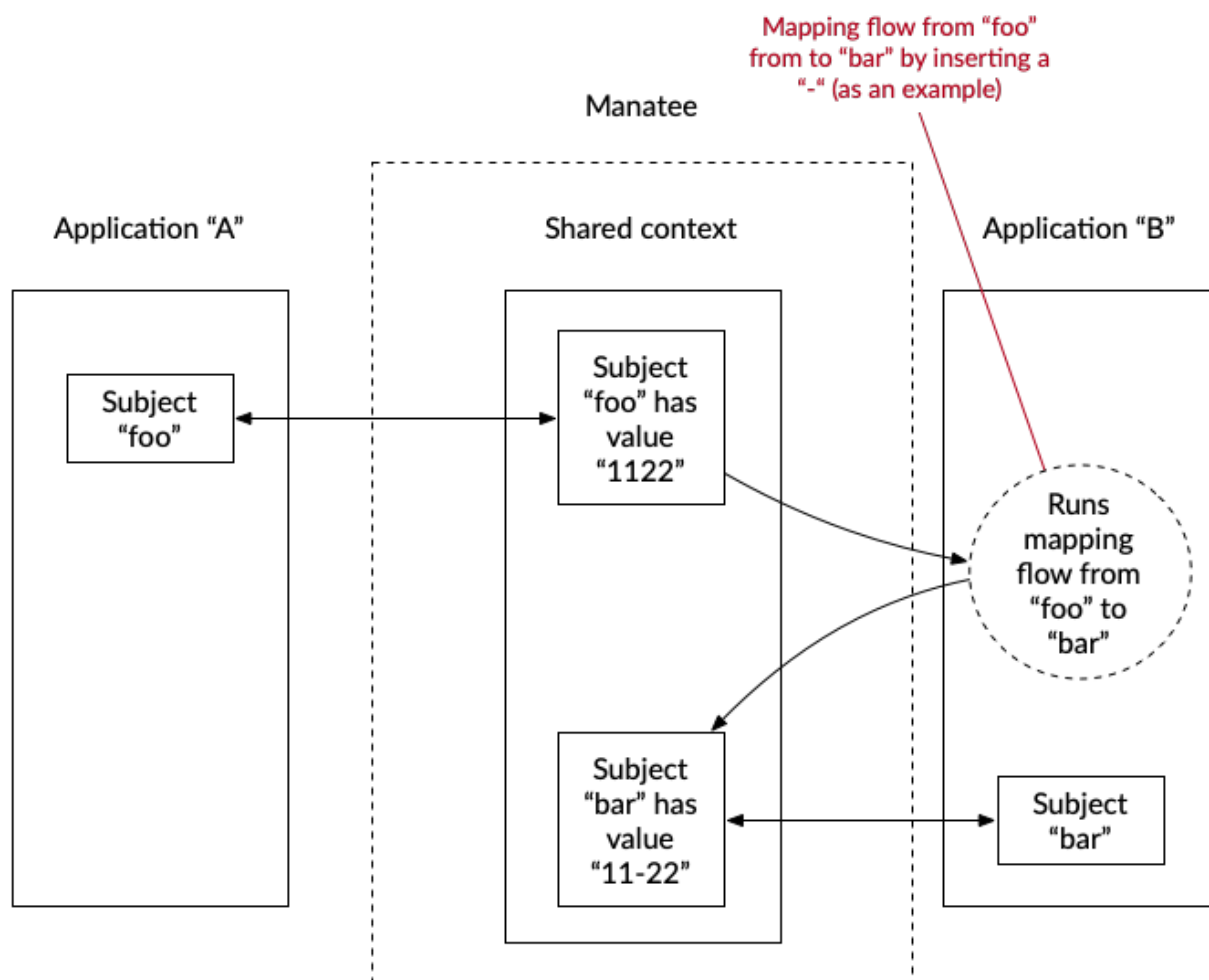


Figure 18: Mapping a value in the shared context between two formats

A mapping flow operates on *one subject only* so you need two mapping flows for a two-way mapping.

Action flows

An action flow resembles an RPC invocation either internally in an application or between applications. A flow always belongs to an application and will be run in the context of that application. It

cannot be run unless the application is running and attached to a session in Manatee, so Manatee will always attempt to connect to an existing application before starting a new instance when asked to run an action flow for a given application.

Action flows can be started in numerous ways. Two of them are either by a trigger or by using a flow-menu which appears in an application in a context session when the user right-clicks while holding the alt key down.

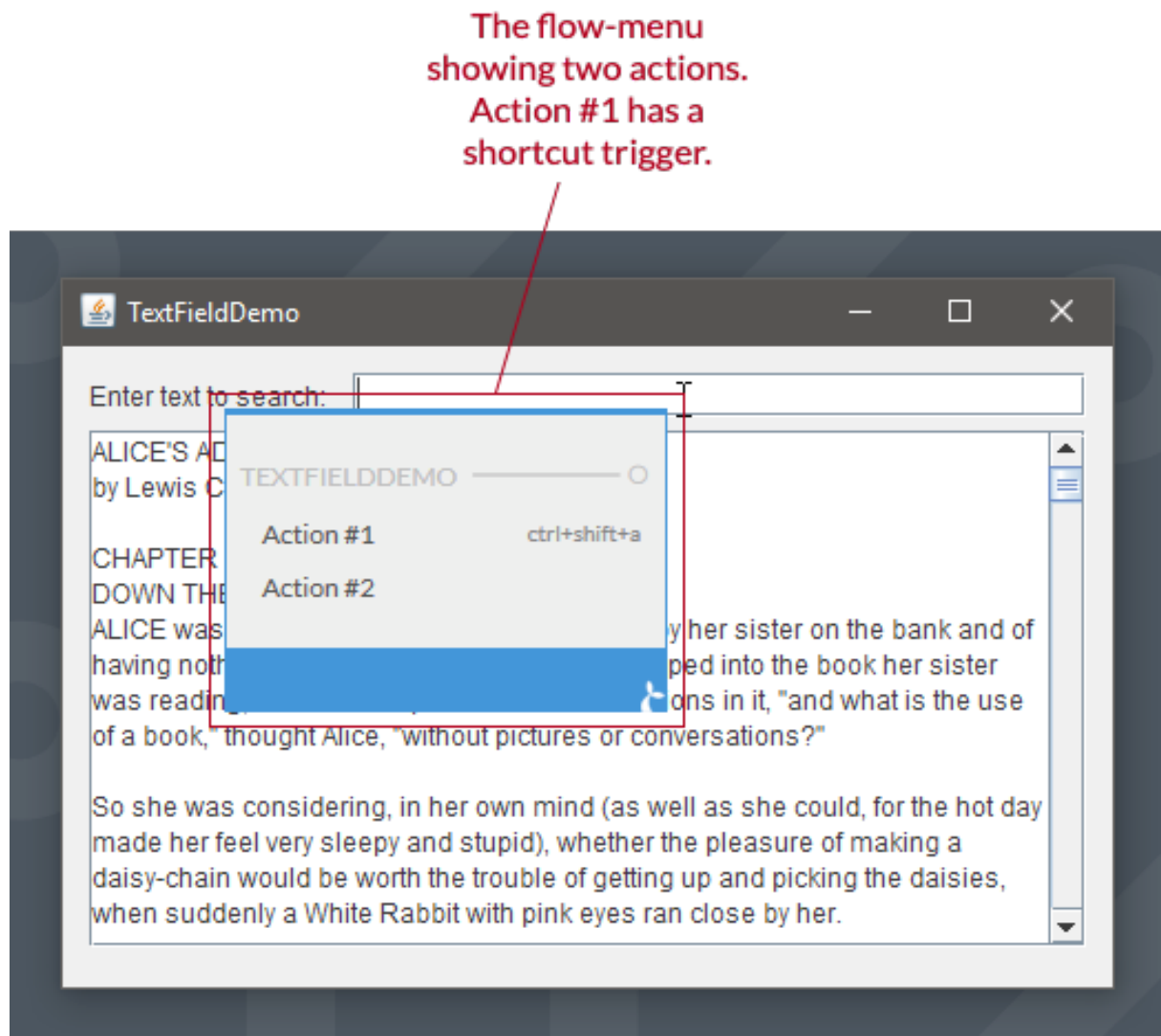


Figure 19: The flow-meni

Action flows can be used for automation and for fire-and-forget frontend integrations. It is also possible to create action flows, which spans multiple-applications to transfer data or to automate multi-application workflows.

Filtering flows shown in the flow-menu

The flow-menu displays flows, which are relevant for the application and location within the application in which it was invoked. It does this in two ways;

The first filtering is done by extracting values for any *subject* from whatever the user might have clicked. If the flow has inputs these must be matched with the *subjects* and values extracted from the context of the click. The purpose of this mechanism is to ensure that the flow has the correct inputs, but it also provides a neat mechanism for *open-ended integration*. If, for example, we've defined a subject for a patient identifier then for a flow which switches patient in an EHR we can define this as an input. Now this flow can be everywhere where we can extract patient identifiers, e.g. you'll be able to switch the EHR to patient found in an Excel spreadsheet or any other type of application where a patient identifier can be extracted.

An example. If a subject is defined for a patient identifier, this subject can be used in a flow, which switches patient in an EHR. This flow can be initiated from everywhere where we can extract a patient identifier, e.g. to switch the EHR to a patient found in an Excel spreadsheet.

The second filtering is done such that anchored flows are only shown if they're the anchor matches the location clicked.

Module flows

Module flows are flows which contain some shared functionality which is made available to other (state and action flows). It is a mechanism by which functionality (code) can be shared between flows. A module flow should follow the following template:

```
1 // Define some functionality that can be useful for other flows
2 var isEven = function (n) {
3   return n % 2 === 0;
4 }
5
6 // The `exports` object should be populated with the functions
7 // and variables that should be exported from the module flow
8 exports.isEven = isEven;
```

There are more details on *module* flows in the flow documentation.

Built-in modules

A flow may interact with the user interface of the application in which it belongs by using any of the defined fields. A number of built-in modules are also made available for a flow. The details and exact nature of these are described in the Manatee v1.29 page for each version of Manatee.

The modules cover a wide range of functionality from inspecting and manipulating the OS windows to accessing external services through [http](#) and creating Excel documents. The flow documentation contains numerous examples of their usage.

Triggers

Triggers are *hooks* which determine when a flow is run (if not invoked manually). It is possible to have multiple triggers for a single action flow.

Once a trigger is added to a flow it will run the flow when the trigger conditions are met on the Manatee instances which the flow is made available to. It should be noted that adding a trigger can thus interrupt a user of s/he is using the machine running the triggered flow and is therefore recommended primarily for unattended machines. In some cases it will make sense to trigger a flow in an attended situation though.

Examples of triggers are;

- When a *window* with a certain title is opened.
- When a *file* in a certain folder is created.
- When the user presses a certain *hot-key* combination.

See more info in the triggering flows section as well as the documentation for each type of trigger.

Creating a flow

A flow must have a name, a type and some auxiliary information depending on the type of flow. For *state* flows you must specify its *subject* and whether the flow *gets* or *sets* the state for the subject.

The flow to be created is a **state flow**

Create Flow

Name *

Name

Type ☒ State ☐ Action ☐ Module

Action

Subject

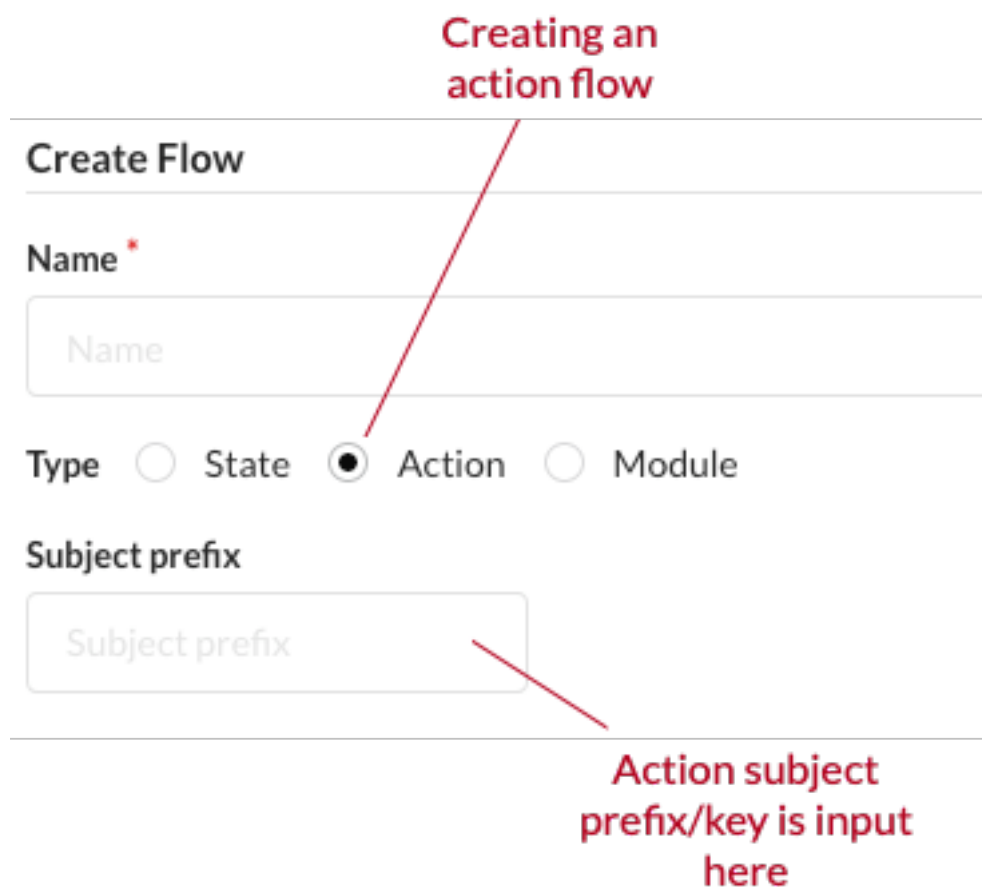
Select whether this flow sets or gets the state value for the defined subject

State subject/key is input here

Maintainer

Figure 20: Creating a state-flow

For an *action* flow the *subject* prefix should be set. This serves as an identifier for the flow.



The image shows a web form titled "Create Flow". It has a "Name" field with a red asterisk, a "Type" section with radio buttons for "State", "Action" (which is selected), and "Module", and a "Subject prefix" field. Two red annotations are present: one pointing to the "Action" radio button with the text "Creating an action flow", and another pointing to the "Subject prefix" field with the text "Action subject prefix/key is input here".

Creating an action flow

Create Flow

Name *

Name

Type ☐ State ☒ Action ☐ Module

Subject prefix

Subject prefix

Action subject prefix/key is input here

Figure 21: Creating an action-flow

For both *state* and *action* flows it is useful to set a maintainer as well. Cuesta can be configured, so only the maintainer can edit the flow.

Groups

A flow will not be available for a Manatee unless it has at least two groups; one that is the same as the primary (production) group of the Manatee instance and one that is in the set of all the (AD) groups of the user (combined with the username and the name of the machine). Both of these groups must be applied to the application as well.

Editing a flow

Once a flow is created the functionality or code of the flow can be edited as text or in the blockly editor. See the flow documentation for details of how to write flow code and examples as well as

documentation on the available modules.

Code

The code of a flow can be edited directly in the code editor. The language we support is Javascript.

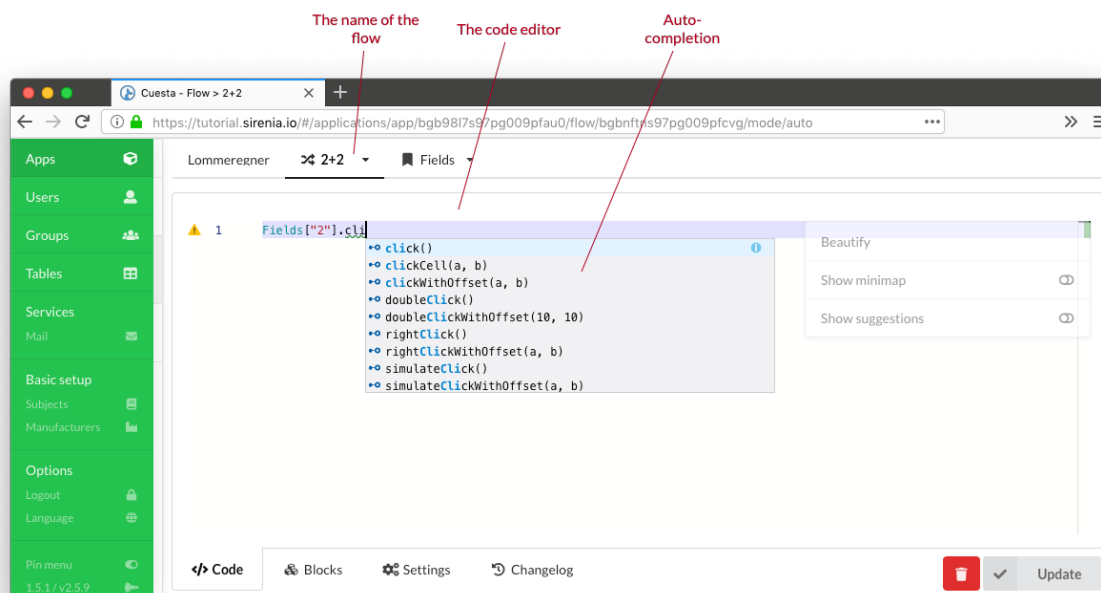


Figure 22: The code editor

The code editor has auto-completion and other advanced features. Hit the **F1** key to show a list of editor functions and their assigned keyboard shortcut (if any). The editor analyzes your code as you type and points out any problems it finds.

@vimeo

As can be seen in the video, detected problems that you are ok with can be ignored with the `// disable-check` comment.

If the editor is running in the manatee embedded browser or in an external browser with the Sirenia extension, the flow can be run with **F10** or debugged with **F11** straight from the editor. If any code is highlighted, only the highlighted code is run.

Javascript

When the 'modern' javascript option is selected while creating a new flow, some additional modern language features become available. They include the following:

- Arrow functions

```
1 var sum = (a, b) => a + b
```

- Block scoped constant variable declarations. This expresses your intent to never change a value.

```
1 const fileName = 'user-list.csv';
```

- Block scoped mutable variable declarations. This expresses your intent that the value may change.

```
1 let fileName = 'user-list.csv';
```

- Template string literals. A powerful mechanism for formatting strings.

```
1 function formatFilename(x) { return `user-list-${x}.csv`; }  
2 const multiLine = `Line 1  
3 Line 2`
```

You can also check out this [guide to ES6](#) for some idea of what has changed from ES5. *Be aware that we don't support all ES6 features.*

Blockly

Blockly is a visual programming tool which uses blocks that combines like a puzzle to define complex functionality. Our modules, field interaction etc. is available as pre-defined blocks. Running a sub-flow (i.e. invoking a flow from another flow) is also possible. Blockly can be used to construct the same flows as done via the code-editor and it is possible (to a certain extent) to switch back and forth between the code editor and blockly.

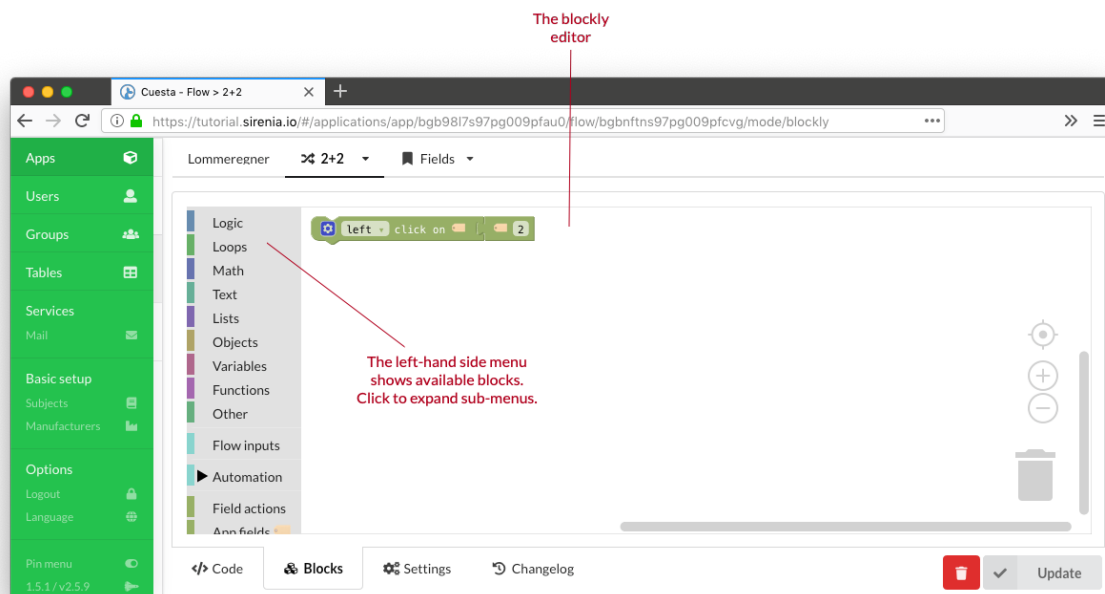


Figure 23: The blockly editor

The simplicity of blockly also makes it possible to setup a workflow where you have dedicated developers constructing the “building blocks” for your automation needs in the code editor while the domain specialists use blockly to piece together actual workflows without writing a single line of code.

Importing flows

It is possible to drag Microsoft Steps Recorder or UiPath projects onto the code editor to import/convert the automations into an existing or new flow.

Anchors

Anchors are used to filter where a flow can be activated using the flow-menu. Adding an anchor is simply done by selecting which field(s) to use as anchors. When added the flow will only show in the flow-menu when activated within one of the fields given. Defining no anchors means that the flow will appear regardless of where the menu was activated.

Inputs and outputs

A flow can accept inputs and produce outputs. Each *input* can be typed to any of the configured *subjects*. This is also used for deciding which flow to show in the flow-menu; only flows which can have

their inputs matched with extracted content is shown. The value of the input can be accessed in the flow using the `Inputs` map, e.g.:

```
1 var myInput = Inputs['myInputSubject'];
```

Outputs from a flow can be any root scope defined variable. If we have the following flow;

```
1 var myValue = 'foo';  
2 var myOtherValue = Windows.topMost.title;
```

then `myValue` and `myOtherValue` can be made outputs in the Settings tab of the flow.

Triggering flows

A flow can be setup to run automatically by using a trigger. Triggers include:

- *Cron* which runs flows at certain pre-defined times.
- *Window* which runs flows when certain windows are shown or hidden.
- *Filesystem* which monitors a folder and runs a flow when files are added, removed or changed
- and several more ...

The group filtering still applies to flows with triggers - thus it is only Manatees which match the groups of a flow that can be triggered.

The **hotkey** trigger has been added with **ctrl+shift+alt+u** as the triggering combo.

Add a new trigger to a flow. Choose from the available listed.

Hotkey

Press ctrl+shift+alt and u to trigger flow

Hotkey modifier key

✕ ctrl

✕ shift

✕ alt

Hotkey modifier key

Hotkey key

✕ u

Hotkey key

☒ Enabled

New Trigger

Choose a trigger to add

External trigger

[Cron](#) [Hotkey](#) [Windows](#) [Table](#)
[Keyboard](#) [Filesystem](#) [Intercept event](#)

Context-based trigger

[Change](#)

Lifecycle trigger

[Attached](#) [Detached](#) [Manatee started](#)
[Navigated](#)

Service trigger

Figure 24: Adding a trigger to a flow

Groups

In order to decide where a certain application or flow can be run we use the concept of a *group*. A *group* in Cuesta is simply a label which can be attached to e.g. an application or a flow. The group itself can contain a few properties which determines how the attached entities behave in Cuesta. An example is adding prompts to be displayed when you add e.g. a flow to a group. Or to disable editing of all flows in a certain group.

Prompts for adding/removing entities from group

☐ Prompt for confirmation before adding this group

Prompt on add message
 Are you sure you want to add item PRODUCTION?

☐ Prompt for confirmation before removing this group

Prompt on remove message
 Are you sure you want to remove item PRODUCTION?

Group flow settings

☐ Disable code autosave in flows associated with this group

☐ Disable code editing in flows associated with this group

Flow-specific settings

GROUP_CONTAINS_STATS

0 USERS	4 APPS	4 FLOWS
User	App	Flow
	Cosmic	Notepad / Demo
	Notepad	Sirenica / Simple we...
	Sirenica	Lommeregner 1 / 2...
	Lommeregner 1	Sirenica / Rens excel

Which users/applications/flows are part of this group

Figure 25: The details of a group

Manatee uses groups to determine which flows are applicable for the specific instance of Manatee. It uses a configurable primary (production) group and a list of secondary groups to do this. The primary group is configured and can be changed by the user. The secondary groups are a union of;

- the (AD) groups of the current user,
- the name of the current machine, and
- the username of the current user.

A flow *must* be in both the primary and at least one of the secondary groups to be available for a given Manatee instance.

The concept of groups thus makes it possible to control in which Manatee instances certain functionality (flows) are available.

Tables

Tables is an internal datastore in Kwanza which can be accessed from Manatee. Typically tables are used for mapping tasks but they are in essence a general purpose data store.

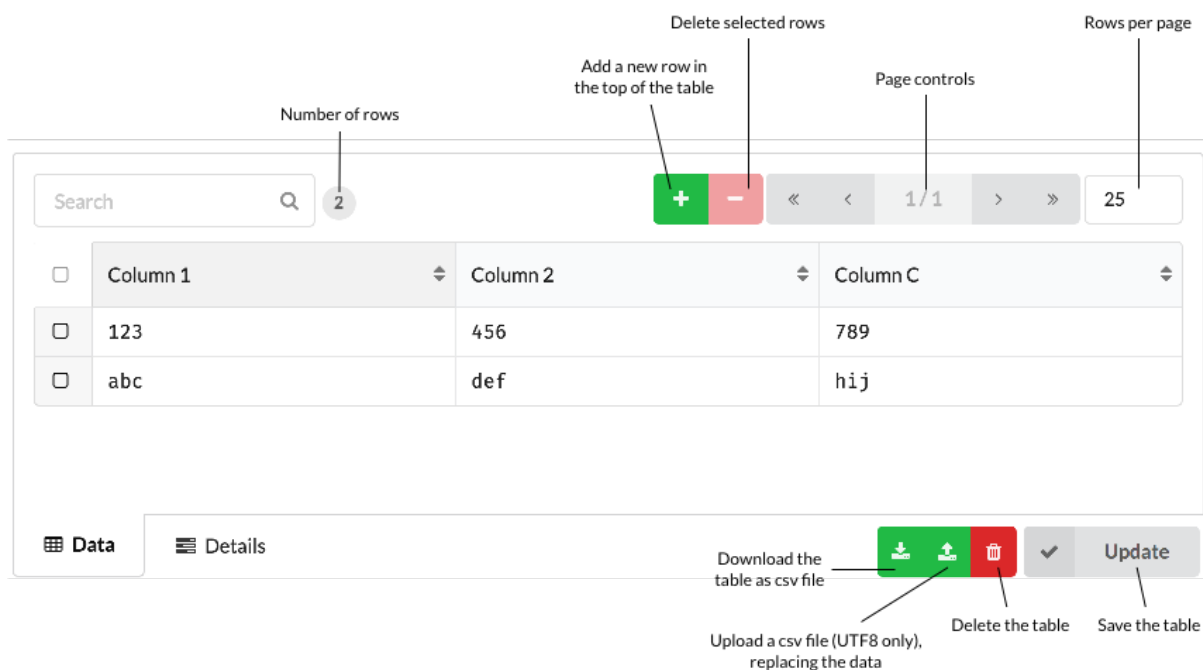


Figure 26: The table editor

Creating and using tables

A table *must* be created in Cuesta before it can be accessed from Manatee. It requires a unique name. Once a table is created it has an interface like a spreadsheet in Cuesta. You can move around using the keyboard and change content of cells, add new rows or columns. Again, press `ctrl+h` for the relevant keyboard shortcuts.

You can also upload or drag-and-drop a CSV file onto a table and it will get uploaded and stored (once you press the update button).

It is also possible to search/filter using the controls at the top of the table.

Services

External services are defined by adding their configuration to Cuesta. Currently we support an IMAP/SMTP and Exchange mail services, serial port services and message queue services. For details see the services document.

Mail

Creating a mail service is simply done by entering the details (e.g. IMAP or Exchange server and credentials) into the service details form. Once a mail service is created it can be used to trigger flows (when a mail matching certain parameters is received) or for sending mail from a flow.

IMAP

Edit Mail

Service name *

MyImapService

Service key *

myImapService

Mail service type ☒ IMAP ☐ Exchange Server**IMAP**

Hostname or IP *

imap.mailserviceprovider.com

Port *

800

Username *

john

Password

••••••••

SMTP

Hostname or IP *

smtp.mailserviceprovider.com

Port *

900

Username *

john

Password

••••••••

Figure 27: Creating/editing an IMAP mail-service**Exchange**

Four separate authentication modes are supported when integrating with exchange servers: basic, oauth delegated, oauth app-only and windows login.

Edit Mail allUsers

Service name *	Service key *
<input type="text" value="allUsers"/>	<input type="text" value="allUsers"/>
Mail service type <input type="radio"/> IMAP <input checked="" type="radio"/> Exchange Server	
Exchange Server	
Exchange Web Service address	
<input type="text" value="https://outlook.com/EWS/Exchange.asmx"/>	
Authentication type	<div>Basic Auth Delegated OAuth App-only OAuth Windows login</div>
<div>Exchange OAuth Your Exchange server administrator needs to register Manatee as an app in the Azure portal and set it up for either delegated or app-only authentication. A guide can be found here</div>	
Application ID *	Tenant ID *
<input type="text" value="16aedd0f-8d7b-471b-9fff-56e05eaa516b"/>	<input type="text" value="cbb44be3-21eb-484b-81dc-dbcf7241ae7d"/>
Client secret *	
<input type="text" value="....."/>	
<input type="checkbox"/> Legacy mode (before 2010 SP1)	

Figure 28: Creating/editing an Exchange mail-service

Serial port

A serial port service specifies how to connect to a device attached to the local host through a serial port connection. To do this it needs to know how to choose the serial port to connect to as well as the various standard connection parameters.

Port name

The hard part is helping manatee choose the right serial port to connect to. On different hosts running the manatee driver platform, the same type of device may be connected to different COM ports. If the ports are shown in windows with just `COM1/COM2`, this can be a challenge that requires some more

fine grained configuration work.

If the device is connected to the computer through a usb-based serial port, there will often be extra text in the port description in windows, which can be used by manatee to identify the port to connect to. For this reason, the port name field in the configuration interface supports regular expressions.

Encoding

The `encoding` setting does not need a value if the service is only going to be used for communicating binary data. It defaults to US-ASCII. Note that if the service is going to be used in a serial port trigger, the correct encoding must be provided as the trigger matches against the data in the form of decoded text.

Serial port

Service name *

My device

Service key *

my-device

Port settings

Port name (regexp) *

USB serial port

Baud

9600

Parity

None

Data bits

8

Handshake

None

Encoding

Unicode (UTF-8) (utf-8)

Stop bits

1

Figure 29: Creating/editing a serial port service

Chat

A chat service is defined by its connection parameters similarly to the mail service. For the current implementation using Slack two tokens must be provided:

The screenshot shows a configuration form for a Slack service. It includes fields for 'Service name' (Development Slack) and 'Service key' (devslacks). The 'Type' is set to 'Slack'. Under 'Slack settings', there are two token fields: 'App token' (xapp-1-A02V390N7M2-2980386792630-ffd19b233730bc1d516a3d85a36f280c9bb93f9054b8e43e2d9f) and 'Bot user token' (xoxb-78005313-29995364033-uoXibZNbEU2TYWZgB). Below the bot token field is a list of required scopes: channels:read, channels:history, chat:write, users:read, im:read, and npim:read.

Service name ^{*} Development Slack

Service key ^{*} devslacks

Type ☒ Slack

Slack settings

App token ^{*} xapp-1-A02V390N7M2-2980386792630-ffd19b233730bc1d516a3d85a36f280c9bb93f9054b8e43e2d9f

The app token is used to listen for new messages when the service is used as a trigger. You must have the `connections:write` scope on your token for this to work.

Bot user token ^{*} xoxb-78005313-29995364033-uoXibZNbEU2TYWZgB

The bot user token is used to read/write message from Manatee back into slack. You must have the following scopes on the bot token:

- `channels:read`
- `channels:history`
- `chat:write`
- `users:read`
- `im:read`
- `npim:read`

Figure 30: Slack configuration example

- an **app token** used for retrieving and listening for new messages in channels, and
- a **bot user token** used to send messages.

Conductor

The *Conductor* is a server-side component that presents a live-view of running Manatees. You can use it to query running Manatees, see which flows are executed in real-time and to administer individual Manatees.

You use it to define filters and then a list of matching running Manatees is shown. The list will update as Manatee exit or are started and thus provides a real-time view of the current Manatee “population” matching your filter.

Filtering Manatees

When you first go to the Conductor main page you’ll be presented with a list of all Manatees. In order to filter this view you can add filters in the “simple” view.

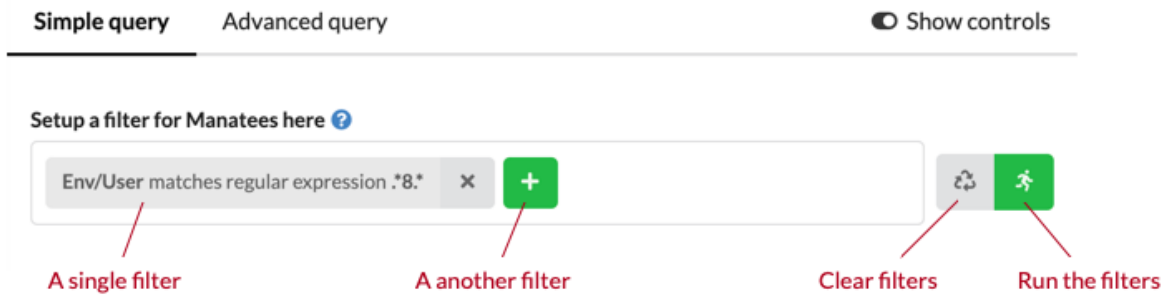


Figure 31: The simple filter editor

All filters must match in order for a Manatee to be displayed in the list below the filter definition. Simple filters consist of three parts:

- **A property to match.** Each Manatee reports its settings and misc environment information which can be selected.
- **An operator.** This can be equals, starts-with etc and determines how the value of the property is matched against the *target value*.
- **A target value.** This is the value that the property found on each Manatee must match in order to be included in the list shown.

Setup a filter for Manatees here ?

Env/Version equals v1.29 x +

Menu to select filter properties

Select property

Ver

Select operator

equals

does not equal

is less than

is less than or equals

is greater than

is greater than or equals

Enter value to compare

v1.29

User	Primary group	Manatee version	Running flow	Details
User 9	PRODUCTION	v1.29	Idling	...
User 8	PRODUCTION	v1.29	Idling	...
User 1	PRODUCTION	v1.29	Idling	...
User 9	PRODUCTION	v1.29	Idling	...
User 9	PRODUCTION	v1.29	Idling	...
User 7	PRODUCTION	v1.29	Idling	...
User 2	PRODUCTION	v1.29	Idling	...
User 7	PRODUCTION	v1.29	Idling	...
User 2	Machine 2	PRODUCTION	Idling	...
User 37	Machine 37	PRODUCTION	Idling	...
User 48	Machine 48	PRODUCTION	Idling	...
User 81	Machine 81	PRODUCTION	Idling	...
User 63	Machine 63	PRODUCTION	Idling	...
User 17	Machine 17	PRODUCTION	Idling	...
User 33	Machine 33	PRODUCTION	Idling	...
User 7	Machine 7	PRODUCTION	Idling	...
User 99	Machine 99	PRODUCTION	Idling	...
User 64	Machine 64	PRODUCTION	Idling	...
User 91	Machine 91	PRODUCTION	Idling	...

Manatees matching filter

100 Manatees Page 1 of 2 (50 per page)

< 1 2 >

Figure 32: Displaying a filtered list of Manatees

Each matching Manatee is displayed with;

- user name,
- machine name,
- primary group,
- version,
- running flow (display a flow if the Manatee is currently running it), and
- details.

You can use the details column to go a detailed view of a single Manatee.

If you want finer control over the filtering and display then you can go the “Advanced query” tab and enter queries in the *Manatee query language* (mql) directly.

MQL

In the “Advanced query” tab you have access to edit the query directly. The language in which to do this is a very simple form of structured query language. The interface looks as below:

The screenshot displays the advanced query interface. At the top, there is a text input field for an MQL query, labeled "Enter an MQL query to filter below". Below this field is a red arrow pointing to the text "Editor for query". The query entered is: `SELECT Env/MachineName, Env/User AS User WHERE Env/MachineName ~ ".*8.*"`. To the right of the editor is a "Saved view" section showing a list of saved queries. A red arrow points to this section with the label "Saved queries/views". Below the editor and saved view section is a table with two columns: "Machine name" and "User". The table contains 19 rows of data. At the bottom of the table, there is a footer that reads "19 Manatees Page 1 of 0 (50 per page)".

Machine name	User
Machine 68	User 68
Machine 48	User 48
Machine 38	User 38
Machine 87	User 87
Machine 83	User 83
Machine 80	User 80
Machine 88	User 88
Machine 89	User 89
Machine 82	User 82
Machine 86	User 86
Machine 58	User 58
Machine 78	User 78
Machine 18	User 18

Figure 33: The advanced query interface

There is an editor to enter the query and then a list of saved views/queries. Once you enter a query, you can save it using the floppy disk iconed button below the editor. The other buttons work similar to what is available on the simple query interface.

An mql query can either take the form;

```
1 SELECT <projection> WHERE <filter>
```

or simply use <filter> to view the standard properties of the matched Manatees.

The <filter> consists of;

- a simple comparison; <property> <comparator> <value>, e.g. `A = 1`.
- a negation; `NOT (<filter>)`, e.g. `NOT (A = 1)`
- a conjunction; `(<filter> AND <filter>)` e.g. `(A = 1 AND B = 2)`
- a disjunction; `(<filter> OR <filter>)` e.g. `(A = 1 OR A = 2)`

::: tip Surround with “(” and “)”

You need to surround conjunctions (AND clauses) and disjunctions (OR clauses) with parenthesis *always*, i.e.:

```
1 (Something OR SomethingElse)
```

:::

Properties are the information available on each Manatee, e.g. `Env/MachineName`.

Comparators are;

- `=` to check for equality (has alias `EQUALS`)
- `>`, `<`, `>=`, `<=` to compare numeric values
- `STARTSWITH` to check for a prefix
- `ENDSWITH` to check for a suffix
- `~` to match a regular expression

Values are integers, strings (surround with `"`) or booleans (`true` and `false`).

The optional **projection** part of the query can be a , -separated list of properties or you can provide an alias for each item. E.g.

```
1 SELECT Env/MachineName AS Machine, Env/Username AS User WHERE ...
```

MQL examples

```
1 // show default properties for all v1.29 Manatees
2 Env/Version = "v1.29"
3
```

```
4 // show default properties for Manatees on machines whose name starts w
   "M" and is followed by 3 digits
5 Env/MachineName ~ "M\d{3}"
6
7 // all Manatees with both these settings matching
8 (Settings/KwanzaFlushStreamInterval = 0 AND Settings/ProductionGroup =
   "PRODUCTION")
9
10 // show version (with header "V") on all Manatees on machines whose
   name begins with "U" or where the username of the user ends with "n"
11 SELECT Env/Version AS V WHERE (Env/MachineName STARTSWITH "U" OR Env/
   Username ENDSWITH "n")
```

Detailed view

The detailed view shows information about a single running Manatee instance.

The screenshot displays the 'Details for Manatee c58oiui3fcq40i5169j0' page. At the top, a green dot indicates the instance is 'Running? Green means yes, black no.' Below this, a table lists basic information: Username (User 84), Machine name (Machine 84), Domain (sirenia), Manatee version (v1.29), Branch (demo/latest), and Uptime (...). To the right of this table are action buttons: 'Run Flow ...', 'Reset Settings', 'Restart', and 'Shutdown'. A red arrow points to these buttons with the text 'Actions that you can make this Manatee do'. Below the basic information is a 'Running flow' section showing 'Idling'. The main part of the page is the 'Settings' tab, which has a search bar and a list of settings. The settings are numbered 0 to 108, with values displayed in input fields. A red arrow points to the settings list with the text 'The settings for this Manatee. Can be edited.'.

Running? Green means yes, black no.

Basic information about this Manatee

Details for Manatee c58oiui3fcq40i5169j0

Username	User 84
Machine name	Machine 84
Domain	sirenia
Manatee version	v1.29
Branch	demo/latest
Uptime	...
Running flow	Idling

Run Flow ...

Reset Settings

Restart

Shutdown

Actions that you can make this Manatee do

Settings

Groups 0

Search

0	a104f820f7df54dc09fe:
1	ce7fad76bc42757ad95:
10	954dabc0f643e3ff784d
100	c1935f104d5d723fcd3:
101	5d47baf21d643c0a53b
102	0ab9c3d93ceb5ae09c3
103	2720beb08df0e4dc03c:
104	555285536cc003aa4d9
105	85752fb8bc958badd43
106	9e6789c2dc0a2d5b3e2
107	4e8ed336ac71f27ee59:
108	f201ce0e0dab6249c48:

The settings for this Manatee. Can be edited.

Figure 34: Information and actions for a single running Manatee instance

On this page you can see all settings and groups (AD and otherwise) and other basic details for the Manatee. You can also perform basic actions on the Manatee, e.g. make it run a flow, restart it or shut it down.

It is also possible to modify its settings.

Alerting

Using the Conductor component it is possible to set up alerts for Manatee instances. This is primarily intended for unattended RPA i.e. where you want to monitor a Manatee instance for certain metrics or ensure it remains running. Alerts can be accessed from the primary menu or from the targeted Manatee instance in the Conductor interface.

An alarm consists of:

- A **target** Manatee identified by a *user* and a *machine*.
- A **name** for the alert.
- An alert **condition** which determines when the alert should activate.
- A number of **actions** to execute when the alert is activated.

Conditions

The *condition* sets up the parameters for *when* an alert should activate. The condition is checked periodically so you need to specify;

- **interval** at which to check the condition, and
- **hits** which is how many positive indications are needed to activate the alert.

The actual action to perform to figure out whether to activate an alert can take two forms.

Query

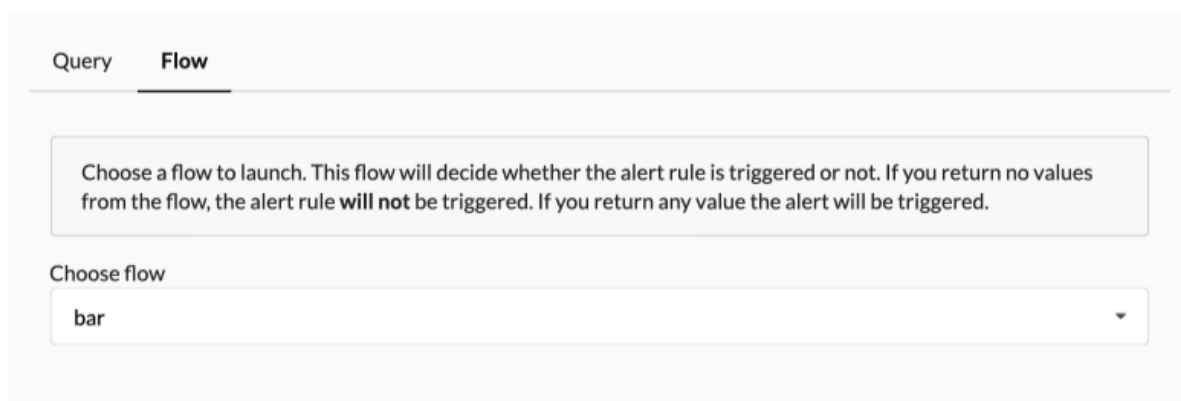
Using a query to determine if alert should activate. We currently only support *uptime* and *downtime* properties. Clicking on the query element will enable you to specify e.g. that the alert should activate if the downtime of the Manatee instance exceeds 10 minutes.

The screenshot shows the 'Query' tab of an alert configuration interface. At the top, there are two tabs: 'Query' and 'Flow'. Below them, a text label reads 'Trigger when the following query is true'. A search bar contains the text 'Alerting/Downtime is greater than 10', with a grey 'x' button to its right and a green '+' button to its left. A dropdown menu is open below the search bar, showing a list of properties and operators. The 'Alerting/Downtime' property is selected. The 'Select operator' section shows a list of operators: 'equals', 'does not equal', 'is less than', 'is less than or equals', 'is greater than' (which is highlighted), 'is greater than or equals', 'contains', 'does not contain', 'starts with', 'ends with', and 'matches regular expression'. Below the operators, there is a section 'Enter value to compare' with a text input field containing the number '10'. To the right of the dropdown menu, there is a green button labeled '+ Add alert action'. Below the button, there is a text label: 'The action will run when the alert switches from inactive to active.'

Figure 35: Setting a condition as a query

Flow

You can also designate a flow to be run as a condition. The flow will be run at the given intervals and the return value of the flow indicates whether to activate the alert or not. If no return value is given then all is well, but any non-empty return value will activate the alert and furthermore serve as the alert message reported.



The screenshot shows a configuration interface with two tabs: 'Query' and 'Flow'. The 'Flow' tab is active. Below the tabs is a text box containing the following text: 'Choose a flow to launch. This flow will decide whether the alert rule is triggered or not. If you return no values from the flow, the alert rule **will not** be triggered. If you return any value the alert will be triggered.' Below this text box is a label 'Choose flow' and a dropdown menu. The dropdown menu is open, showing the text 'bar' and a small downward arrow on the right side.

Figure 36: Setting a condition as flow to be run

Actions

As soon as the alert becomes active all the associated actions will be run. Currently we support:

- sending an *email* or
- sending a *chat* message

as actions to perform. Both of these can send a message to a recipient once the alert becomes active. An example of a configured action is here an email action:

Alert action 1 + Add alert action

The actions to take when a condition is met. The action will run when the alert switches from inactive to active.

Type ☒ Email ☐ Chat

Choose Mail...

Sirenia IMAP

To support@sirenia.eu Subject Alert!!!

Body

Alert said "{{message}}"

× Delete alert action

Figure 37: An email action

In the message field you can use the `message` string enclosed by `{{` and `}}` to include the return value from e.g. the flow that triggered the action as seen above.

Tracking alerts

Using the alert interface you can track the status of an alert. The list of alerts offers a minimal view of the state of the alert in the *last 5 hours*.



Figure 38: The status of the last 5 hours for an alert

Each of the thin vertical bars represent an hour (from now) while the thicker bar is the current state of the alert. Red for an active alert, green for inactive.

Select the alert will take you to the configuration page for the alert where you will also see a chart that displays the state of the alert for *the last 5 days*.

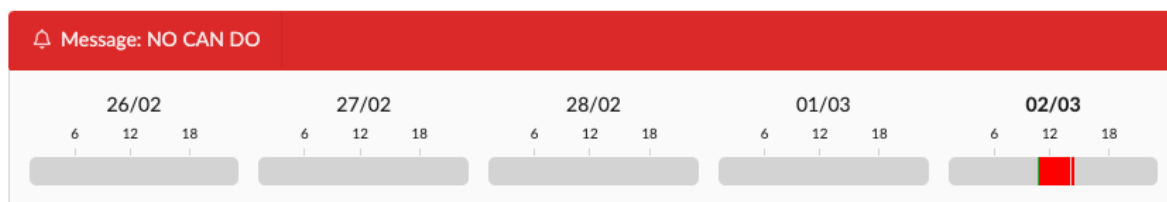


Figure 39: The status of the alert for the last 5 days

Here you can also see the alert message if the alert is currently active.

Restrictions

Restrictions make it possible to centrally configure rules for and actions to take when selected API methods are invoked. As the name implies one of the usages of a restriction is to prohibit the use of an API but it that is just one of the *actions* to take. A restriction targets a selection of Manatee instances by a specified *primary group* and *version*.

Group and version to restrict

Group	Version
PRODUCTION	1.29

Figure 40: Restriction targets a primary group and version

Once you have selected the version you can begin to add restrictions. Each restriction consists of a module and a method to target and a list of actions to take when this method is invoked. Restriction actions can be reordered and will be applied in the order given. The available actions are:

Deny

Simply deny access to the API. This will stop the flow once the API is invoked.

Restrictions

Module

Wait

Method

forSeconds

=

Type of restriction

Deny (stops the flow)

Deny (abort)

✕ Delete Action

▼ Hide restrictions

💡 Affected flows

✕ Delete restriction

+ Add Action

+ Add restriction

Figure 41: Deny access to Wait.forSeconds

Prompt the user

Manatee will prompt the user who should allow/deny the invocation. If the user denies the invocation the flow is stopped.

Restrictions

Module

Wait

Method

forSeconds

=

Type of restriction

Prompt the user

The message that the user will be prompted with

Are you sure that we should wait?

Timeout for the message prompt

100

×

Delete Action

▼

Hide restrictions

💡

Affected flows

×

Delete restriction

+

Add Action

+

Add restriction

Prompt user

Figure 42: Prompt the user on all Wait.forSeconds invocations

Log the invocation

Log an entry in the remote/local log with a given message. This can be e.g. used for statistical analysis in our Analytics product.

Restrictions

Module

Wait

Method

forSeconds

=

Type of restriction

Log the usage and continue

The message to log

Wait.forSeconds invoked

Log and continue

Delete Action

Hide restrictions

Affected flows

Delete restriction

Add Action

Add restriction

Figure 43: Log all invocations of Wait.forSeconds

Check parameters

If you need more fine-grained control over when to perform a restriction you can use the check parameters restriction action.

Restrictions

Module

Wait

Method

forSeconds

=

Param check

Type of restriction

Check parameters

Which inputs should be checked?

Min secs

10

Max secs

20

↓

Action to take if inputs match.

Type of restriction

Deny (stops the flow)

Log the usage and continue

Prompt the user

Hide restrictions

Affected flows

Delete restriction

Add Action

+ Add restriction

Figure 44: Check parameters on Wait.forSeconds

If the invocation is invoked with matching arguments then the configured actions will be run. You can add a *deny*, *log* and *prompt* action here.

Affected flows

If you click the *Affected rows* button you can get an overview of the impact of the restriction i.e. in which flows it is invoked. Note that we only match on module and method not on argument so all results shown are not necessarily affected if your restriction checks parameters.

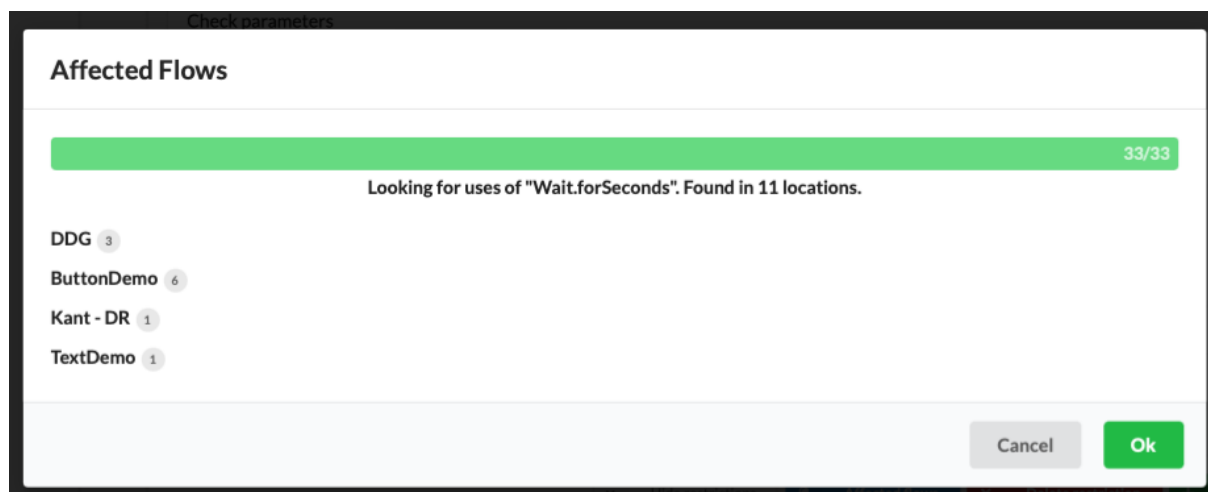


Figure 45: Result of clicking affected flows

You can expand the results to show more information about each affected application and flow.

Settings

You can override most of the default Manatee settings on a *primary group* and *version* basis centrally. This is done by selecting the target parameters (group and version) and then adding settings to override.

Select primary group and version to target

Group	Version
PRODUCTION	1.29

Add settings to override

	Setting value	
Max tasks for window title resolution		
MaxConcurrentTasksForWindowResolvers	10	✕
The list of colors given to sessions		
SessionColors	#001100, #D0021B, #369E2C	✕
Show notification when attaching to an application repeatedly fails		
ShowNotificationOnAttachFailure	<input type="checkbox"/>	✕

+ Add setting

Figure 46: Override default Manatee settings example

In the example above we have overridden three default settings. These will take effect either on restart or immediately depending on whether Manatee is able to dynamically apply the given setting change.

Localization

Both Cuesta and Manatee store their localized strings and other resources in Kwanza and you can use Cuesta to edit these in a simple table view.

Localization

Search

+ 887 rows - + 3 columns - < > ↑ ↓ << < 1 / 36 > >> 25

<input type="checkbox"/>	key	da	en
<input type="checkbox"/>	2FA_ENABLED	2-faktor login	2-factor authentication
<input type="checkbox"/>	2FA_ENABLE_AND_VERIFY_CODE	Aktiver og log ind	Enable and login
<input type="checkbox"/>	2FA_ENTER_RECOVERY_CODE	Skift mellem engangs- og backup-kode	Click this button to switch between c
<input type="checkbox"/>	2FA_PREFERENCE_DESCRIPTION	Aktiver 2-faktor login. Du skal bruge	Enable 2-factor authentication. You a
<input type="checkbox"/>	2FA_RECOVERY_CODES	Kopier og gem disse backup koder hvis	Copy and save these recovery-codes ir
<input type="checkbox"/>	2FA_RECOVERY_CODES_PROCEED	Fortsæt	Proceed
<input type="checkbox"/>	2FA_VERIFY_CODE	Valider kode	Verify code
<input type="checkbox"/>	ABOVE_TITLE	Målet kan findes under referencen.	Target can be found below the referer
<input type="checkbox"/>	ABSTRACT	Kort beskrivelse	Short description
<input type="checkbox"/>	ACCESS_DENIED	Du har ikke adgang til {location}. Kc	You cannot access {location}. Contact
<input type="checkbox"/>	ACTION	Handling Handler	Action Actions
<input type="checkbox"/>	ACTION_CLICK	Klik	Click
<input type="checkbox"/>	ACTION_FOCUS	Fokuser	Focus
<input type="checkbox"/>	ACTION_INPUT	Input	Input
<input type="checkbox"/>	ACTION_READ	Læs	Read
<input type="checkbox"/>	ADD	Tilføj {thing}	Add {thing}
<input type="checkbox"/>	ADD_ITEM_CONFIRM	Er du sikker på, at du vil tilføje el	Are you sure you want to add item {t
<input type="checkbox"/>	AFFECTED	Påvirkede {thing}	
<input type="checkbox"/>	ALERT	alarm alarmer	alert alerts

Cuesta Manatee (strings) Manatee (resources)

Update

Figure 47: Editing the localized strings for Cuesta

Each column represents the strings (or resources) for a given language and each row is then the translations for a given key. So if you want to change the text for the Danish translation of [ACTION_CLICK](#)

you need to find the row and then edit the cell in the *da* column. Cells with a yellow background indicates that a translation is missing for that language.

Resources are base encoded binary blobs. Most resources are images, icons etc used in Manatee/Cuesta.

You can also download the localized strings/resources as a csv file, edit by hand or import into e.g. Excel and edit, then re-upload the edited version.

Hub

The Hub is a place to share flows. It works similarly to an app store for flows where developers publish apps and users download and use them. In the Hub you publish and download flows.

Hubs can also exchange flows internally by configuring them to pull flows from another hub. In this manner we can setup a global hub containing flows that everyone can access, then we can e.g. setup a nordic hub which gets all the global flows but also contain some nordic-only flows. Going further we can setup e.g. a Danish hub which pulls flows from the nordic hub (and thus also gets the global flows) but which also contains some flows exclusive to the Danish hub.

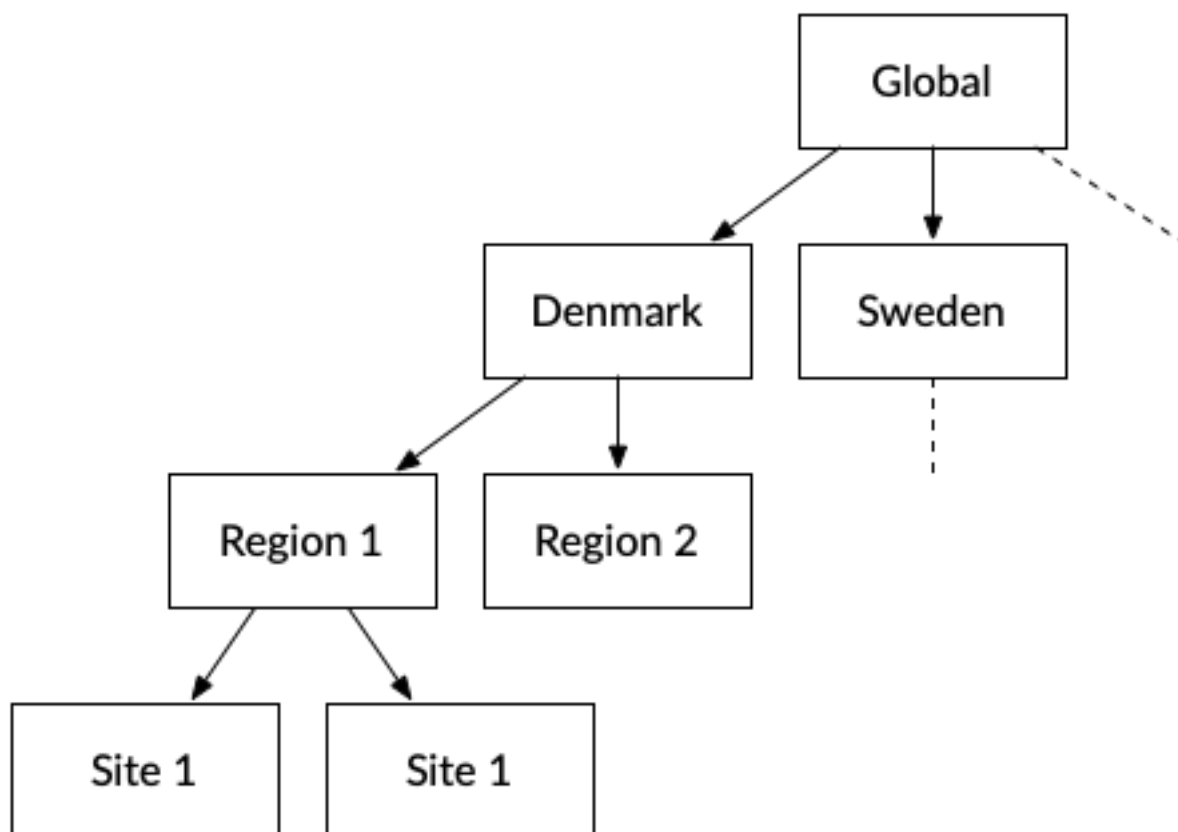


Figure 48: Published items can flow from the global store towards more specific store instances

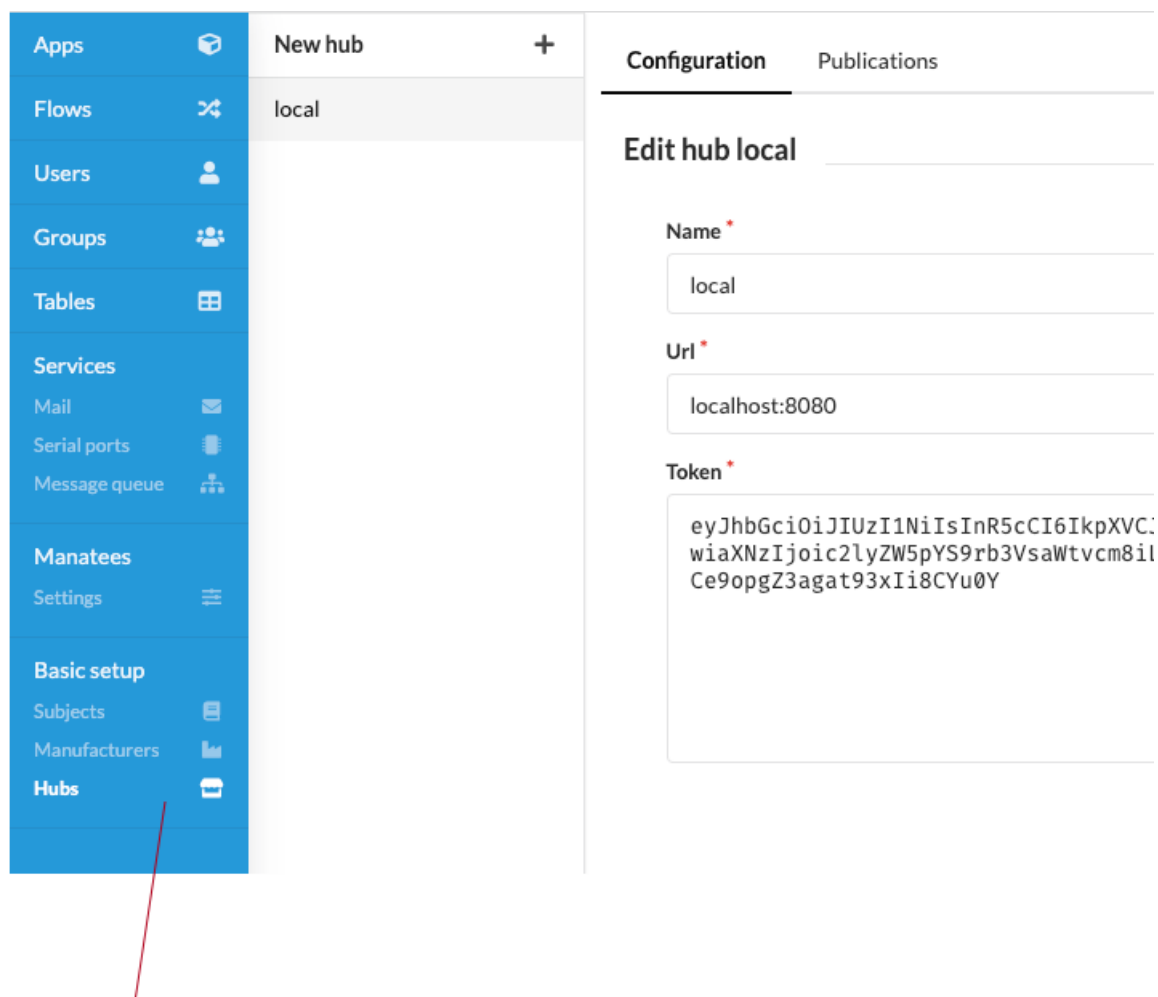
A public facing Hub is available at <https://demo.hub.sirenia.io/> from which flows published from our demo Cuesta instance can be published.

In order to interface with a Hub instance from Cuesta you need to configure it with the location etc of the hub.

Configuring hubs

Each hub is defined with a name, url and an auth token. Once Cuesta knows about at least one hub you can start to both publish flows to and get published flows from the hub. There are multiple workflows involved as described below.

You can add, edit and delete hubs by selecting the hub icon in the persistent menu on the left-hand side of the UI.



Hubs can be configured in the menu entry

Figure 49: Configure a hub in Cuesta

Each hub needs a name, a url designating the location of the hub and a token to authenticate itself. You can find/generate the token by logging into a hub with administrative access.

Publications

To see which flows from the registry has been published to a particular hub, you select the “Publications” tab.

Configuration **Publications**

All publications from this registry to local.

Name	Publication type	Short description	Version
Test flow	Single flow		1.7
1 row Page 1 of 1 (50 per page)			

Figure 50: Hub publications

Publish flows

You can publish any flow once or multiple times to the same hub. Use “hub” tab in the bottom flow menu.




Figure 51: Hub tab

This will bring you to a page on which you can publish the current flow.

Publish a new flow

After selecting the “Hub tab” you will be presented with options for publishing a new flow.

Create publication

**Inlined flow**

An inlined publication of a flow involves pre-processing the flow s.t. any fields and other flows referenced are included in the publication.

Users cannot see the code of an inlined publication making applying updates to the flow easy and painless. Use this publication if you need to publish something that can be tailored via extensions when downloaded.

Publication type
Inlined flow

Hub
local

Create publication

Figure 52: Publish a flow

Here you can choose between publishing a *single flow*, or an *inlined flow*. Then you choose the hub to publish to and click create-new-publication and proceed to fill out the details for the publication.

Publication

Publish an initial or an updated version of this flow to a hub.

Name *
Test flow

Category

App

Short description

Description ?

Figure 53: Fill out info about the publication

When all looks good you need to press the “save” button in the bottom of the form.

Attachments

No attachments. Drag files here to attach.

Releasenote

Version ^{*}

1.0

Releasenote [?]

Some notes on the released flow


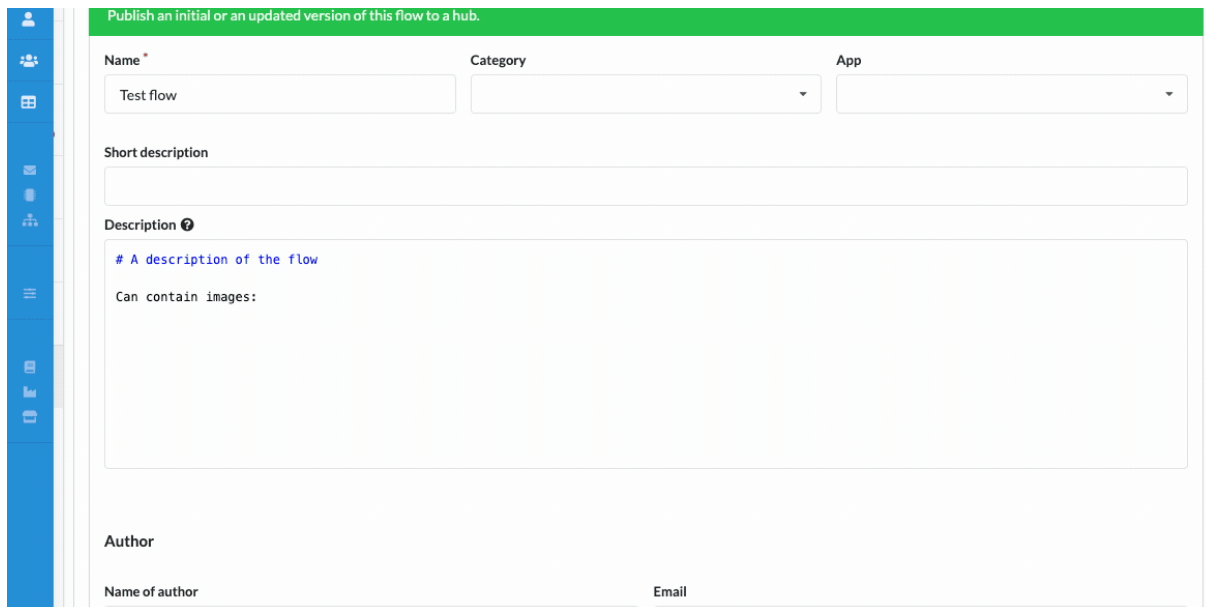
 Cancel Save

Figure 54: Save a new publication

You can use markdown and attach files to the publication by dragging them either to the description editor or the list of attachments.



The screenshot shows a web interface for publishing a flow. On the left is a blue sidebar with icons for user, team, flow, test, share, list, and document. The main content area has a green header bar that says "Publish an initial or an updated version of this flow to a hub." Below this is a form with the following fields:

- Name ***: A text input field containing "Test flow".
- Category**: A dropdown menu.
- App**: A dropdown menu.
- Short description**: A text input field.
- Description ?**: A large text area containing the text "# A description of the flow" and "Can contain images:". Above this area is a small icon of a document with a plus sign.
- Author**: A section with two input fields: "Name of author" and "Email".


Figure 55: Dragging files from the desktop onto the description to add them as attachments

Once you click publish, the flow will get packaged and sent to the hub for others to see and download.

Publish an update to an already published flow

If you've updated the source flow and you want to publish a new version, then you again go to the hub tab and an option to publish an update to the flow.

Publication

PUBLICATION TYPE Single flow	NAME Test flow	LICENSE MIT	CATEGORY	APP
RELEASENOTE This is the 2nd release				
AUTHOR Jonathan Bunde-Pedersen	VERSION 1.7 1 changes since last publication, 2 hours ago		HUB local	
		Edit publication Test flow Publish new version		

Edit the details of **this version** of the flow. No new version will be published, the existing one will be modified.

Flow has changed and you can publish a new version of it to the hub.

Figure 56: Publish an update to an existing publication

You must change the release version to publish an update and it is very, very much recommended to also add a new release note describing the changes from the latest published version. You can click the “N changes since ...” link to see which changes you’ve made to the code since the flow was last published.

Clicking “save” will then publish a new version for others (or yourself to get). The change to be published will be tagged with the latest name and version published of this flow.

Errors when publishing

We validate that the published flow is given a name as well as a version.




	You need to provide a name.
	The release-note version must be changed to publish a new version.
	Cancel Save

Figure 57: Validation error on name/version

When publishing a new version you also need to update the version (and optionally, but please do,

the release note).

If you're publishing an inlined flow then we also do a number of validations on the flow code itself. These will also be shown in the publication view:

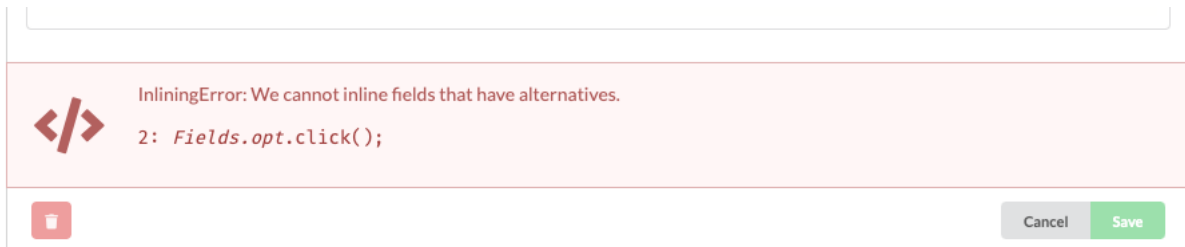


Figure 58: Validation error on inlining

Here, we are using a field `opt` (in line 2 of the flow) that is defined with alternative matches and that cannot currently be handled by the inlining engine.

Get flows from a hub

You can get flows from a hub to via any connected Cuesta. This means that you can get a flow to the same Cuesta as the one that originally published it (as long as you change the name of the publication to not match the original flow). The idea behind this is that you can use the hub as a staging platform for production flows, i.e. publish flows when they are ready for prod while keeping a develop/test version that you can update independently of the production version.

There are multiple ways to get flows from hub.

Via links in the hub

The first will work if you've let Cuesta register itself as a handler for hub-links. In Chrome you do this by clicking the icon shown below and allowing the registration to take place.

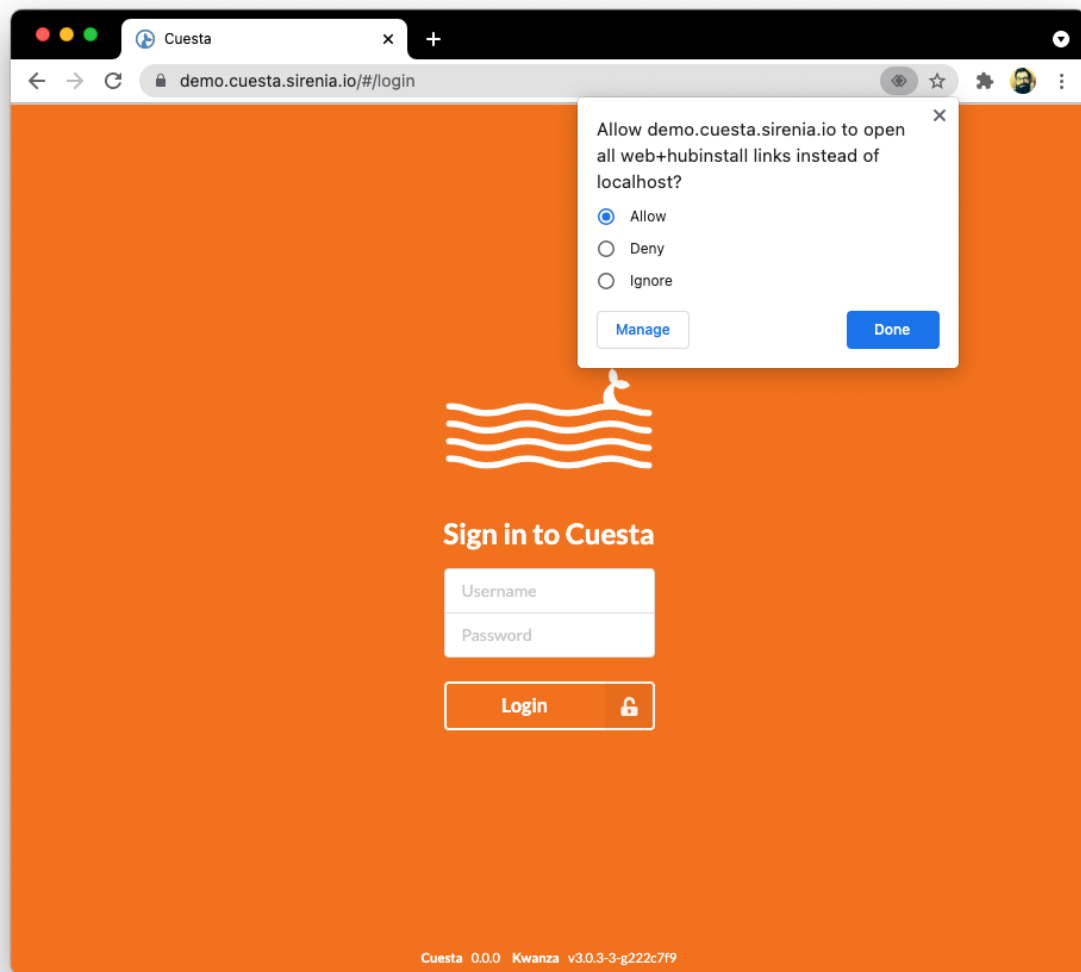


Figure 59: Allow Chrome to handle web+hubinstall links

Once this is done you can use the GET links from the hub to open an install page in Cuesta from where you can get the flow.

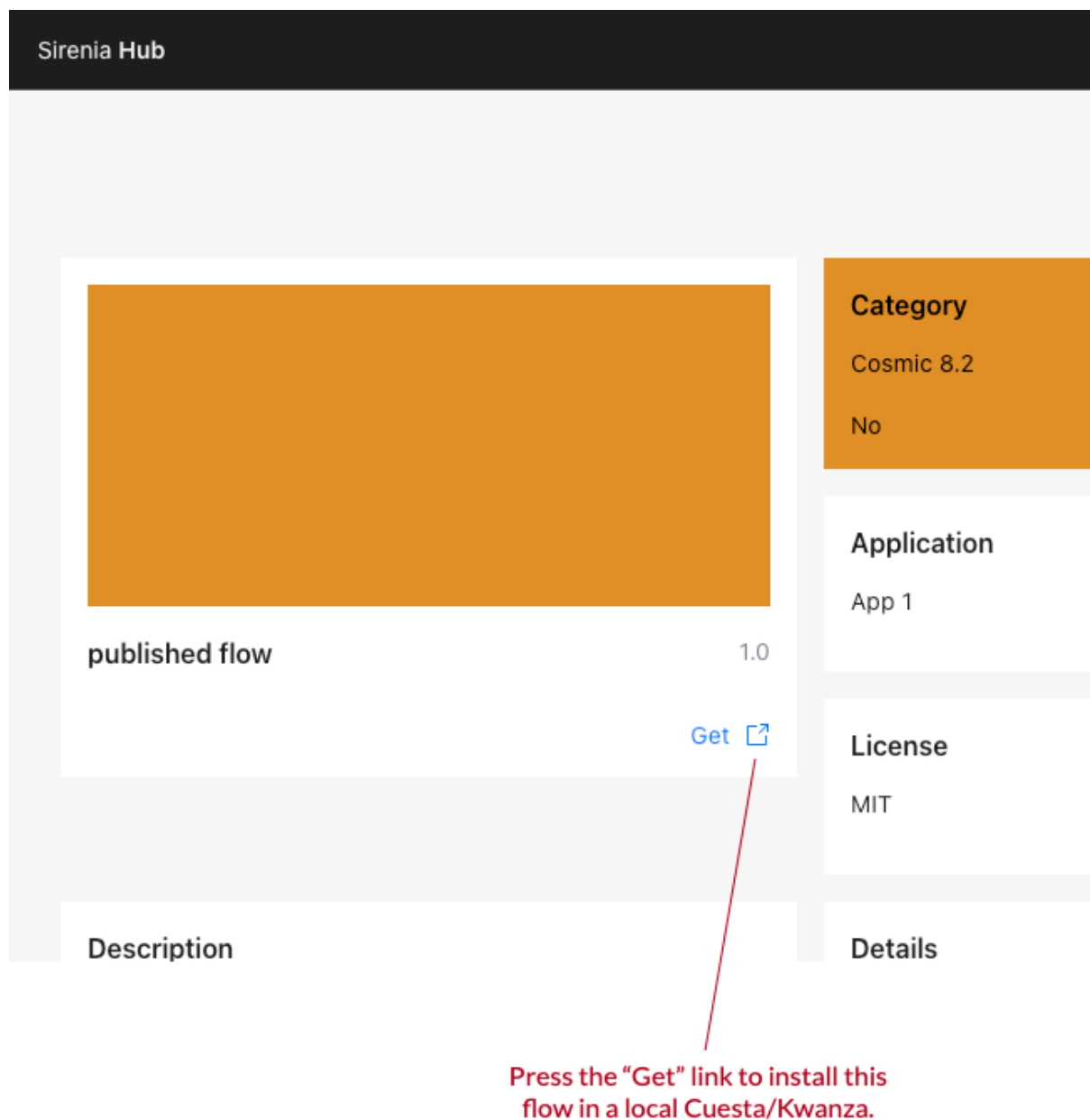


Figure 60: Install publication with link from hub

You will then be taken to a location in Cuesta where you can select the app in which to place the flow.

Choose which app to install the flow "Test flow" in

Notepad

PUBLICATION TYPE	NAME	LICENSE
Single flow	Test flow	MIT

Get flow

Figure 61: Install publication with link in Cuesta

Via searching for the flow in Cuesta

If you want to get a published flow directly in Cuesta, then you can select the GET flow in the flow menu.

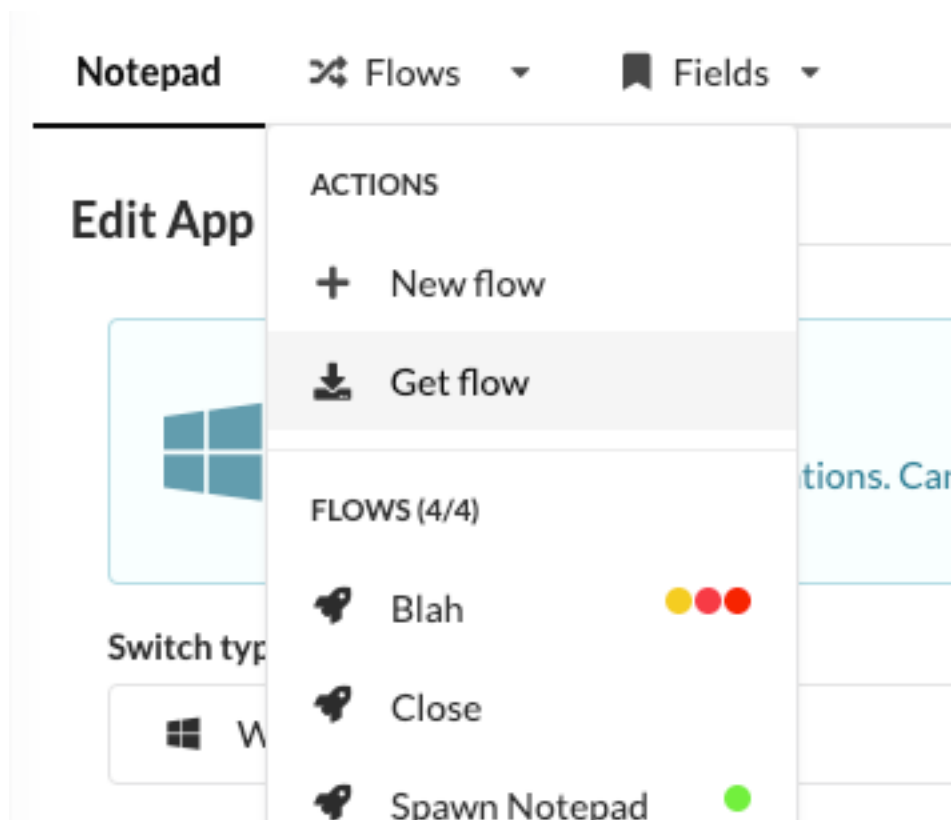



Figure 62: Choose to get a flow from a hub


This will take you to a page where you can search for a flow to get.

Update existing or get new flows from a hub

Find the flow to get or update by using the controls below. You can see all flows from all configured hubs here.

Search Category App

PUBLICATION TYPE Single flow	NAME published flow	LICENSE MIT	CATEGORY Cosmic 8.2	APP App 1
				 Get flow

PUBLICATION TYPE Single flow	NAME to pub rt64456	LICENSE MIT	CATEGORY ffsds	APP App 1
				 Get flow


PUBLICATION TYPE Single flow	NAME to pub 2345gh	LICENSE MIT	CATEGORY Green Category	APP App 1
				 Get flow

Figure 63: Search for a flow to get from within Cuesta

This list will also display already downloaded flows s.t. you can update to new versions for existing flows.


PUBLICATION TYPE Single flow	NAME TEST	LICENSE MIT	CATEGORY ffsds	APP App 1
 The downloaded flow is the most recent version. Go to "TEST" .				

Figure 64: Update an existing flow from the search list

Once you click “get flow” then the flow is downloaded to the local registry and you can start to make it available to users by assigning it groups etc

Updating a published flow to a new version

If you already have a version of published flow and a new version has been published, then you can update to this version by finding the published flow in the list of hub published flows (same as when you initially get the flow) or by going to the settings page for the flow. You’ll see information about the new version and controls to update in the top of the page.




Flow is downloaded from a hub in version 1.0				
PUBLICATION TYPE Single flow	NAME Test flow	LICENSE MIT	CATEGORY	APP
RELEASENOTE This is the 2nd release				
 A new version of has been published. You can update to the latest version here.				
 The flow has local changes. You can update, but it will overwrite any local changes you have made.				
				 Update flow

Figure 65: Update a flow from its settings page

A warning will also be shown (as you can see here) if you've modified your flow locally. By updating the flow, these modifications will get merged into the resulting flow or be overwritten by the new version depending on what you've changed. You can always revert to a previous version of a flow in the changes tab.

Whenever you get a new version of a flow, we'll automatically tag the change to make it easier to rollback to a previously downloaded version.

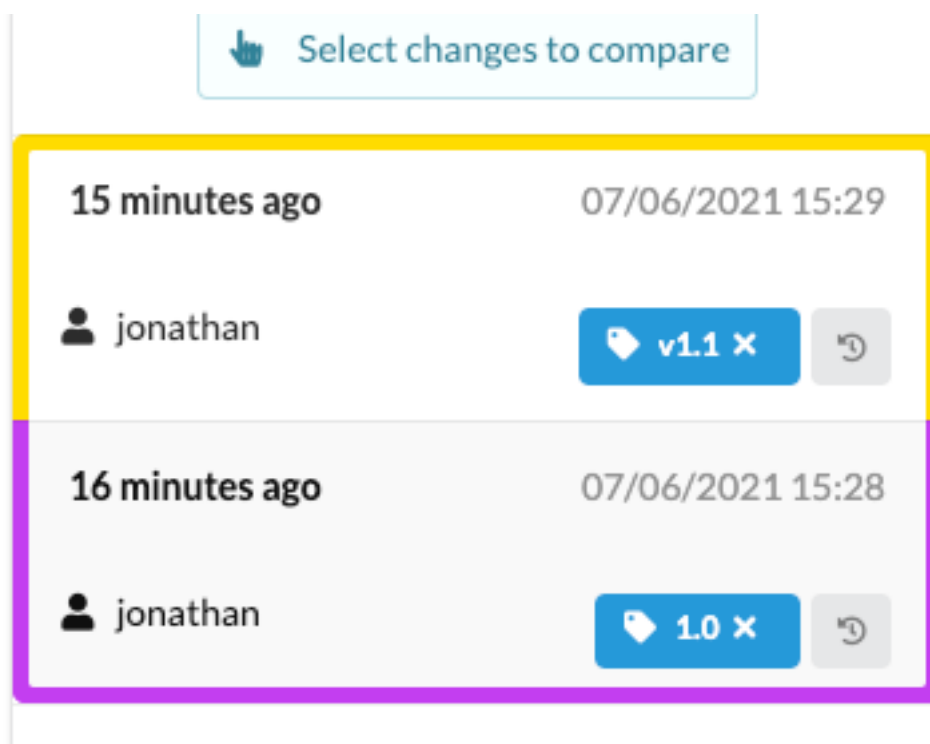


Figure 66: Flows gets tagged automatically when getting them from a hub or when publishing them