Flows

v1.26

Sirenia

September 20, 2019

Contents

This page documents the API for interacting with fields and the global objects in flows.

In general flows are JavaScript code and thus any valid JavaScript is allowed. See https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript for a JavaScript intro. The objects `Inputs`, `Fields` and all the rest of the modules listed below are made available for all flows s.t. they can be accessed in the JavaScript code.

We include the utility library `lodash 4.7.10` in all flows. See https://lodash.com/docs the available functions.

There is also a tutorial to the Cuesta tool which also contains a few examples of flows. It is highly recommend that you go through the tutorial before diving into this reference documentation.

1 Inputs

Inputs to a flow can be accessed via the `Inputs` array. Inputs are generally strings.

Example

```
var mi = Inputs["myinput"];
```

---

2 Fields

Fields represent user-interface elements which can be manipulated from a flow. The basics of defining a fields and how to use it in flows is described in detail in the field documentation.

---

3 Global objects

The *global objects* listed below are available in all flows.

3.1 Dialog

The dialog object contains methods for presenting the user with information or requesting information from the user at runtime.

3.1.1 Info dialog

Shows a blue information dialog with an OK button. The flow does not proceed until the user has clicked OK. Options is an optional parameter.

Parameters

- `header` is the title of the dialog
- `text` is the text content shown
- `options` is a JavaScript object, supported properties:
  - `buttons` is an array of buttons to display in the bottom part of the dialog
  - `timeout` an int determining when the dialog should auto-close
  - `sound` a string (one of `asterisk`, `beep`, `exclamation`, `hand`, `question`) which indicates a system sound to play once the dialog is shown

The `buttons` array consists of `button` objects with the following properties:

- `value` the text to display on the button (should be unique for a dialog)
- `isDefault` (boolean) a true/false value indicating whether or not this button is the default (i.e. will be activated on the enter-key) - should only be set to `true` for one button per dialog -- default is `false`
- `isCancel` (boolean) indicating whether or not the button should cancel the dialog -- default is `false`

The default value for buttons is an ”OK” button:

```
[
  { 'value': 'OK' }
]
```

The button clicked will be available as a property named `button` on the return value from the dialog. If the user clicks a cancel button then an exception is thrown.

Example

```
Dialog.info(”Hello”, ”This is some text to be shown.”, {});
```

With options:

```
Dialog.info(
  ”Hello”,
  ”Some text - I will max be shown for 10 secs.”,
  { timeout: 10 }
);
```

With pre-defined buttons:

```
var r = Dialog.info(
  ”Hello”,
  ”Do you want to continue”,
  { timeout: 10
  , buttons: [
      { 'value': 'No', 'isCancel': true },
      { 'value': 'Maybe' },
      { 'value': 'Yes' },
    ]
  }
);
if (r.button == 'Yes') {
  // user answered yes - we can continue
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.1.2 Warn dialog

Shows a red warning dialog to the user with an OK button. Similar to the info dialog, but red. Options is an optional parameter.

Parameters

- `header` is the title of the dialog
- `text` is the text content shown
- `options` is a JavaScript object, supported properties:
  - `buttons` is an array of buttons to display in the bottom part of the dialog (see info-dialog for further information)
  - `timeout` an int determining when the dialog should auto-close
  - `sound` a string (one of `asterisk`, `beep`, `exclamation`, `hand`, `question`) which indicates a system sound to play once the dialog is shown

Example

```
Dialog.warn("Warning!!", "This is some text to be shown. Consider yourself warne
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.1.3 Input dialog

Shows a dialog into which the user may input data. The type of data which can be input is determined by the `options` parameter.

Parameters

- `header` is the title of the dialog
- `text` is the text content shown
- `options` is a JavaScript object which determines the input the user should provide. Each property on the object is one input the user must provide. The name of each property is used when returning the results. It can also contain the following properties which affect the dialog itself:
  - `buttons` is an array of buttons to display in the bottom part of the dialog (see info-dialog for further information)
  - `submitOnValidation` is a boolean flag that determines whether or not the dialog will be automatically submitted when all fields validate - or not

- `maxDialogWidth`/`maxDialogHeight` (int) change the default maximum width and height for the window,
- `promptWidth` sets the with of the label/prompt
- `sound` a string (one of `asterisk`, `beep`, `exclamation`, `hand`, `question`) which indicates a system sound to play once the dialog is shown
- `dialogPositionTop`/`dialogPositionLeft` (int) to change the default position of the dialog. Note that if one of these properties are set then the dialog will be positioned on the main display.

Inputs given as complex objects

If the value of a property of `options` is either a complex object or a function it is treated as an input element. If you supply an object then the following properties are available to specify:

Each input should contain the following variables:

- `type` to determine which UI element to display, `TEXT`, `PASSWORD`, `FILE`, `SELECT`, `RADIO`, `DATE`, `MULTITEXT`, `TYPEAHEAD`, `HEADER`, `DIVIDER`, `SPACER` and `DESCRIPTION` are the supported options - see options for each type below
- `dependsOn` is an expression that determines **when** this input should be shown. You can either specify the name of another property - in which case the input will be shown if the other property has a value, or you can specify a `<name-of-other-property>=<value>` type string - in which case the input will be shown if the other property has the given value. If `dependsOn` is empty the input will always be shown. Using a ~ instead of = in the expression will cause the value to be interpreted as a regular expression (from 1.8.0).
- `promptWidth` sets the with of the label/prompt

An example of an input dialog with a few objects is:

```
Dialog.input('header goes here', 'text goes here', {
  myTextInput: {
    prompt: 'Input text here'
    type: 'TEXT',
    value: 'Default value'
  },
  anotherInput: {
    prompt: 'Another prompt',
    type: 'PASSWORD'
  }
);
```

This will display a dialog with two inputs, one for text and one for password.

Inputs given as functions

If the value of an option is a function then that function is invoked with the current state of the form allowing you to build complex interacting elements. The following example demonstrates this by having the properties of the RADIO input determined by the previous inputs.

```
Dialog.input('header goes here', 'text goes here', {
  radioOptions: {
    prompt: 'Options separated by ","',
    type: 'TEXT',
    value: 'a,b,c'
  },
  radioPrompt: {
    prompt: 'The prompt for the radio',
    type: 'TEXT',
    value: 'Radio'
  },
  radioPromptWidth: {
    prompt: 'The prompt width for the radio',
    type: 'TEXT',
    value: '150'
  },
  radioOrientation: {
    prompt: 'Orientation',
    type: 'RADIO',
    selectBetween: ['vertical', 'horizontal']
  },
  radioTableLayout: {
    prompt: 'Table layout',
    type: 'MULTITEXT',
    texts: [
      { name: 'columns', prefix: 'Columns', value: "0" },
      { name: 'rows', prefix: 'Rows', value: "0" }
    ]
  },
  radioVisible: {
    prompt: 'Show RADIO',
```

```
    type: 'RADIO',
    selectBetween: ['No', 'Yes'],
    value: 'No'
  },
  d: { type: 'DIVIDER' },
  radio: function (s) {
    return {
      prompt: s.radioPrompt,
      selectBetween: s.radioOptions && s.radioOptions.split(','),
      orientation: s.radioOrientation,
      promptWidth: parseInt(s.radioPromptWidth || "150"),
      columns: parseInt(s.radioTableLayout && s.radioTableLayout.columns || "0")
      rows: parseInt(s.radioTableLayout && s.radioTableLayout.rows || "0"),
      dependsOn: s.radioVisible == 'Yes',
      type: 'RADIO'
    };
  }
});
```

When you run this flow you can use the inputs above the divider to control the appearance of the RADIO. The dependsOn property can be set to a boolean value to do complex dependency validations.

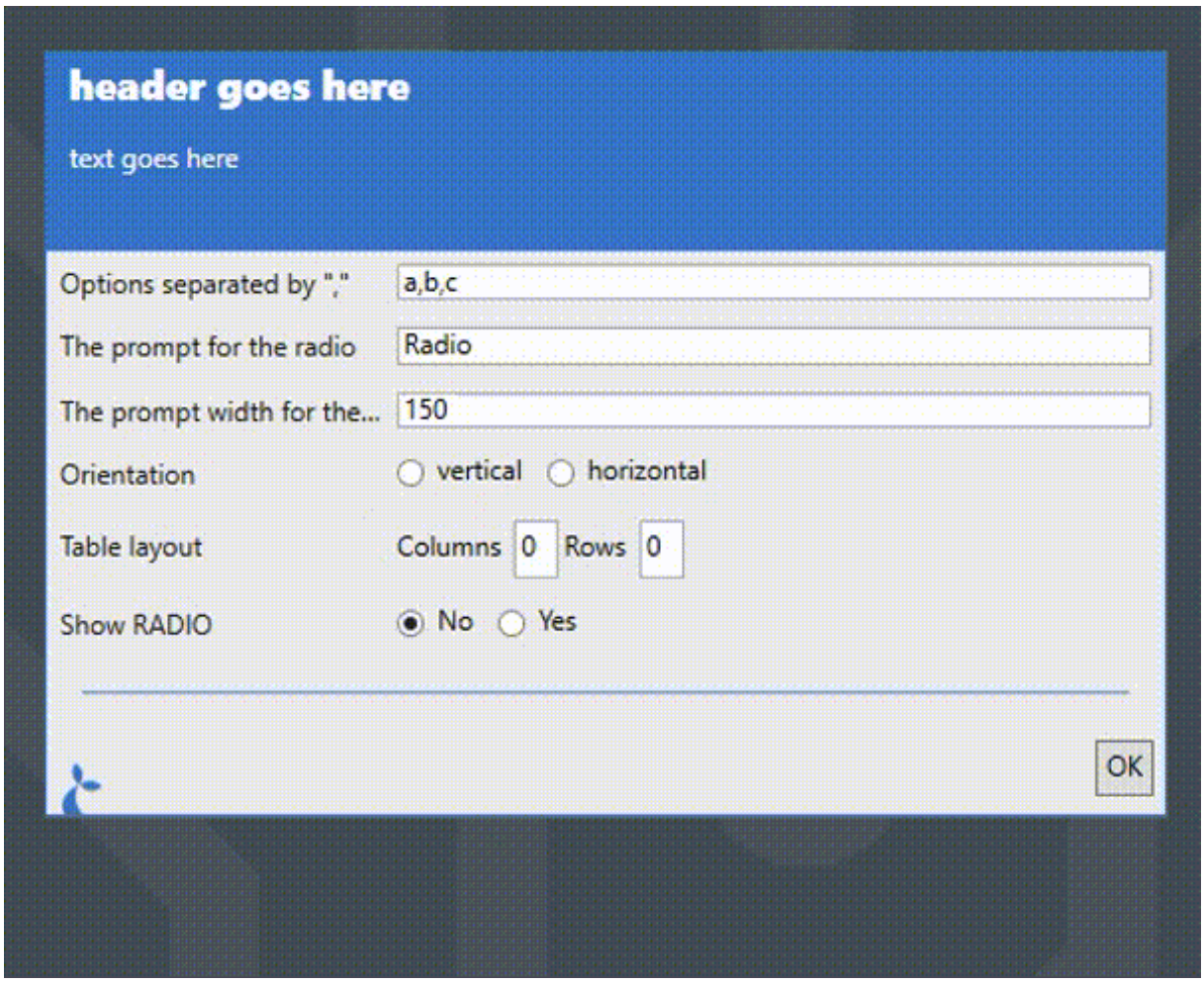

Figure 1: Example of a dynamic radio input

The properties of each `option` item depends on the value of its `type`:

TEXT and PASSWORD

- `prompt` is the text which is displayed as an hint to the user for this option.
- `promptAlignment` is the alignment the prompt should follow. Available options are: "Center", "Justify", "Left" (default), "Right".
- `value` is an *optional* default value for the input.
- `prefix` and `suffix` are texts to be shown before and after the input field.
- `focus` is whether to focus this field - if multiple fields have focus set to true then the last one will be focused.
- `multiline` whether multiple lines are allowed (default `false`).

- `validation` is a validation object (see below).

FILE and DATE

- `prompt` is the text which is displayed as an hint to the user for this option.
- `value` is an *optional* default value for the input - for DATE this may be a javascript Date object
- `focus` is whether to focus this field.
- `validation` is a validation object (see below).

SELECT and RADIO

- `prompt` is the text which is displayed as an hint to the user for this option,
- `value` is an *optional* default value for the input,
- `selected` is whether or not the option starts out as selected (checked) or not - only applicable for `SELECT`
- `selectBetween` is a array of strings which determines the available dropdown options if the `type` has value `SELECT`,
- `orientation` can be either 'vertical' or 'horizontal' and determines the layout direction,
- `columns` is the number of columns to display input elements in to help with alignment - setting `columns` will void the `orientation` setting,
- `rows` is the number of rows to display input elements in to help with alignment,
- `focus` is whether to focus this field
- `validation` is a validation object (see below).

CHECKBOX

- `prompt` is the text which is displayed as an hint to the user for this option,
- `value` is an *optional* default value for the input,
- `selected` is whether or not the option starts out as selected (checked) or not - only applicable for `SELECT`
- `options` is a array of objects which determines the checkboxes,
- `orientation` can be either 'vertical' or 'horizontal' and determines the layout direction,
- `columns` is the number of columns to display input elements in to help with alignment - setting `columns` will void the `orientation` setting,
- `rows` is the number of rows to display input elements in to help with alignment,
- `focus` is whether to focus this field
- `validation` is a validation object (see below).

Each object in the `options` array can have the following properties:

- `name` the name of the item,
- `value` the value
- `selected` whether the checkbox is selected.

A simple example of a CHECKBOX input could be:

```
Dialog.input(..., {
  cb: {
    prompt: "Checkbox example",
    type: "CHECKBOX",
    options: [{name: "cb1", selected: true}, {name: "cb2"}]
  }
});
```

HEADER and DESCRIPTION

- `value` is used as the text displayed.

MULTITEXT

- `texts` is an array of text inputs to show - each input may have the following properties set;
  - `name` is used to refer to the input,
  - `prefix` and `suffix` are texts to be shown before and after the input field,
  - `value` is the default value,
  - `multiline` whether multiple lines are allowed (default `false`)
  - `focus` is whether to focus this field
  - `preselect` an object which will be pre-selected
  - `validation` is a validation object (see below).

TYPEAHEAD

- `selectFrom` is the construction which determines what the user is able to select from.

The value of `selectFrom` can be a list of strings in which case the list is simply displayed. E.g.:

```
...
myProp: {
  type: 'TYPEAHEAD',
  selectFrom: ['Option 1', 'Option 2']
}
...
```

It can be a list of `objects` with a `value` or `display` property that is displayed for the user. As in the example below where the user can select or get auto-completion on 'a' and 'b'.

```
...
myProp: {
  type: 'TYPEAHEAD',
  selectFrom: [
    {display: 'a', id: 100},
    {display: 'b', id: 100}
  ],
  preselect: { display: 'a', id: 100}
}
...
```

The value of the `myProp` property after the input dialog is completed will be the full object selected, e.g. `{display: 'a', id: 100}`.

You can also supply arbitrary `objects` and a formatting string.

```
...
myProp: {
  type: 'TYPEAHEAD',
  selectFrom: {
    format: '{{foo}} with id {{id}}',
    items: [
      {foo: 'a', id: 100},
      {foo: 'b', id: 100}
    ],
  }
}
...
```

This will display e.g. "a with id 100" in the suggestion dropdown. The object selected will be available in the `myProp` property (not just the formatted string). In addition to the `format` string, you can also set the following options:

- `minInputLength` the minimum number of characters the user must input in order to get suggestions
- `filterMode` which mode should be used to filter the suggestions; select from `'contains'`, `'startswith'`, `'endswith'`.

A callback function can also be used. The function supplied will get invoked with the string entered by the user. E.g.:

```
...
myProp: {
  type: 'TYPEAHEAD',
  selectFrom: {
    format: '{{foo}} with id {{id}}',
    items: function(searchString) {
      return [
        {foo: 'a', id: 100},
        {foo: 'b', id: 100}
      ];
    },
  }
}
...
```

In this case we're not using the input for anything but other cases might do so, like when fetching options from e.g. a remote resource (via http or similar).

Lastly, the contents of a Table can be used as options.

```
...
myProp: {
  type: 'TYPEAHEAD',
  selectFrom: Table.map('nameOfTable', 'propToIndexBy').selectFrom('{{foo}} with
}
...
```

This will use the table rows and generate a formatted string for each row - the result will again be an object representing the row.

TABLE

The TABLE input can be used for tabular (ie like a spreadsheet) input. It supports the following properties.

- `tableHeader` is a list of strings or a list of objects with a `name` and a `type` and defines the columns of a table
- `tableRows` is the initial list of rows - the user may add more rows to the table

An example is given here:

```
var result = Dialog.input('Table Example', 'Show a table in an input dialog', {
  t: {
    type: 'TABLE',
    prompt: 'Enter names and ages',
    tableHeader: [{name: 'Name', type: 'string'},{name: 'Age', type: 'int'}],
    tableRows: [['Alice', 42], ['Bob', 43]]
  }
});
```

LIST OF

The LISTOF input type is a compound input type. It can be used to allow the user to input multiple items each composed of a number of other input types. For instance; to input a number of measurements we could make a configuration as follows:

```
var results = Dialog.input('List of example', 'Input a number of measurements',
  measurements: {
    prompt: 'Add measurements here',
    type: 'LISTOF',
    template: {
      mtype: {
        prompt: 'Measurement type',
        type: 'RADIO',
        selectBetween: ['TypeA', 'TypeB']
      },
      mvalue: {
        prompt: 'Measurement value',
        type: 'TEXT'
      }
    },
    maxItems: 5,
    maxHeight: 200,
    initialItems: 2
  }
});
```

Which will result in the following dialog being shown:

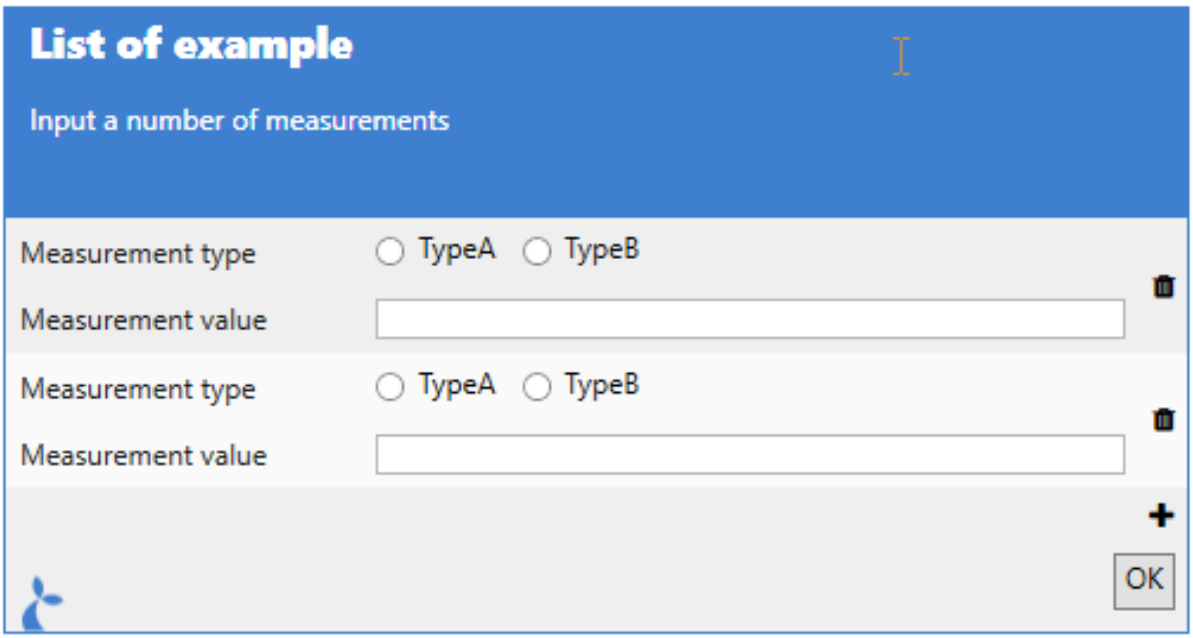

Figure 2: LISTOF dialog input

The property values available for a `LISTOF` input is:

- `template` which contains an object defining a single input element
- `maxItems` the max number of items a user is allowed to input,
- `maxHeight` the max height of the `LISTOF` input
- `initialItems` the number of initial items in the `LISTOF` list

DIVIDER

The `DIVIDER` type does not support any options.

SPACER

Has no options either, will provide some vertical space.

Validation

Input fields may have a validation object in their *options* which determines valid values for the inputs. The validation object has the following properties;

- `isRequired` boolean value indicating whether a value **must** be supplied for the field,
- `regex` is a regular expression which must match the given input in order for the field to validate,

- `message` is an *optional* message to be displayed in case validation fails.

Use either `isRequired` or `regex`, not both at the same time.

Example

```
var result = Dialog.input(
  'This is a demo',
  'Some description goes here.', {
    'submitOnValidation': true,
    'maxDialogHeight': 1000,
    'maxDialogWidth': 2000,
    'name': {
      'prompt': 'Name',
      'type': 'TEXT',
      'suffix': 'mm'
    },
    'colorRadio': {
      'prompt': 'Choose color',
      'type': 'RADIO',
      'selectBetween': ['red', 'green', 'blue']
    },
    'foo': {
      'prompt': 'Show only on blue',
      'dependsOn': 'colorRadio=blue',
      'type': 'TEXT'
    },
    'colorCombo': {
      'prompt': 'Choose color',
      'type': 'SELECT',
      'selectBetween': ['red', 'green', 'blue'],
      'validation': {'isRequired': true, 'message': "Color must be selected"}
    },
    'header' : {
      'type': 'HEADER',
      'value': 'Header #1'
    },
    'desc': {
      'type': 'DESCRIPTION',
```

```
      'value': 'Super long description possible. When a moon hits your eye like
    },
    'date': {
      'type': 'DATE'
    },
    'multi': {
      'type': 'MULTITEXT',
      'prompt': 'Some complex texts',
      'texts': [
        { 'name': 'a', 'prefix': 'pre', 'suffix': 'suf', 'validation': { 'regex'
        { 'name': 'b', 'prefix': '>', 'suffix': '<' }
      ]
    }
  }
);
// Now use the input values for something
var name = result.name;
var eyecolor = result.colorRadio;
```

This will result in the dialog shown below.

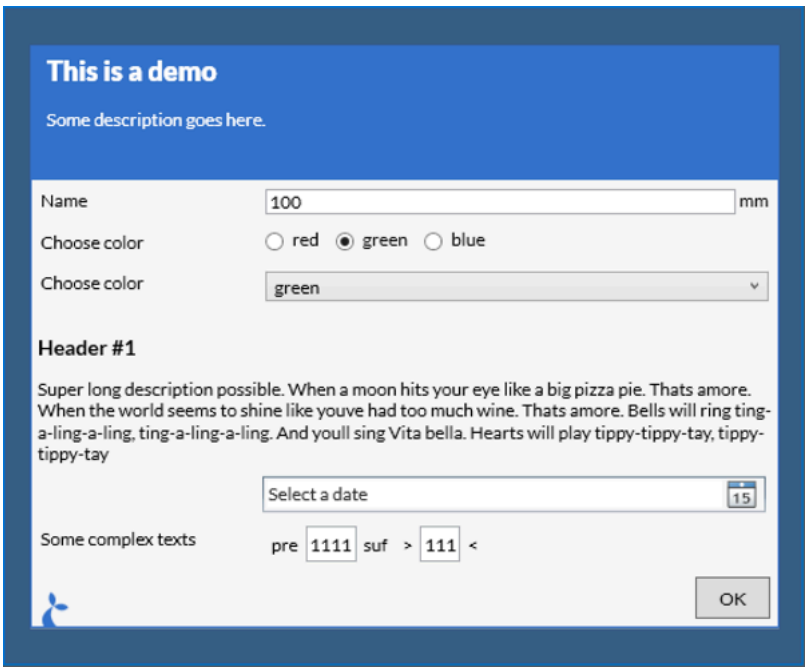

Figure 3: Input dialog examples

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.1.4 HTML based input dialog

In addition to the normal native input function we also support using HTML input forms. This approach does not bring as much built-in functionality - validation, conditional displays etc - but offers a larger degree of customization in the appearance of the displayed form. It works by taking the form, either HTML directly or a URL to a page containing the form and then displaying this in a dialog. When the user accepts the form (clicks ”ok”) the page is parsed and information about the contents of the individual fields are extracted for use in the flow.

The input values entered can be retrieved from the dialog result by using the `name` or `id` property of the input element. For more info on forms see e.g. https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms. For a concrete example with a number of different input elements see e.g. http://sirenia.eu/tutorial/form.html. `div` tags with a `input` tag will also be returned - the `id` of the `div` will be used as key.

Parameters

- `header` - [string] the header to display
- `text` - [string] a longer text to display
- `options` - [object] containing options for the dialog itself:
- `source` - [string] the form to display - either HTML directly or a URL
- `embed` - [bool] if true, manatee will add some styling and html/body tags to the page, if false nothing is added
- `maxDialogWidth` - [int] the max width the dialog must take
- `maxDialogHeight` - [int] the max height the dialog must take

Example

Source directly as an option.

```
var result = Dialog.inputHtml(
  'Header',
  'Some more text',
```

```
  {
    source: "<input type='text' id='myText'></input>",
    embed: true
  });
// The result will have a `myText` property since we added the `id` property wit
Debug.showDialog("Result was "+result.myText);
```

Using a remote document.

```
var result = Dialog.inputHtml(
  'Header',
  'Some more text',
  {
    source: "http://sirenia.eu/tutorial/form.html",
    embed: true
  });
// The result will have a `myText` property since we added the `id` property wit
Debug.ger(result);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.2 Flow

The flow object provides a mechanism to invoke other flows. This allows some flows to become *superflows* connecting multiple flows together. Flows from other applications may also be invoke in this fashion.

3.2.1 Shared functionality

You can use the `include(...)` method to include code from a `MODULE` typed flow. This is great if you have some code that you want to share between multiple flows.

The code in the module flow can export its functionality by assigning variables to the global `exports` object. See the example below.

Parameters

- `name` the name or subject of the module to include

Examples

We'll define a handy `math` module (given the `subject` = `math`):

```
var times = function(a, b) {
  return a*b;
}

var plus = function(a, b) {
  return a+b;
}

var bigNumber = 10000;

exports.times = times;
exports.plus = plus;
exports.bn = bigNumber;
```

and this can then be used in another flow:

```
var math = Flow.include('math');
var ten = math.times(2, 5);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.2.2 Run flow

Run another flow with the `run(...)` method. You provide the input to the flow and will get the outputs of the flow.

Parameters

- `name` the name of the flow to run - if there are 0 or more than 1 flow with this name an `Error` will be thrown
- `environment` is a JavaScript object containing the input to the flow. Each property on the object will be mapped to an input. Currently only string values are supported. Inputs are accessed in the running flow with `Inputs["<inputname>"]` e.g. `Inputs["myinput"]` or simply `<inputname>` e.g. `myinput` (if the is a valid JavaScript identifier).

Examples

Simple

```
var result = Flow.run("MyOtherFlow", { "inputA": "AAA", "inputB": "BBB" });
// "MyOtherFlow" will now get executed, the inputs may be accessed via e.g. Inpu
var outputC = result.outputC; // providing "MyOtherFlow" has a defined output ca
```

Chaining flows

It is possible to chain flows like:

```
var result = Flow.run("RunLast", Flow.run("RunFirst", {}));
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.3 Wait

3.3.1 Wait for seconds

Wait the given amount of seconds.

Parameters

- `timeout` the number of seconds to wait

Example

```
Wait.forSeconds(2);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.3.2 Wait for milliseconds

Wait the given amount of milliseconds.

Parameters

- `timeout` the number of milliseconds to wait

Example

```
Wait.forMilliseconds(200); // Wait for 0.2 seconds
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.3.3 Wait for field

Wait for the given field to appear - will return when field appear or throw an exception when the given amount of seconds has elapsed.

Parameters

- `field` the field to wait for e.g. `Fields["myfield"]`
- `timeout` the max amount of seconds to wait for the field to appear

Example

```
Wait.forField(Fields["myfield"], 10);
```

Support

- ChromeDriver
- JavaDriver

3.3.4 Wait for field to disappear

Wait for the given field to disappear - will return when field disappears or throw an exception when the given amount of seconds has elapsed.

Parameters

- `field` the field to wait for e.g. `Fields["myfield"]`
- `timeout` the max amount of seconds to wait for the field to disappear

```
Wait.forFieldToDisappear(Fields["myfield"], 10);
```

Support

- ChromeDriver
- JavaDriver

3.3.5 Wait for window

Wait for the given window to appear - will return when a matching window appears or throw an exception when the given amount of seconds has elapsed. There is also a `forWindowToDisappear` variant.

Parameters

- `title` the title of the window to wait for
- `timeout` the max amount of seconds to wait for the field to appear

Example

```
// Wait for a window with Notepad in its title to appear, max 10s
Wait.forWindow("Notepad", 10);
// Wait for Notepad to disappear again
Wait.forWindowToDisappear("Notepad", 10);
```

3.4 Xml

The Xml module enables parsing information stored in local or remote xml files.

3.4.1 Load xml

Parse the given string as xml and return an XmlDoc object which can be queried or turned into JSON.

Parameters

- `xml` an xml formatted string to parse

Example

```
var d = Xml.load("<hello>world</hello>");
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.4.2 Load XML from url

Fetch a local or a remote file and parse as xml. Returns an XmlDoc object.

Parameters

- `url` is a local or remote path to an xml file

Example

```
// A remote file
var remote = Xml.loadFrom("http://somewhere/over/the/rainbow.xml");
// A local file
var local = Xml.loadFrom("c:\\somewhere\over\the\rainbow.xml");
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.5 XmlDoc

An XmlDoc is an object that wraps an xml document and which has a few functions for querying the underlying document.

3.5.1 XPath

Execute an XPath query and return the results. The result is a list of objects, each object represents the matching xml node.

Parameters

- `xpath` a well-formed XPath expression

Example

```
var doc = Xml.load("<hello>world</hello>");
var allHellos = doc.xpath("//hello");
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.5.2 JSON

Returns a JSON/JavaScript version of the document which can then be inspected in the flow.

Example

```
var doc = Xml.load("<hello>world</hello>");
var docObject = doc.json();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.6 HTTP

The Http module enables http requests to be sent within a flow.

3.6.1 GET

Send a HTTP GET request. Returns a `reply` object containing;

- `status` the http status-code
- `data` a string containing the data received
- `headers` an object containing the headers received

Parameters

- `url` the url to GET
- `opts` options, an object which may contain the following properties:
  - `credentials` (optional) for basic-auth - an object containing;
  - `user` username for the http resource
  - `pass` password for the http resource
  - `headers` (optional) an object defining additional headers to include in the request
  - `useragent` (optional) a string overriding the default useragent
  - `timeout` (optional, default 60000) how many ms to wait for the request to complete
  - `contenttype` (optional) the contenttype of the request

Example

```
// Anonymous
var reply = Http.get("http://somewhere/over/the/rainbow.txt", {});
if (reply.status == 200)  { // Status: OK
  ...
}
// With basic-auth user/pass
Http.get("http://somewhere/over/the/rainbow.txt", { 'credentials': {'username':
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.6.2 POST

Send a HTTP POST request. Returns a `reply` object containing;

- `status` the http status-code
- `data` a string containing the data received

Parameters

- `url` the url to POST to
- `data` a string to POST
- `opts` options, an object containing additation options for the request (see description in Http.get)

Example

```
// Anonymous
var reply = Http.post("http://somewhere/over/the/rainbow.txt", "data=123", {});
if (reply.status == 200)  { // Status: OK
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.6.3 PUT

Send a HTTP PUT request. Returns a `reply` object containing;

- `status` the http status-code
- `data` a string containing the data received

Parameters

- `url` the url to PUT to
- `data` a string to PUT

- `opts` options, an object containing addtation options for the request (see description in Http.get)

Example

```
// Anonymous
var reply = Http.put("http://somewhere/over/the/rainbow.txt", "data=123" {});
if (reply.status == 200)  { // Status: OK
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.6.4 DELETE

Send a HTTP `DELETE` request. Returns a `reply` object containing;

- `status` the http status-code
- `data` a string containing the data received

Parameters

- `url` the url to DELETE
- `opts` options, an object containing addtation options for the request (see description in Http.get)

Example

```
// Anonymous
var reply = Http.delete("http://somewhere/over/the/rainbow.txt", {});
if (reply.status == 200)  { // Status: OK
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.7 FTP

The Ftp module enables reading and writing files on ftp servers.

3.7.1 Read

Read a file.

Parameters

- `url` the url to the file to read
- `opts` options, an object which may contain the following properties:
  - `user` username for the ftp server, blank if anonymous access is allowed
  - `pass` password for the ftp server

Example

```
// Anonymous
var data = Ftp.read("ftp://somewhere/over/the/rainbow.txt", {});
// With user/pass
var data = Ftp.read("ftp://somewhere/over/the/rainbow.txt", { 'user': 'John', 'p
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.7.2 Write

Write a file to a remote ftp server.

Parameters

- `url` the url to the file to write
- `data` the content of the file
- `opts` options, an object which may contain the following properties:
  - `user` username for the ftp server, blank if anonymous access is allowed
  - `pass` password for the ftp server

Example

```
Ftp.write("ftp://somewhere/over/the/rainbow.txt", "red, green, blue", {});
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.8 Db

The Db module has functionality for connecting to databases. It currently supports `sqlite`, `mssql` and `msaccess` databases.

3.8.1 Connect

The `connect` method initialises a connection to a given database and returns a Database object.

Parameters

- `type` the type of the database, currently this should be "mssql", "sqlite" or "msaccess".
- `connection` the connection-string which contains information about how to connect to the database in question

Example

```
var db = Db.connect('sqlite', 'Data Source=C:\\MyFolder\\Test.db;Version=3;');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.8.2 Database

The database object returned from a `Db.connect(...)` invocation represents a database connection. It has two primary methods for interacting with a database; `query` and `exec`.

Exec

The `exec` method will execute a non-query (e.g. `INSERT`, `UPDATE`) and return the number of affected rows.

Example

```
var affectedRows = db.exec('CREATE TABLE Test (id int, name string)');
```

Query

The `query` method is used for queries (e.g. `SELECT` etc) and returns an array of objects representing the result of the query.

Example

```
var rows = db.query('SELECT id, name from Test');
for (var i=0; i<rows.length; i++) {
  Debug.showDialog("id="+row.id+", name="+row.name);
}
```

Begin

The `begin()` method is used to initiate a transaction.

Example

```
var tx = db.begin();
```

3.8.3 Transaction

A `transaction` object is conceptually similar to the database object. It has the same `query` and `exec` methods, but will delay the execution of the query or command until `commit()` is invoked and of course maintains transactional integrity. If the `rollback()` method is invoked the `query` and `exec` operations already made are discarded.

Commit

A `commit()` invocation will commit the tx to the db.

Example

```
tx.exec("INSERT INTO Test (id, name) VALUES (1, 'John')");
tx.exec("INSERT INTO Test (id, name) VALUES (2, 'Jane')");
// Commit John and Jane
tx.commit();
```

Rollback

A `rollback()` invocation will rollback the tx.

Example

```
tx.exec("INSERT INTO Test (id, name) VALUES (1, 'John')");
tx.exec("INSERT INTO Test (id, name) VALUES (2, 'Jane')");
// John and Jane are not needed anyways
tx.rollback();
```

3.8.4 Database

The database object returned from a `Db.connect(...)` invocation represents a database connection. It has two primary methods for interacting with a database; `query` and `exec`.

Exec

The `exec` method will execute a non-query (e.g. `INSERT`, `UPDATE`) and return the number of affected rows.

Example

```
var affectedRows = db.exec('CREATE TABLE Test (id int, name string)');
```

3.9 Csv

The `Csv` module can be used for parsing, manipulating and generating comma-separated files.

3.9.1 Parse

The `parse` method takes a csv formatted string and returns an array of objects or arrays - one for each row in the string. There is also a `parseFile` variant which is identical to the `parse` method except that it takes a filename as its first argument.

Parameters

- `content` the csv string
- `options` provides the options for the parser

The `options` object can have the following fields:

- `delimeters` a list strings used to separate the columns of the content - default is `[',',';']`
- `header` can be set to
- `true` to indicate that a header is present in the first line of the content or you can set it to an
- array of strings to provide the header manually (the first line is treated as normal data) or you can
- leave it or or set it to `null` (the default) which will cause the parsed result to be an array of arrays instead of an array of objects
- `quotedFields` which will strip quotes from the data (if present in the content) - default `false`

Examples

```
var csv = Csv.parse('foo;bar\n100;200', {header: true})
```

The csv variable will now contain:

```
[
  { foo: 100, bar: 200 }
]
```

or if there is no header:

```
var csv = Csv.parse('100;200\n300;400', {})
```

The csv variable will now contain:

```
[
  [ 100, 200 ],
  [ 300, 400 ]
]
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.9.2 Stringify

The `stringify(arr, quoteStrings, delim)` method will take an array of objects or an array of arrays generate a csv string.

Parameters

- `arr` the array to convert to a csv string
- `quoteStrings` a boolean value indicating whether to add quotes to strings or not (default `false`)
- `delim` the delimeter string to separate fields (default `','`)

Example

```
var arr1 = [['foo','bar'],[1,2]];
var arr2 = [{foo: 3, bar: 4}];
var csvStr1 = Csv.stringify(arr1);
var csvStr2 = Csv.stringify(arr2);
```

`csvStr1` and `csvStr2` will now both have the value `foo,bar\n1,2`.

3.10 Excel

3.10.1 Load

Load and parse and Excel spreadsheet. It can either return the entire spreadsheet or a selected range of cells. If the `header` option is set then the returned value will be be a map/object with the column names as keys - otherwise an array is used. If `index` is set then then values in the index column will be used as keys - otherwise an array is used. If both are set then both dimensions will use values as keys. See the examples below.

Parameters

- `file` path for an Excel spreadsheet to load
- `options` options for parsing the spreadsheet - use `{}` to return the entire spreadsheet
- `table` define a table to return
  - `range` which range does the table reside in e.g. `'A1:D20'`
  - `header` is a boolean to determine if the top row of the header is a table
  - `index` is a boolean to determine if the initial column is an index column
  - `worksheet` is the name of the sheet to load data from

Example with simple table

Given the following simple spreadsheet in the worksheet named 'Sheet1':

cell 1	cell 2
cell 3	cell 4

The following code will load the spreadsheet and pick out the value stored at `cell1`.

```
var table = Excel.load('myspreadsheet.xlsx', {});
var cell1 = table["Sheet1"][0][0];
```

Example with table with header defined by range

Given the table below, situated in worksheet "Sheet1" at `A1:B3`:

header 1	header 2
cell 1	cell 2
cell 3	cell 4

Use the following code to pick out `cell4`.

```
var table = Excel.load('myspreadsheet.xlsx', { table: { range: 'A1:B3', workshee
var cell4 = table[2]['header 2']; // 3rd row (0 is first row), column with heade
```

Example with both header and index

Given the table below, situated in worksheet "Sheet1" at `A1:B3`:

	header 1	header 2
I1	cell 1	cell 2
I2	cell 3	cell 4

Use the following code to pick out `cell2`.

```
var table = Excel.load('myspreadsheet.xlsx', { table: { range: 'A1:C4', workshee
var cell2 = table['I1']['header 2'];
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.10.2 Delete a sheet

Removes a single sheet from the workbook.

Parameters

- `filename` the path to the excel file to be updated
- `sheet` the name of the sheet to delete

Example

```
Excel.deleteSheet('data.xlsx', 'Sheet1');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.10.3 Update single cell

Update the value stored in a **single cell** in a spreadsheet.

Parameters

- `filename` the path to the excel file to be updated - if the file does not exist a new one will be created
- `sheet` the name of the sheet to update
- `address` an "address" to a cell, e.g. "A1"
- `value` the value to write into the cell

Example

```
// write 1000 into A3 of Sheet1 in data.xlsx
Excel.updateCell('data.xlsx', 'Sheet1', 'A3', 1000);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.10.4 Update multiple cells

Update values stored in a spreadsheet. This method is a lot more performant than the single cell version if you need to store multiple values.

Parameters

- `filename` the path to the excel file to be updated - if the file does not exist a new one will be created
- `sheet` the name of the sheet to update
- `address` an ”address” of the starting cell
- `values` the valued to write into the cells - this should be a 2 dimensional array (like a table)

Example

```
// The data to write
var data = [
  [10, 20, 30],
  [40, 50, 60]
];
// write data into data.xlsx, Sheet1 starting at A1
Excel.updateCells('data.xlsx', 'Sheet1', 'A1', data);
```

This will result in a table that looks like:

	A	B	C
1	10	20	30
2	40	50	60

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.11 Settings

The `Settings` object contains values that can be read/written to affect the behaviour of a flow. The following properties are available:

- `CommandRetries` (int - read+write) defines the number of times a command is retried before it is considered to fail. Default is 3.
- `CommandRetryDelays` (Array - read+write) defines the delays in milliseconds between each retry. Default is `[100, 200, 400, 800, 1600]`. When the number of retries exceed the given delays the last value in this array is used for all overflowing retries.

3.11.1 Example writing a value

```
Settings.CommandRetryDelays = [100, 100, 100];
```

3.11.2 Example reading a value

```
var retries = Settings.CommandRetries;
Debug.showDialog("Retries: " + retries);
```

3.12 Log

3.12.1 Warn

Inserts a warning in the log.

Parameters

- `key` the key of the message - keep this as a constant
- `text` the text to insert

Example

```
Log.warn('greeting','hello there');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.12.2 Info

Inserts a informational line in the log.

Parameters

- `key` the key of the message - keep this as a constant
- `text` the text to insert

Example

```
Log.info('greeting','hello there');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.12.3 Set log level

Controls the log verbosity of the application driver.

Parameters

- `level` the new log level. Must be one of the following: none, fatal, error, warn, info, debug.
- `options` optional additional options
  - `useStdOut` (defaults to false) boolean value indicating if instrumentation log should go to the application stdout or to manatee log.

Example

```
Log.setDriverLogging('info', { useStdOut: true });
```

Support

- ChromeDriver
- JavaDriver

3.13 Window

The Window module has functionality for dealing primarily with the main window of an application. In contrast the Windows module supports interacting with all the windows on the desktop.

3.13.1 Title

Get the title of the main window. Optionally supply a timeout for the operation - default timeout is 500ms normally.

Example

```
var title = Window.title();
// or with a timeout of 2s
title = Window.title(2000);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.2 Minimize

Minimize the main window.

Example

```
Window.minimize();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.3 Is minimized

Check if the main window is minimized.

Example

```
if(Window.isMinimized()) {
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.4 Maximize

Maximize the main window.

Example

```
Window.maximize();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.5 Is maximized

Check if the main window is maximized.

Example

```
if(Window.isMaximized()) {
  ...
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.6 Focus

Put focus on the main window.

Parameters

- `options` optional object with options for focus. Supported options:
  - `useCachedUI` boolean indicating if UI component lookup should use the UI itself or the underlying model. Defaults to `false` (underlying model traversal).

Example

```
Window.focus();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.7 Send keys

Send keyboard events (simulated typing) to a window. Supports special strings for sending special keys.

Parameters

- `keys` the keys to send - this is a string
- `options` optional object with options for sendkeys, supported options:
  - `focus` [bool] whether to focus the window prior to sending the keys

Example

```
Window.sendKeys("foo bar");
// or to focus the window prior to sending the keys
Window.sendKeys("foo bar", { focus: true });
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.8 Restore

Restore the main window to a previous state and location.

Example

```
Window.restore();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.13.9 Window with modal dialog shown

Get whether or not a modal (dialog) is shown.

Example

```
var modalIsShown = Window.modalShown();
```

Support

- JavaDriver

3.13.10 Shown with title

Get whether or not a window with the given title is shown.

Example

```
var windowIsShown = Window.withTitleShown("My Window");
```

Support

- JavaDriver

3.13.11 Dim

Dims the window owned by the flow.

Parameters

- `level` the amount of dimming, 0-255. 255 is opaque and 0 is transparent.

Example

```
Window.dim(100)
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.14 Windows

The Windows module has functionality to inspect and manipulate the Windows of the desktop.

3.14.1 All windows

The `all()` method will return an array of window proxy objects representing all windows on the desktop.

Example

```
var allWindows = Windows.all();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.14.2 Windows for current application

The `forApp()` method returns an array of window proxy objects representing all the windows of the application.

Example

```
var applicationWindows = Windows.forApp();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.14.3 Primary window

The `primary` property returns a single window proxy object representing the primary or main window of the application.

Example

```
var pw = Windows.primary;
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.14.4 Frontmost/focused window

Get the frontmost or focused window with this command.

Example

```
var w = Windows.focused;
// and the same can be done via
w = Windows.frontMost;
```

3.14.5 Window Proxy

The window proxy object returned by the `Windows` module methods represents a desktop window and can be manipulated with the following methods and properties.

Move

Move the window to the given `x`, `y` coordinates.

```
var pw = Windows.primary;
// Move the window to (100,100) from the topmost left corner of the screen.
pw.move(100, 100);
```

Resize

Resize the window to the given dimensions.

```
var pw = Windows.primary;
pw.resize(100, 100);
```

Focus

Make the window the focused (topmost) window.

```
var pw = Windows.primary;
pw.focus();
```

Maximize

Maximize the window.

```
var pw = Windows.primary;
pw.maximize();
```

Minimize

Minimize the window.

```
var pw = Windows.primary;
pw.minimize();
```

Restore

Restore the original state of the window (after having maximized or minimized it).

```
var pw = Windows.primary;
pw.restore();
```

Screenshot

Grab a screenshot of the window. The screenshot will be returned as a base64 encoded string.

```
var pw = Windows.primary;
// img is a base64 encoded string
var img = pw.screenshot();
```

SendKeys

Send keyboard strokes to the window.

```
var pw = Windows.primary;
pw.sendKeys("abc");
```

Title

Get the title of the window.

```
var pw = Windows.primary;
var t = pw.title;
```

Class

Get the class of the window.

```
var pw = Windows.primary;
var t = pw.class;
```

IsPrimary

Get/set whether this window is considered the primary for the application.

```
var ws = Windows.forApp();
if (!ws[0].isPrimary) {
  ws[0].isPrimary = true;
}
```

IsMaximized

Get a boolean value indicating whether or not the window is maximized.

```
var ws = Windows.forApp();
if (!ws[0].isMaximized) {
  // do something then
}
```

IsMinimized

Get a boolean value indicating whether or not the window is minimized.

```
var ws = Windows.forApp();
if (!ws[0].isMinimized) {
  // do something then
}
```

Bounds

Get/set the bounds (location and size) of the window.

```
var pw = Windows.primary;
var bounds = pw.bounds;

// Move 10px left and down
bounds.x = bounds.x + 10;
bounds.y = bounds.y + 10;
// Decrease width and height with 10px
bounds.width = bounds.width - 10;
bounds.height = bounds.height - 10;
```

```
// Update the window bounds with the new values
pw.bounds = bounds;
```

3.15 Processes

The Processes module similarly to the windows module is used to enumerate and manipulate processes running on the local machine.

3.15.1 All processes

The `all()` methods enumerates all processes on the local machine. It returns an array of process proxy objects.

```
var ps = Processes.all();
for (var i=0; i<ps.length; i++) {
  // then do something with each process proxy
}
```

3.15.2 Current

Get the current process for the application (for which the flow is defined).

```
var p = Processes.current;
Debug.showDialog(p.name);
```

3.15.3 Spawning new processes

The `spawn(...)` method can be used to create new processes. It takes 3 arguments;

- `path` to the executable to launch
- `arguments` for the executable (*optional - default null*)
- `working directory` (*optional - default null*)
- `shell` (boolean) whether to launch the process in a shell environment - this must be set to true for url-handlers to activate (*optional - default false*)

It returns a process proxy object fronting the process spawned.

```
var p = Processes.spawn("C:\\Path\\To\Executable.exe");
Debug.showDialog(p.name);
```

3.15.4 Process proxy

Kill

Kills a process.

```
var p = Processes.all()[0];
p.kill();
```

Wait for a process to exit

The `wait(...)` method will wait for the given process to exit. It takes an integer, the maximum number of milliseconds to wait for the process as its argument. It returns a boolean indicating whether the processes exited (`true`) or the given timespan elapses (`false`).

```
// Wait max 1s for the first process to exit
if (Processes.all()[0].wait(1000)) {
  // it exited
} else {
  // 1s elapsed
}
```

Send input (via standard-in)

Sending some input to a running process is achieved with the `stdin(...)` method.

This can normally only be done for processes spawned by yourself via the `Processes.spawn(...)` *](#spawning-new-processes) method.*

```
var p = Processes.spawn(...);
p.stdin("hello");
```

Read from process output (standard-out)

Reading from the output of a process is done via the `stdout(...)` method. It takes an int - the number of lines to read - and returns a task which completes with the lines read as an array of strings once the given number of lines has been read.

This can normally only be done for processes spawned by yourself via the `Processes.spawn(...)` *](#spawning-new-processes) method.*

```
var p = Processes.spawn(...);
var lines = null;
// Read 3 lines, then kill the process
p.stdout(3).then(function(threelines) {
  lines = threelines;
  p.kill();
});
p.wait(20000);
Debug.ger(lines);
```

It is also possible to read from standard-error output - simply use the `stderr(...)` method instead of `stdout(...)`.

Process Id

Get the id of the process.

```
var pid = Processes.current.id;
```

Process name

Get the name of the process.

```
var pname = Processes.current.name;
```

Process path

Get the path of the executable for the process.

```
var path = Processes.current.path;
```

Process working directory

Get the working directory of the executable for the process.

```
var pwd = Processes.current.wd;
```

Process mem usage

Get the virtual or private memory (integers) usage of the process.

```
var virtualMem = Processes.current.vmem;
var privateMem = Processes.current.pmem;
```

Process exited?

Gets a boolean indicating whether the process has exited.

```
if (Processes.current.exited) {
  // whoops
}
```

Process uptime

Gets the number of milliseconds elapsed since the process was spawned (as longs as it has not exited).

```
var uptime = Processes.current.uptime;
```

Process arguments

Get the arguments supplied to the process - can only be counted on to return valid arguments if process was spawned by Manatee.

```
var args = Processes.current.arguments;
```

3.16 Debug

3.16.1 Show dialog

Show some text in a debug dialog. Essentially the same as `Dialog.info("Debug", text)`.

Parameters

- `text` the text to display

Example

```
Debug.showDialog(”hello there”);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.16.2 ger

The `Debug.ger()` method pauses the running flow (as any other dialog) and shows a debugger dialog which includes an inspector and a REPL (read-eval-print loop).

Inspector

The inspector window lets you inspect the global values in the flow as well as the argument given. The variables are displayed in a tree which can be expanded to reveal the structure of the objects.

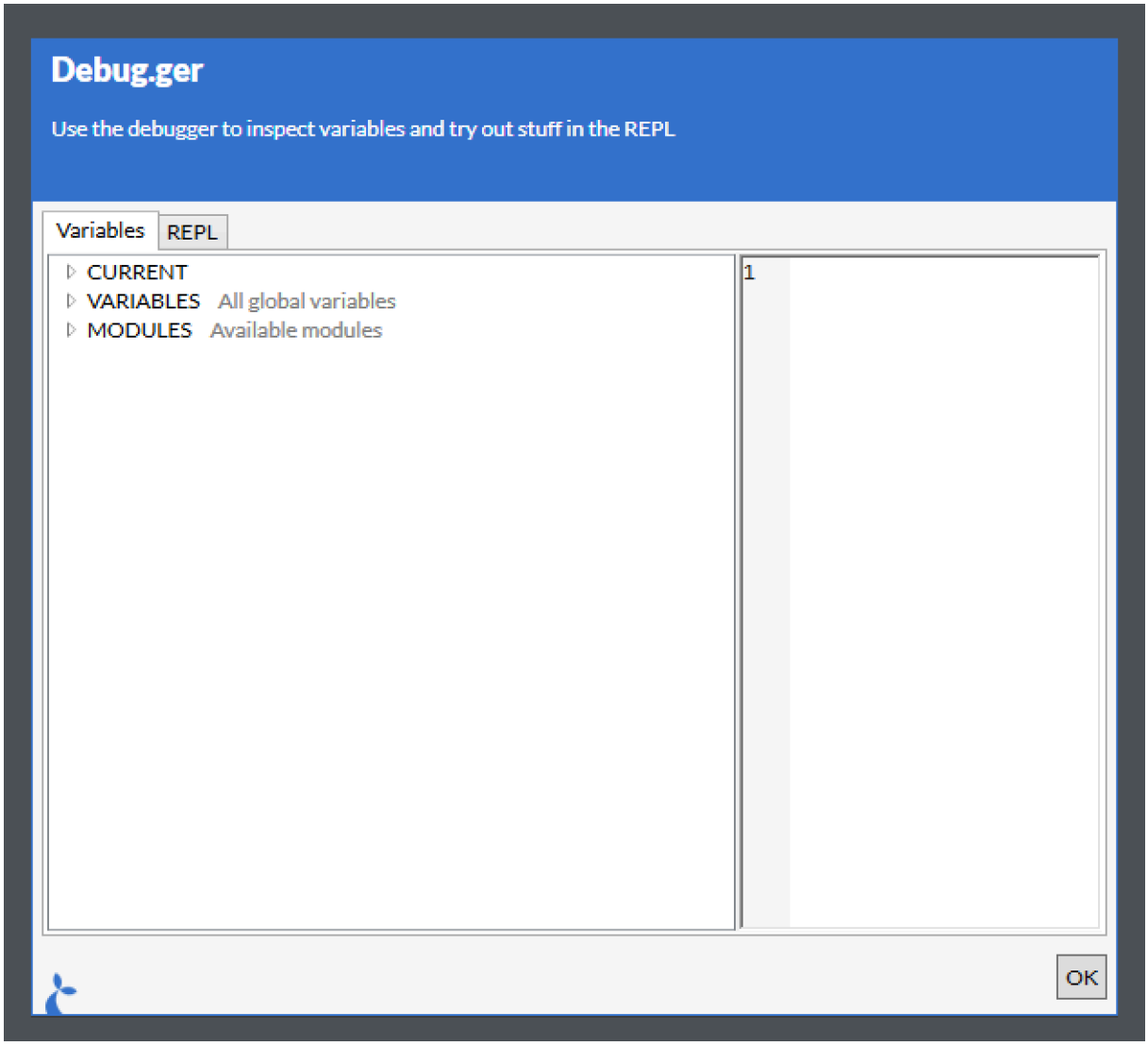

Figure 4

The debugger shown above was shown with the following code:

```
var x = { foo:'bar', baz: 123 };
Debug.ger(x);
```

Expanding the CURRENT node will give you:

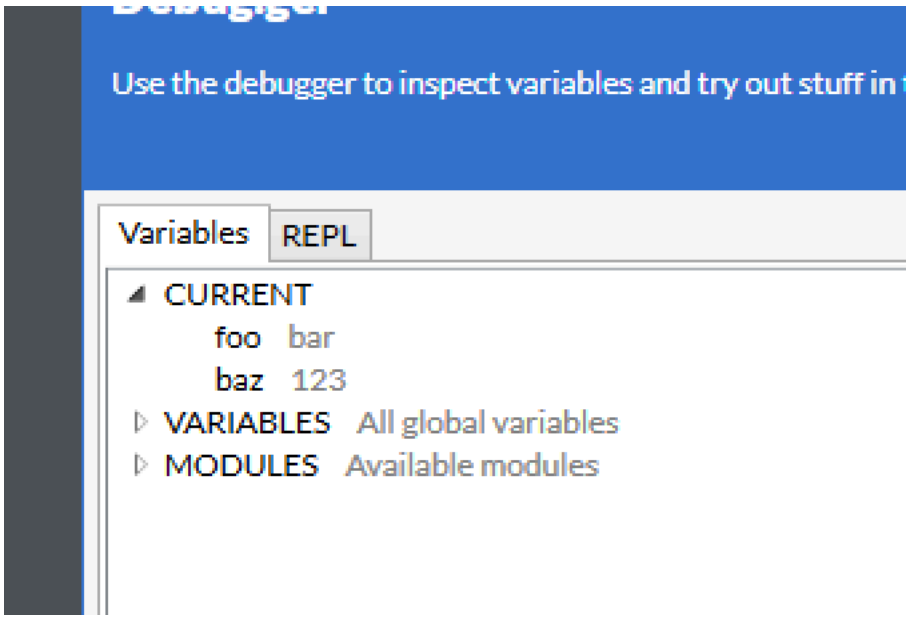

Figure 5

You can also explore the global variables (those defined in the outermost) scope of a flow. Here we show a field.

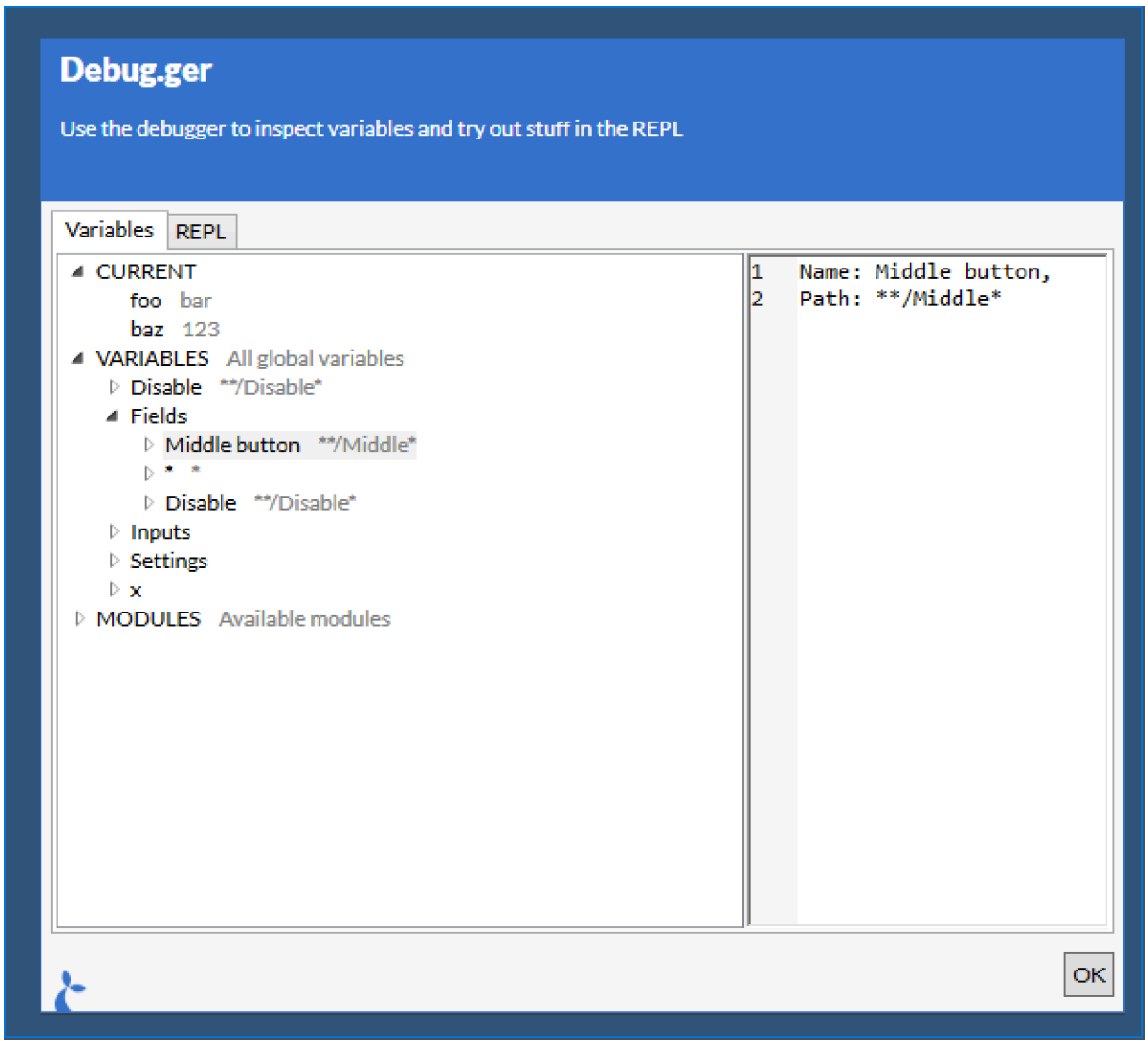

Figure 6

REPL

The REPL tab of the `Debug.ger` can be used to try running small snippets of code in the context of the current flow. You can do anything via the REPL that you can do in a flow.

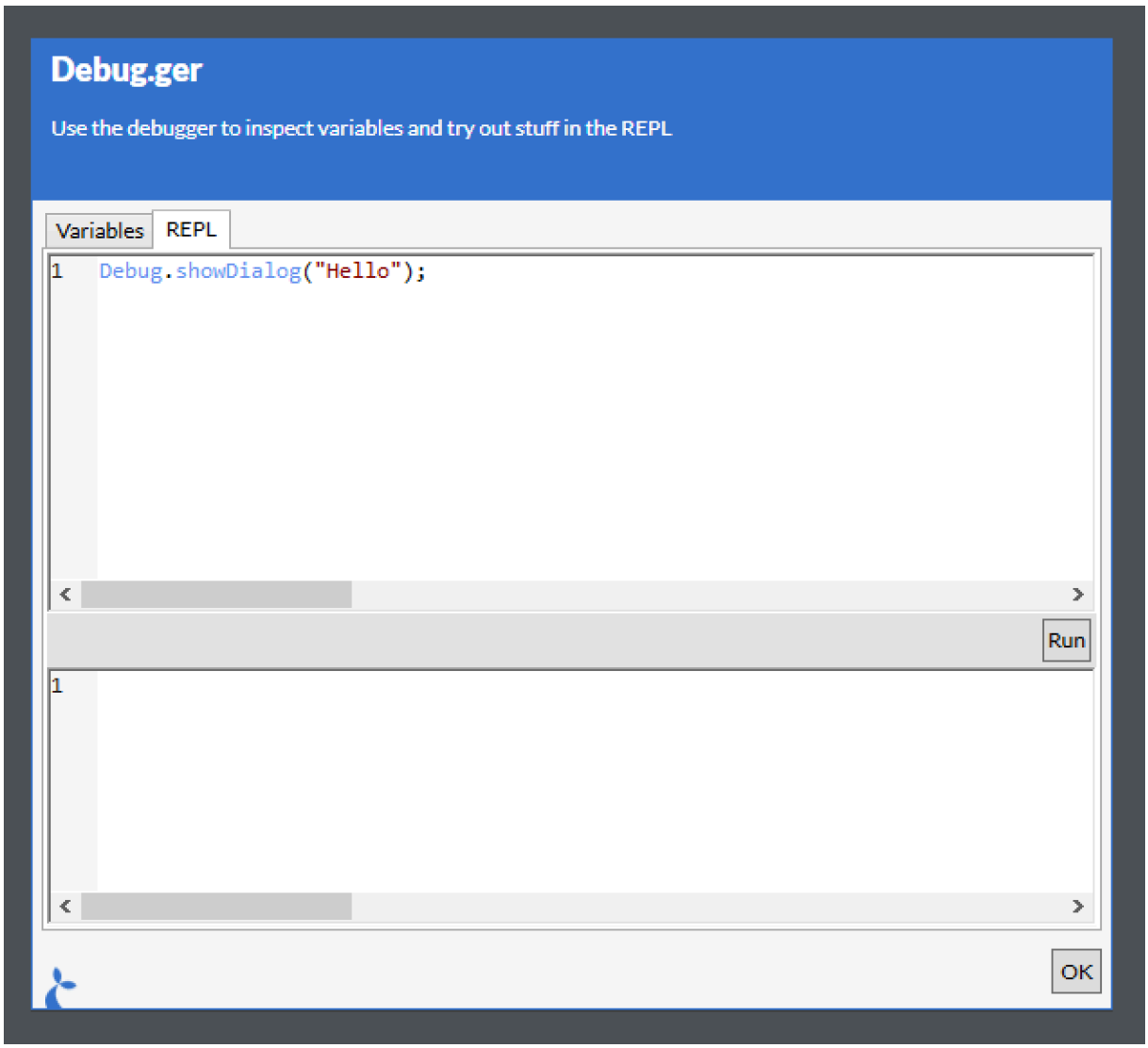

Figure 7

Clicking the ”Run” button will run the code written and display the time it took to run as well as the result.

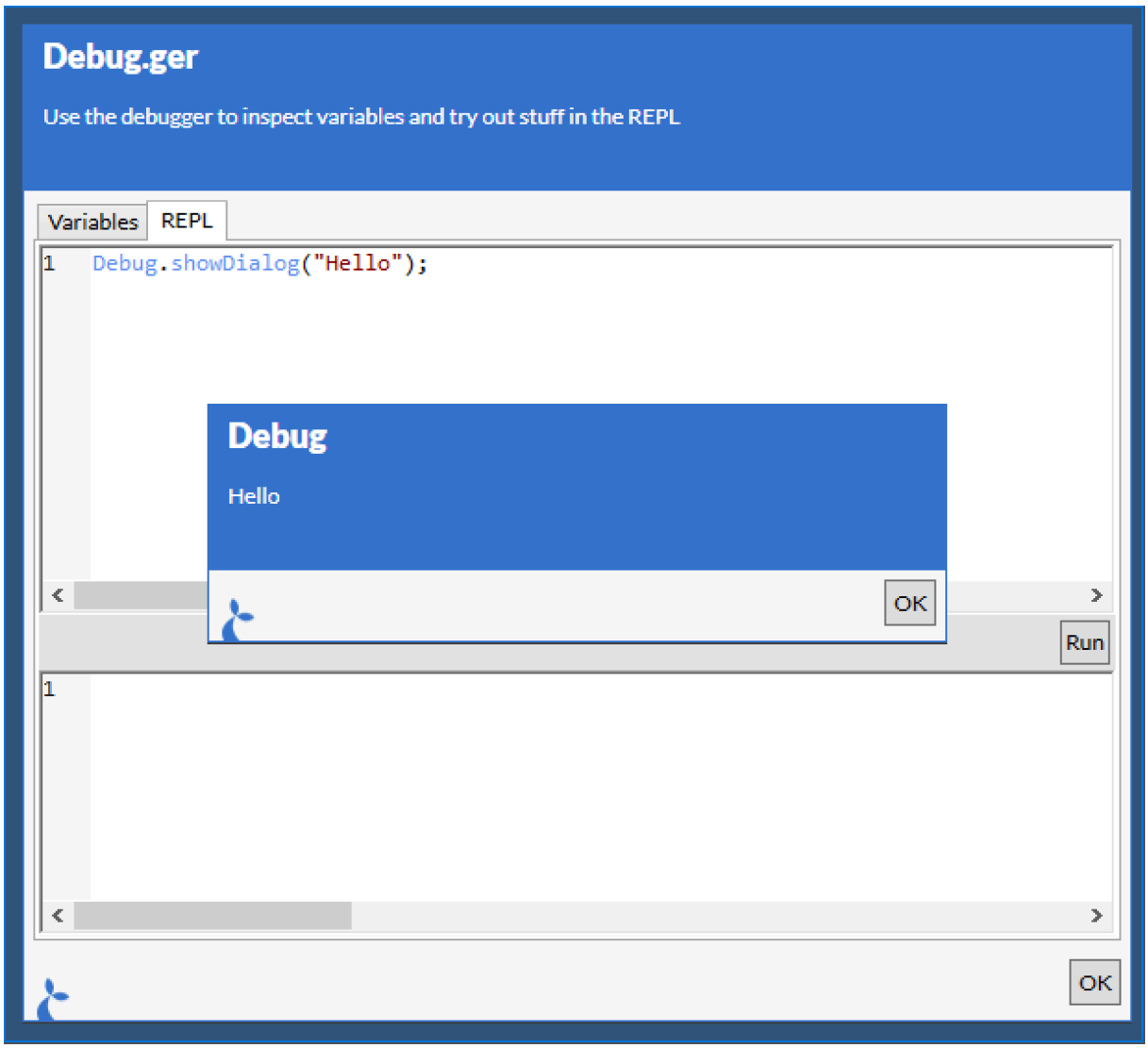

Figure 8

This method can also be used as `Debug.attach()` and `Debug.inspect()` but some of us prefer the simplicity and raw hipster essence of `Debug.ger()`.

Debug dialog size

You can pass an option object as the second argument. It accepts the following properties:

- `maxWidth`: Allows control of the width of the debug window

```
var s = 'data for the variable';
Debug.ger(s, { maxWidth: 1200 });
```

3.17 Fs

The Fs module is used to interact with the filesystem of the local machine.

3.17.1 System folders

Provides access to the following system folders: - `tmpfolder`: A directory for temporarily storing files - `desktop`: The user's windows desktop - `appdata`: The user app data folder. Applications can write user specific data here without requiring administrator privilege - `startup`: The folder which contains shortcuts to applications that should start when the user logs in - `personal`: The root user folder - eg `C:\Users\<user name>`

Example

```
var folder = Fs.tmpfolder;
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.2 List (ls)

Returns a list of files found in the directory given by the `path` argument. The path may contain wildcards `*` in its last segment.

A second option argument can be passed, which can have the boolean property `deepMatch`. When this property is set to true, files matching the filename given in the `path` argument in any sub-folder will be returned.

Default behavior is to do a shallow file listing.

Return value

The resulting array can be used as a string array of the paths to the files. It can also be used as an array of objects with detailed information about the files. Each such object has the following properties:

- `folder` is the folder part of the path. `C:\folder\file.txt` has the folder path `C:\folder`.
- `path` is the full path of the item. Corresponds to the string value of the object.
- `extension` is the extension of the item. `C:\folder\file.txt` has the extension `.txt`.
- `name` is the name of the item. `C:\folder\file.txt` has the name `file.txt`. `C:\folder` has the name `folder`.
- `readonly` boolean value indicating if the file is read only.
- `size` is the size of the file in bytes.
- `created` is the time of creation.
- `modified` is the time of the last modification.
- `accessed` is the time of the last file access.

The objects further have the following methods: - `mv` moves the file. Pass the new path as an argument. - `cp` copies the file. Pass the new path as an argument. - `rm` deletes the file. - `encrypt` encrypts the file. - `decrypt` decrypts the file.

Example

```
// Get all .txt files prefixed with somefile in somedir
var files = Fs.ls('c:\\somedir\\somefile*.txt');

// Get all .txt files in any sub directory under C:\somedir - at any depth
var files = Fs.ls('c:\\somedir\\*.txt', { deepMatch: true });

// Copy readonly files to a backup sub directory
var readonlyFiles = files.filter(function(file) { return file.readonly; });
_.each(readonlyFiles, function(file) { file.cp(file.folder + '\\backup\\' + file
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.3 Move file

Move a file to a different path

Example

```
Fs.mv('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.4 Copy file

Copy a file to a different path

Example

```
Fs.cp('C:\\some\\path\\file.txt', 'C:\\some\\other\\path\\file.txt');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.5 Delete file

Delete a file

Example

```
Fs.rm('C:\\some\\path\\file.txt');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.6 Check file presence

Determines if a file exists at a given path

Example

```
if (!Fs.exists('C:\\some\\path\\file.txt')) {
  // Create the file
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.7 Encrypt file

Activates windows file encryption for the file at the given path. Only the currently logged in user will be able to read the file.

Example

```
Fs.encrypt('C:\\some\\path\\file.txt');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.8 Decrypt file

Deactivates windows file encryption for the file at the given path. Any user will be able to read the file.

Example

```
Fs.decrypt('C:\\some\\path\\file.txt');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.9 Read

Read the contents of a file with the `read(...)` function.

Example

```
var html = Fs.read('c:\\somedir\\somefile.html');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.10 Write

Writes arbitrary text to an arbitrary text file. If the file exists, it will be overwritten. If the file doesn't exist, it will be created with the given contents. The contents are written using UTF-8 encoding without a byte order mark (BOM).

Throws appropriate exceptions if the write fails.

Example

```
Fs.write('c:\\somedir\\somefile.html', '<html><body><h1>Generated html!</h1></bo
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.17.11 Temp file

The `tmpfile` function will generate a random, non-conflicting filename in the temp folder.

Example

```
var tmpFilePath = Fs.tmpfile();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.18 App

The App variable contains functions relating to the app itself.

3.18.1 Location

Returns the current location (if applicable for the given application type -- non-webapps do not support this).

Example

```
var loc = App.location();
```

Support

- ChromeDriver
- IEDriver

3.18.2 Navigate

Navigates to the given url. If the url is relative (e.g. `somefolder/somefile.html`) it will get appended to the current url.

Parameters

- `url` a string representing the destination for the navigation act

Example

```
// Absolute url
App.navigate("http://dr.dk");

// Relative url
App.navigate("news");
```

Support

- ChromeDriver
- IEDriver

3.18.3 Session write

Store a value in the current session storage. This will be available across flows and for all applications.

Parameters

- `key` a string denoting the key to store the value under
- `value` an object to store
- `options` an optional options object. Supported options are;
  - `expires` a timeout in minutes - after this interval has passed the value will be deleted. Default is 1440 min (= 1 day).

Example

```
// Storing a simple value - a string
App.session().write('mykey', 'myvalue');

// Storing an object - expires in 1 hour
App.session().write('myotherkey', { greeting: 'hello' }, { expires: 60 });
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.18.4 Session read

Read a value stored in the current session.

Parameters

- `key` a string denoting the key to retrieve the value for

Example

```
var v = App.session().read('mykey'); // e.g. will return 'myvalue'
```

Support

- ChromeDriver
- IEDriver

- JavaDriver
- NativeDriver

3.18.5 Session delete

Delete a value.

Parameters

- key a string denoting the key to delete

Returns

The value deleted.

Example

```
var v = App.session().delete('mykey'); // e.g. will return 'myvalue'
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.18.6 Quit

Quits the application - **be aware that this is a hard shutdown and the user will not be prompted to save any information before the application exits**.

Example

```
App.quit();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.19 Sticky

A sticky is a persistent window which can be configured to remain top-most as long as it's shown. The user is able to interact with the items shown in the sticky e.g. clicking on links, opening pdf previews etc. Keyboard interaction is also supported, use:

Key	Action
↓	Focus next (down) item
↑	Focus last (up) item
`.` or `-`	Toggle collapsed state of item
`<space>`	Run primary action (depends on the type of the item)
`<enter>`	Run secondary action
`<esc>`	Close sticky (or exit from search if search field is focused)
any char	Open search field

3.19.1 Open

Open a new sticky window with the given `name` and `opts`. The function can be called multiple times with the same name argument in order to update an already showing sticky.

Parameters

- `name` the name of the window to open, only one sticky-window can have this name
- `opts` is an object containing the configuration for the sticky, it may have the following properties:
  - `embed` (boolean, default `false`) should the sticky be embedded in the window of its owner application? When embed is set to true some of the below options are not relevant

- `resizable` (boolean, default `false`) should it be possible to resize the sticky window
- `movable` (boolean, default `false`) should it be possible to move the sticky window
- `searchable` (boolean, default `false`) should the contents of the sticky be searchable
- `showFooter` (boolean, default `false`) should a foother with a nice logo be shown
- `fontSize` (int, default 12) the base font size to use for items in the sticky
- `focusOnUpdate` (boolean, default `false`) when the sticky is updated should we focus the sticky window again?
- `topMost` (boolean, default `false`) should the sticky be top-most *always*
- `title` the title of the sticky window
- `location` determining where the sticky should be shown, contains:
- `type` which type of position - currently only 'absolute' is allowed
- `top` px from the top of the screen
- `left` px from the left side of the screen
- `width` px width of sticky
- `height` px wheight of sticky
- `items` a list of sticky items to show in the window, each is defined by:
- `type` which type of item - *see below*
- *more depending on the type, see below*

Items

We support the following types of items.

GIF

The first is GIF which simply shows an (animated) gif - it may have the following properties:

- `source` an url for a gif, can be remote or local

ACTION

An ACTION will run the flow with the name given when the sticky is clicked. For the ACTION type the following are valid.

- `name` the name of the action to launch - this should be unique
- `header` and `body` if set these will be shown instead of action name on sticky
- `height` the height of the item in pixels
- `inputs` is an object containing the named inputs for the action
- `focus` whether or not the item should have focus (only the first item with this property set to true will be focused)

PDF document

Will show a pdf with an optional preview. The options are:

- `source` an url (remote or local) to the pdf to show
- `header` and `body` if set these will be shown instead of the source
- `linkText` an optional text (or unicode icon) to show as a link to the source file
- `link` an optional link to direct the user to (default is value of source)
- `height` the height of the preview pane in pixels
- `collapsible` whether or not the preview should be collapsible (default `false`)
- `collapsed` the initial state of the preview (default `false`)
- `saveable` whether or not it should be possible to save the pdf (default `true`)
- `printable` whether or not it should be possible to print the pdf (default `true`)
- `focus` whether or not the item should have focus (only the first item with this property set to true will be focused)

HTML page

Will render a HTML snippet or a whole HTML page into an item. Should be used for render styled text, e.g. headers and such - not recommended for complete pages. Options are:

- `source` html text or an url (remote or local) to the pdf to show
- `height` the height of the item
- `focus` whether or not the item should have focus (only the first item with this property set to true will be focused)

LINK

Will act as a link (e.g. to an internet resource or a local file).

- `link` the link to activate (when clicked)
- `text` optional - the text to display (default is the url of the link)
- `prefix` optional - the text to display before the link text
- `suffix` optional - the text to display after the link text
- `focus` whether or not the item should have focus (only the first item with this property set to true will be focused)

Example

```
Sticky.open(
  'mySticky',
  {
    embed: true,
    location: {
      type: 'absolute',
      top: 100,
```

```
      left: 100
    },
    items: [
      {
        type: 'GIF',
        source: 'http://gifs.com/cat'
      },
      {
        type: 'ACTION',
        name: 'SomeOtherAction',
        header: 'Some other action',
        body: 'Click to run'
      },
      {
        type: 'PDF',
        source: 'http://pdfworld.com/arandompdf.pdf',
        link: 'http://pdfworld.com/aboutarandompdf',
        height: 100,
        collapsible: true,
        collapsed: false,
        saveable: false,
        focus: true
      },
      {
        type: 'HTML',
        source: '<h1>Big header</h1><h2>Smaller header</h2>',
        height: 50
      },
      {
        type: 'LINK',
        link: 'http://sirenia.eu',
        prefix: 'Go to ', text: 'Sirenia', suffix: ' now'
      }
    ]
  }
);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.19.2 Model

Get the model used to construct the sticky,

Parameter

- name the name of the sticky to retrieve the model for (must be opened prior...)

Example

```
var m = Sticky.model('mySticky');
// Perhaps do some changes to model m and then
// Sticky.open('mySticky', m);
// to update the stikcy with the changes made to its model
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.19.3 Close

Close a named sticky.

Parameter

- name the name of the sticky to close (must be opened prior...)

Example

```
Sticky.close('mySticky');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.19.4 Hide

Hide a named sticky.

Parameter

- name the name of the sticky to hide (must be opened prior...)

Example

```
Sticky.hide('mySticky');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.19.5 Show

Show a previously hidden sticky.

Parameter

- name the name of the sticky to show (must be hidden prior...)

Example

```
Sticky.show('mySticky');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.20 Timer

The timer module provides a simple interface for timing parts of flows. It is especially useful in combination with our Analytics product allowing you to time crucial parts of your flows.

3.20.1 Start

Start a named timer. If you invoke this method twice with the same name (argument) you'll reset the timer every time.

Parameter

- name the name of the timer to start

Example

```
Timer.start('myTimer');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.20.2 Log

Log an event on a named timer. Useful only in combination with our Analytics product. The logged event will contain the name of the timer, the milliseconds since the timer was started and the given message.

Parameter

- `name` the name of the timer to log an event on
- `message` the message to log

Returns

The number of milliseconds since the timer was started.

Example

```
Timer.log('myTimer', 'A message goes here');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.20.3 Stop

Stop a named timer.

Parameter

- `name` the name of the timer to stop
- `log` whether or not a message should be logged

Returns

The number of milliseconds since the timer was started.

Example

```
// Will log an event and stop 'myTimer'
Timer.stop('myTimer', true);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.21 Notifications

The notifications module makes it possible to display non-interactive notifications.

3.21.1 Show

Shows a notification.

Parameter

- `name` the name of the notification, save this for future update invocations
- `header` the header text to show
- `body` the body text to show
- `options` is an object with the following additional options:
  - `severity` the severity of the notification, choose between "INFO", "WARN" and "ERROR". Default is "INFO".
  - `timeout` seconds for the notification to show. Default is 30.
  - `callback` a javascript function to execute when the user clicks the notification. Default null.
  - `embed` defines whether the notification should be embedded in the current application or shown on the desktop (default is `false` = show on desktop)
  - `sound` a string (one of `asterisk`, `beep`, `exclamation`, `hand`, `question`) which indicates a system sound to play once the notification is shown.

Example

Show an INFO notification for 30 seconds.

```
Notification.show('hello', 'Seasonal greetings!', 'Felice navidad', {});
```

Show a WARN for 5 seconds.

```
Notification.show('warn', 'Its complicated', 'Something broke down', { severity:
```

Notifications with callbacks.

```
function RaiseTheAlarm() {
  Notification.show('Oh no!', 'You clicked the first notification', { severity:
}

// Callback to previously defined function
Notification.show('warn', 'Its complicated', 'Something broke down, click here',

// Callback to anonymous function
Notification.show(
  'warn',
  'Its complicated',
  'Something broke down, click here',
  {
    severity: 'WARN',
    timeout: 5,
    callback: function() {
      Log.info('clicked', 'Notification was clicked');
    }
  });
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.21.2 Update

Update the information in an already shown notification.

Parameter

- `name` the name of the notification
- `header` the header text to change
- `body` the body text to change
- `options` is same as for invoking show

Example

Update the notification named ”hello”.

```
Notification.update('hello', 'Seasonal greetings anew!', 'Merry Christmas', {});
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.21.3 Close

Close an open notification. Notifications will automatically be hidden but this can force that action.

Parameter

- `name` the name of the notification

Example

Close the notification named ”hello”.

```
Notification.close('hello');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.22 Tasks

The Tasks module can be used to paralellize parts of a flow. This is useful for e.g. doing concurrent http requests or running background tasks. It is not intended for use with field-operations i.e. interacting with a host applications UI since this interaction cannot be parallelized. Furthermore you should not display dialogs in parallelized tasks as they can block the calling flow.

3.22.1 Run

Use the `run` method to start a new task.

Parameters

- `fun` a function to run in parallel

Returns

- a `Task` object.

Example

Run some tasks and wait for the result.

```
var t = Task.run(
  function() {
    var i = 0;
    while (i<1000) {
      i = i + 1;
    }
    return i;
  });
```

```
// Wait for t to complete or 1000ms to elapse
if (t.wait(1000)) {
  // Access the result
  if (t.done && !t.failed) {
    Debug.showDialog("It completed with result="+t.result);
  } else (t.failed) {
    // only access t.error if t.failed == true
    Debug.showDialog("Took too long or errored? "+t.error !== null);
  }
} else {
  // 1 sec elapsed without the task completing
}
```

Run a task and execute a function when the task is done.

```
Task.run(...).then(function(result){
  // do something with the result of the task
});
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.22.2 Wait for *all* tasks to complete

This is used to wait until *all the tasks given as arguments complete* or given milliseconds elapse.

Parameters

- `tasks` - an [array of tasks or javascript functions] to run asynchronously (and then wait for)
- `timeout` [int] denoting the max number of milliseconds to wait for the tasks to complete

Returns

A [bool] indicating wether or not all tasks completed.

Example

```
var t = Task.run(function() { ... });
var tasks = [Task.run(function() { ... }), function() { ... }, t];

// Wait for tasks to complete or 1000ms to elapse
if (Task.waitAll(tasks, 1000)) {
  for (var i=0; i<tasks.length; i++) {
    Debug.showDialog("Task "+i+" resulted in "+tasks[i].result);
  }
  Debug.showDialog("It completed!");
} else {
  Debug.showDialog("Took too long");
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.22.3 Wait for *any* tasks to complete

This is used to wait until *one of the tasks given as arguments completes* or given milliseconds elapse.

Parameters

- `tasks` - an [array of tasks or javascript functions] to run asynchronously (and then wait for one of)
- `timeout` [int] denoting the max number of milliseconds to wait for any of the task to complete

Returns

An [int] denoting the index of the first task to complete or -1 if no tasks complete within given deadline.

Example

```
var t = Task.run(function() { ... });
var tasks = [Task.run(function() { ... }), function() { ... }, t];

// Wait for tasks to complete or 1000ms to elapse
var idx = Task.waitAny(tasks, 1000);
if(idx > 0) {
  Debug.showDialog("We have a winner: "+idx);
} else {
  Debug.showDialog("Took too long. Everybody lost.");
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.22.4 JavaScript Task

A javascript representation of a .NET task. It has 2 methods; `wait(milliseconds)` which can be used to wait for the task to complete or the given milliseconds to elapse, whichever comes first and `then(func)` which can be used to run a function when the task completes.

For an example see the Run method on the `Task` module.

3.23 Guid

This very simple module provides utility functionality for dealing with globally unique identifiers - aka standardized random strings. Use these if you need to generate a unique file name or unique string in general.

3.23.1 Get

Returns a new random standard globally unique identifier

Example

```
var guid = Guid.get();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.24 Tables

The tables module provides functionality to read and write information stored on Kwanza and accessible from the configuration interface (Cuesta). It is meant to provide an easy way to add mapping or other types of tabular data to a flow. The UI for managing tables are shown below.

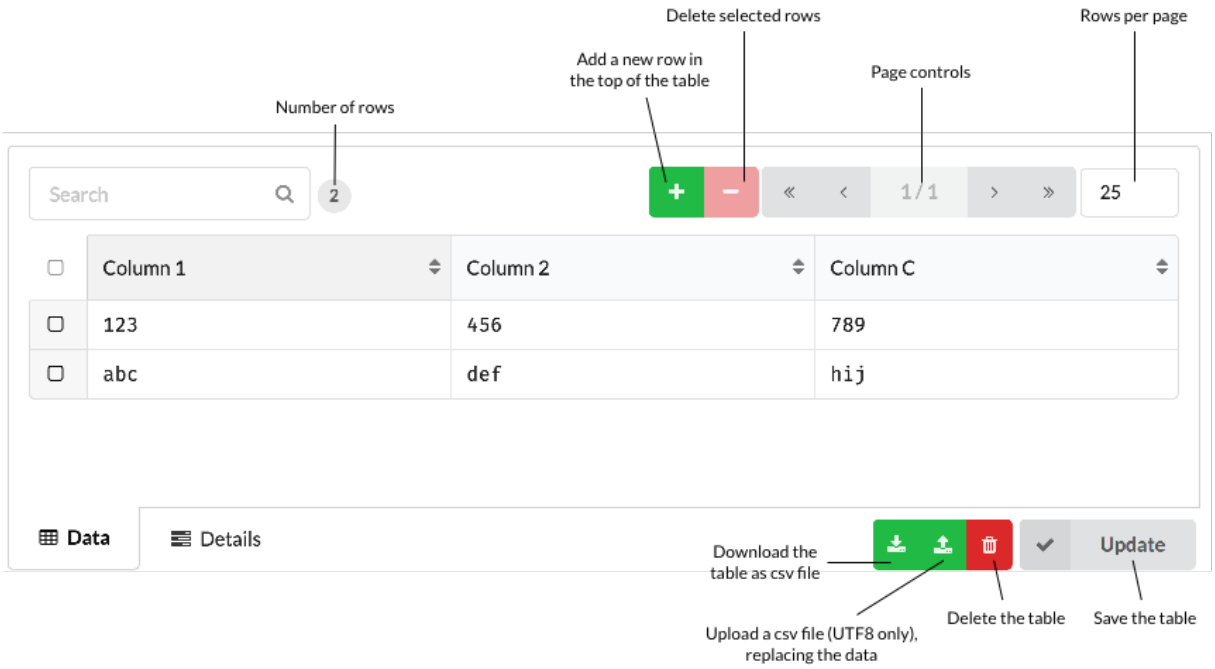

Figure 9

Note that only *UTF8* formatted csv files are supported.

Navigation

Navigating the indvidual cells in the table can be done via the keyboard in a spreadsheet like manner. `alt+<arrow-key>` will move the focus depending on the arrow-key pressed. The video below shows an example of this (the keys pressed are shown in the bottom left corner of the video).

{{< youtube qzjc3XP12qc >}}

Inserting and removing rows

Inserting and deleting rows can also be done via the keyboard. Press `ctrl+n` to insert a row directly below the currently focused row.

{{< youtube TZPgM5eF3Es >}}

Deleting a row is done via the `ctrl+backspace` key. It will remove the currently focused row.

{{< youtube sURkABQTds4 >}}

Inserting and removing columns

This is done similarly to adding and removing rows but the cursor must be placed in the column header. `ctrl+n` adds a new column, while `ctrl+backspace` removes the current.

{{< youtube ssCeIqqX05Y >}}

Shortcuts

Key	Action
`alt+↓`	Focus cell below
`alt+↑`	Focus cell above
`alt→`	Focus cell right
`alt+←`	Focus cell left
`ctrl+shift+a`	Insert new row/column
`ctrl+shift+backspace`	Remove row/column

3.24.1 Read table as a map

The `.map` function will parse a table as a map, meaning that it will use a given column as an index. This is mainly useful if there is a column with unique values to use for the index. The returned structure will be a map with the column headers as keys.

Example

Given the table named `foo`:

A	B
idx1	val1
idx2	val2

And the code:

```
var m = Table.map('foo', 'A');
```

You'll get the following object back:

```
{
  'idx1': { 'A': 'idx1', 'B': 'val1' },
  'idx2': { 'A': 'idx2', 'B': 'val2' }
}
```

Which can then be used in the following manner:

```
var idx2val = m['idx2']['B'];
// or if the column names are valid javascript identifiers
var idx1val = m.idx1.B;
```

Parameters

- `name` - [string] the name of the table to create a map from
- `index` - [string] the name of the column to use as an index

Support

- ChromeDriver
- IEDriver

- JavaDriver
- NativeDriver

3.24.2 Read table as list of rows

The `.rows` function will return the raw table as a javascript array of arrays.

Example

Given the table named `foo` identical to the table from `.map` and the code:

```
var m = Table.rows('foo');
```

You'll get the following object back:

```
{
  rows: [
    ['idx1', 'val1'],
    ['idx2', 'val2'],
  ]
}
```

Which can then be used in the following manner:

```
var idx2val = m.rows[1][1];
```

Parameters

- `name` - [string] the name of the table to create from

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.24.3 Update the contents of a table

The object returned from both `.map` and `.rows` contains a `.save` function which can be used to write data back to a table.

Examples

Update existing entries

Given the table from the previous examples and the code:

```
var m = Table.rows('foo');
m.rows[0][1] = 'newval1';
m.save();
```

Will change the value of the specified cell and update the table. This also works if `.map` is used:

```
var m = Table.map('foo', 'A');
m.idx1.A = 'newval1';
m.save();
```

Add new entries

Adding to a table read by the `rows` approach:

```
var m = Table.rows('foo');
m.rows.push(['idx3', 'val3']);
m.save();
```

This will add a new row with `idx3` and `val3`. When using rows the order of the input elements matter and should match the order of the columns.

The same information can be added when the table is read via the `map` approach as follows:

```
var m = Table.map('foo', 'A');
m.rows['idx3'] = { 'A': 'idx3', 'B': 'val3'};
m.save();
```

Remove entries

Removing a row from a table read by the `rows` approach is done by removing the corresponding array entry:

```
var rowToDelete = 0;
var foo = Table.rows("foo");
foo.rows.splice(rowToDelete, 1); // Delete the row w index 0
foo.save();
```

and the equivalent delete of a entry from a map table:

```
foo = Table.map('foo', 'A');
delete foo["idx1"]; // Delete the entry with key 'idx1'
foo.save();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.24.4 Use the contents of a Table as options for a typeahead

This is achieved by calling the `selectFrom` method on the structure created by the `.map` function. The `selectFrom` function takes either a format string or an object with options to generate the content for a typeahead.

```
var m = Table.map(...);
m.selectFrom('{{someColumn}} some text {{someOtherColumn}}');
```

Using a format string (above) and an object with options (below).

```
var m = Table.map(...);
m.selectFrom({
  format: '{{someColumn}} some text {{someOtherColumn}}',
  minInputLength: 3,
  filterMode: 'contains'
});
```

3.25 Env

The env module provides some contextual information for flows.

3.25.1 Username

Get the username for the current user.

Example

```
var u = Env.userName;
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.25.2 Name of machine

Get the name of the machine.

Example

```
var m = Env.machineName;
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.25.3 Domain

Get the domain for the current user.

Example

```
var u = Env.userDomain;
```

Support

- ChromeDriver
- IEDriver

- JavaDriver
- NativeDriver

3.25.4 Primary screen

Get information about the primary screen of the local machine.

Example

```
var s = Env.primaryScreen;
```

s will now be an object like so:

```
// s
{
  width: 1024,
  height: 768,
  primary: true
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.25.5 Screens

Get information about all the screens attached to the local machine.

Example

```
var screens = Env.screens;
```

screens will now be an array of screen objects, like so:

```
// screens
[
  {
```

```
    width: 1024,
    height: 768,
    primary: true
  },{
    width: 1280,
    height: 1024,
    primary: false
  }
]
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.26 Crypto

The `Crypto` module can be used to encrypt/decrypt secrets and other sensitive information. It can be used together with e.g. the Table module to keep passwords or similar items for use in flows but not visible for other than the intended users.

3.26.1 Encrypt

Make an encrypted string from the given input and access-scope. Access-scope can be:

- `Crypto.forUser` to allow only the current logged in user to decrypt the information. Decryption may happen on a different machine or using a different application than Manatee, but only the current logged in Windows user will be able to do the decrypt.
- `Crypto.forMachine` to only allow users on the current machine to decrypt. Again decrypting is not limited to Manatee - any program on the local machine will be able to decrypt.
- a `string` password to only allow users who know the supplied password to decrypt the message (min 12 characters).
- `null` or `undefined` or no argument given to make the encrypted string decryptable only by the Manatee application across all users and all machines.

Examples

```
// for the current user
var encryptedString = Crypto.encrypt(”my secret”, Crypto.forUser);
// for the current machine
encryptedString = Crypto.encrypt(”my secret”, Crypto.forMachine);
// for users with the correct password
encryptedString = Crypto.encrypt(”my secret”, ”password12345678”);
// for Manatee eyes only
encryptedString = Crypto.encrypt(”my secret”);
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.26.2 Decrypt

Take an ecnrypted string and decrypt. Supply it with the same access-scope used when the string was `encrypted`.

Examples

```
// for the current user
var originalString = Crypto.decrypt(encryptedString, Crypto.forUser);
// for the current machine
originalString = Crypto.decrypt(encryptedString, Crypto.forMachine);
// for users with the correct password
originalString = Crypto.decrypt(encryptedString, ”password12345678”);
// for Manatee eyes only
originalString = Crypto.decrypt(encryptedString);
```

Support

- ChromeDriver
- IEDriver

- JavaDriver
- NativeDriver

3.27 Clipboard

The `Clipboard` module lets you interact with the windows clipboard for programmatic copy and paste purposes.

3.27.1 Get

Get the current string value of the system clipboard

Examples

```
var copyValue = Clipboard.get();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.27.2 Set

Sets the current value of the system clipboard to a string

Examples

```
Clipboard.set('This text can be pasted by the user or by the flow');
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.27.3 Clear

Clears the current value of the system clipboard. Useful if the flow needs to temporarily put sensitive data in the clipboard.

Examples

```
try {
  Clipboard.set('This is not for everyone to see');
  Clipboard.paste();
} finally {
  Clipboard.clear();
}
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.27.4 Copy

Carries out a standard copy (Ctrl + c) operation

Examples

```
Clipboard.copy();
var copiedValue = Clipboard.get();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.27.5 Cut

Carries out a standard cut (Ctrl + x) operation

Examples

```
Clipboard.cut();
var cutValue = Clipboard.get();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.27.6 Paste

Carries out a standard paste (Ctrl + v) operation

Examples

```
Clipboard.set('some text to paste');
Clipboard.paste();
```

Support

- ChromeDriver
- IEDriver
- JavaDriver
- NativeDriver

3.28 Desktop

The `Desktop` module is a Windows 10 only can be used for manipulating virtual desktops and for moving application windows between desktops.

3.28.1 All

Get a list containing the ids of all virtual desktops.

Example

```
var desktops = Desktop.all();
for (var i=0; i<desktops.length; i++) {
  Debug.showDialog("Desktop "+desktops[i]);
}
```

3.28.2 Current

Get the id of the current/active virtual desktop.

Example

```
var current = Desktop.current();
```

3.28.3 Add a new desktop

Will create a new virtual desktop and return its id.

Example

```
var d = Desktop.add();
```

3.28.4 Moving windows between virtual desktops

The `moveWindow`, `moveWindowRight` and `moveWindowLeft` methods can be used to move a window between virtual desktops.

Example

```
// Move the window of the current application to an identified desktop (123)
var success = Desktop.moveWindow("123");
// Move window to the desktop to the right of the current desktop
```

```
var idOfDesktopMovedTo = Desktop.moveWindowRight();
// ... same for left
idOfDesktopMovedTo = Desktop.moveWindowLeft();
```

3.28.5 Switching between desktops

Use the `switchTo`, `switchRight` and `switchLeft` methods to switch between virtual desktops.

Example

```
// Switch to an identified desktop
var idOfDesktopSwitchedTo = Desktop.switchTo("123");
// Switch to a desktop to the left/right of the current
vidOfDesktopSwitchedTo = Desktop.switchLeft();
idOfDesktopSwitchedTo = Desktop.switchRight();
```

3.29 Html parsing and querying

The `Html` module can be used to parse and query html formatted files and remote pages.

3.29.1 Loading data

The methods `load` and `loadFrom` can be used to load and parse a html document. They both return a HtmlDoc object which can be used for querying/extracting information.

Example

```
// Load html from a string
var doc = Html.load("<html><body>Hello, world!</body></html>");
// Load html from an url
doc = Html.loadFrom("http://sirenia.eu");
```

3.30 HtmlDoc

The `HtmlDoc` object return from `Html.load` and `.loadFrom` has two primary methods for querying and extracting information from the html document it represents - the first is via an XPath query and the second is to convert the html to json.

3.30.1 XPath

The `xpath` method can be used to query the `HtmlDoc` with a given XPath query.

Example

```
var d = Html.load("<html><body>Hello</body>");
var body = d.xpath("//body");
Debug.showDialog(body.innerText); // shows "Hello"
```

3.30.2 Converting to json

Converting the html to json is done with the `.json()` method. Each node in the resulting tree of objects has the following properties:

- `attrs` an object containing the attributes of the html node
- `children` is an array of child json nodes
- `innerText` is a textual representation of the contents of the node
- `tagName` is the name of the original html node

It also has an `xpath` method which can be used to query the subtree of the json node.

Example

```
var d = Html.load("<html><body>Hello</body>");
var json = d.json();
Debug.showDialog(json.tagName);
```

3.31 Tracer

The `Tracer` module enables remote (via flows) controlling of the tracer functionality. To enable the UI of the Tracer open the settings for Manatee and search for "Tracer". When the UI is enabled it will

show a small window (notification-style) in which output from the current flow is shown. Output includes which API functions are called, which fields are interacted with etc. The window also holds buttons to pause, resume and step forward in the flow as well as a button to pause and bring up the debug.ger window. By using this module in a flow you can control much of the same functionality.

Note that care should be taken using the Tracer functionality in production flows. It is primarily a developer tool.

3.31.1 Delay

The `delay` methods controls how fast your flow is running. By setting a >0 delay you can slow down your flow.

Example

```
// Delay each flow ”step” 1s
Tracer.delay(1000);
```

3.31.2 Pause

Allows you to pause the flow. Resuming can only be done in the flow-tracer UI.

Example

```
Tracer.pause();
```

3.31.3 Resume

Resume a paused flow. Be aware that you can only resume a flow using this method if it is running asynchronously.

Example

```
Tracer.resume();
```

3.31.4 Message

Show a message in the Tracer UI.

```
Tracer.msg("Hello from a flow");
```