



[www.rexcontrols.com/rex](http://www.rexcontrols.com/rex)

---

# RexHMI – A Web-based HMI for REX

## User guide

REX Controls s.r.o.

Version 2.50.1

2016-11-07

Plzeň (Pilsen), Czech Republic

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                                | <b>2</b>  |
| <b>2</b> | <b>WebWatch</b>                                    | <b>5</b>  |
| 2.1      | Automatic Generation of HMI from RexDraw . . . . . | 5         |
| 2.2      | Advanced Usage . . . . .                           | 6         |
| <b>3</b> | <b>RexHMI Designer</b>                             | <b>8</b>  |
| 3.1      | A Graphical Designer of Web HMI . . . . .          | 8         |
| <b>4</b> | <b>WebBuDi</b>                                     | <b>11</b> |
| 4.1      | Simple Buttons and Displays on the Web . . . . .   | 11        |
| 4.2      | Available Rows and Components . . . . .            | 14        |
| <b>5</b> | <b>REX.HMI library</b>                             | <b>21</b> |
| 5.1      | How to Use the Library . . . . .                   | 21        |
| 5.2      | Reference Guide for REX.HMI . . . . .              | 22        |
| 5.3      | Reference Guide for REX.HMI.Graph . . . . .        | 28        |
|          | <b>Bibliography</b>                                | <b>31</b> |

# Chapter 1

## Introduction

RexHMI covers all tools and libraries necessary for creating human-machine interfaces (visualizations) for the REX control system. There are three different types of visualization the **WebWatch**, the **WebBuDi** and the one created by **RexHMI Designer**.

- **WebWatch**(Chapter 2) is an auto-generated HMI from the **RexDraw** development tool during project compilation. It has similar look, attributes and functions as the online mode of the **RexDraw** development tool. The **WebWatch** is a perfect tool for instant creation of HMI that is suitable for system developers or integrators. It provides a graphical interaction with almost all signals in the control algorithm.
- **RexHMI Designer**(Chapter 3) creates a standard SVG file with the *RexHMI* extensions. The **RexHMI Designer** is a great tool for creating graphical HMI that is suitable for operators and other end users.
- **WebBuDi**(Chapter 4), which is an acronym for Web Buttons and Displays, is a simple JavaScript file with several declarative blocks that describe data points which the HMI is connected to and assemble a table in which all the data is presented. It provides a textual interaction with selected signals and is suitable for system developers and integrators or may serve as a fall-back mode HMI for non-standard situations.

All the tools result to the HTML5 web page served from the internal REX web server. The HMI can be accessed using desktop, tablets and also mobile devices. Recommended web browsers are **Google Chrome** and **Mozilla Firefox**.

Usually the HMI is downloaded to the target device using *HMI* block in the *exec.mdl* file. In the project directory create *exec.mdl* file (or use predefined templates from the Start-up wizard). Add the *HMI* block to the executive, set the *IncludeHMI* parameter and then enable *Web Watch* visualization (*GenerateWeb Watch* parameter) or create your custom one. Once configured the visualization is downloaded using *Compile and Download* function in the **RexDraw** design tool to the target device. The HMI is accessible from the internal server of the target device for example at <http://127.0.0.1:8008/hmi>.

The HMI uses HTML, CSS3, JavaScript and WebSockets.

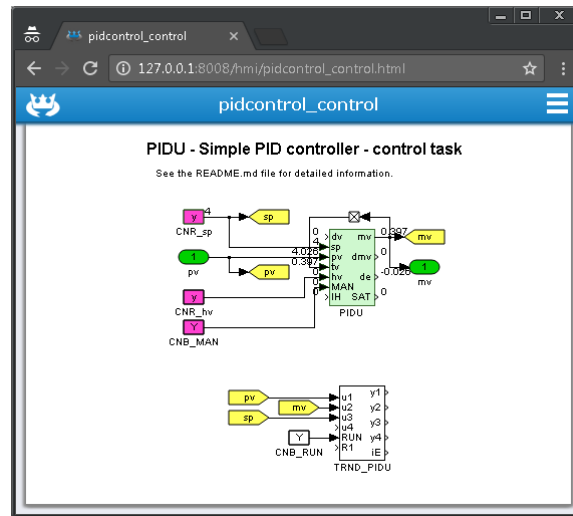


Figure 1.1: The example of *WebWatch* visualization

For the special cases user can create its own HMI based on the JavaScript and custom HTML or SVG components. For that purpose the *REX.HMI* function library is available. Chapter 5 describes all functions with parameters.

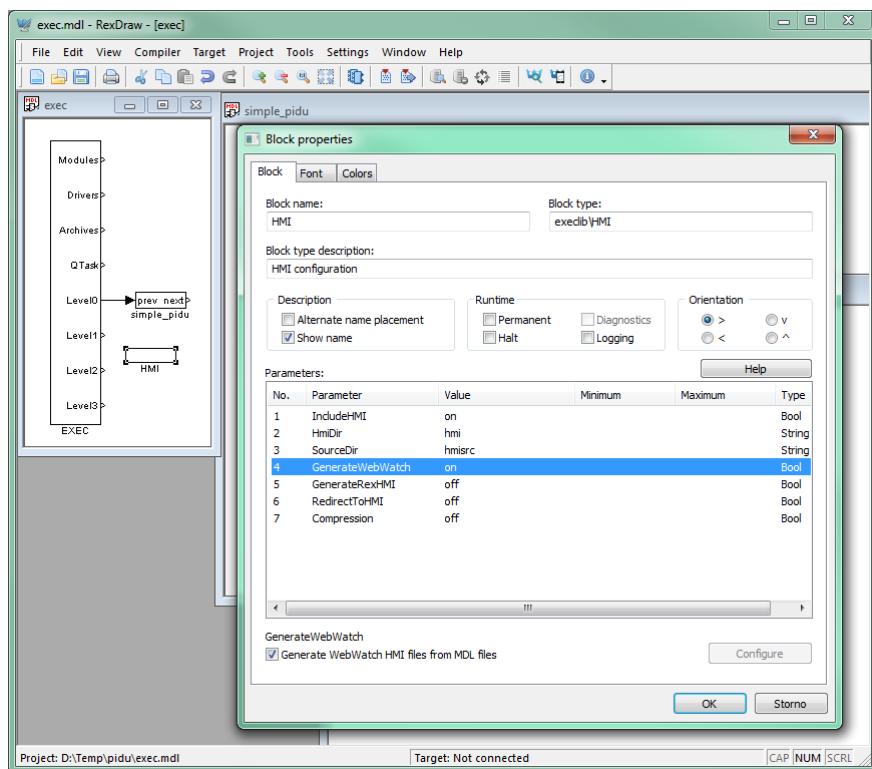


Figure 1.2: Configuration of the *HMI* block in RexDraw

## Chapter 2

# WebWatch

### 2.1 Automatic Generation of HMI from RexDraw

WebWatch is automatically generated HMI based on the project structure. It is similar to the *Online Monitoring* tools in RexDraw. The whole scheme is generated to web page. User can monitor all signals from selected blocks, change block parameters and read the data from TRND blocks. The WebWatch is generated automatically using *HMI* block in project executive file.

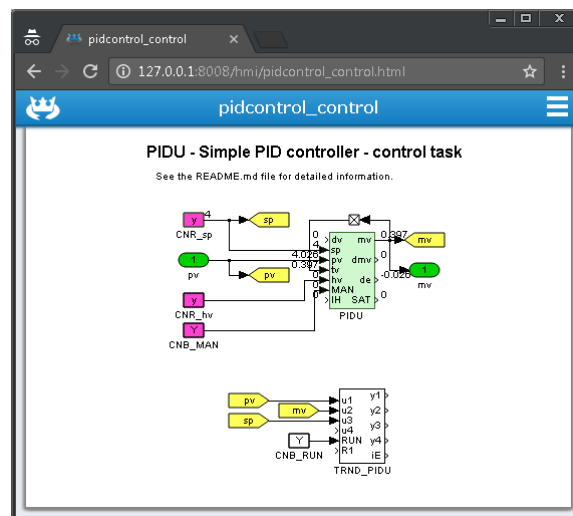


Figure 2.1: The example of WebWatch visualization

1. Insert HMI block to the *exec.mdl* file of your project
2. Check `GenerateWebWatch` and `IncludeHMI` to enable WebWatch generation
3. Run *Compile and Download*

4. Open the web browser on your target device eg. <http://127.0.0.1:8008/hmi>.
5. Use *left mouse click (touch)* for changing the block parameters and *right mouse click (long touch)* for enabling the block monitoring.

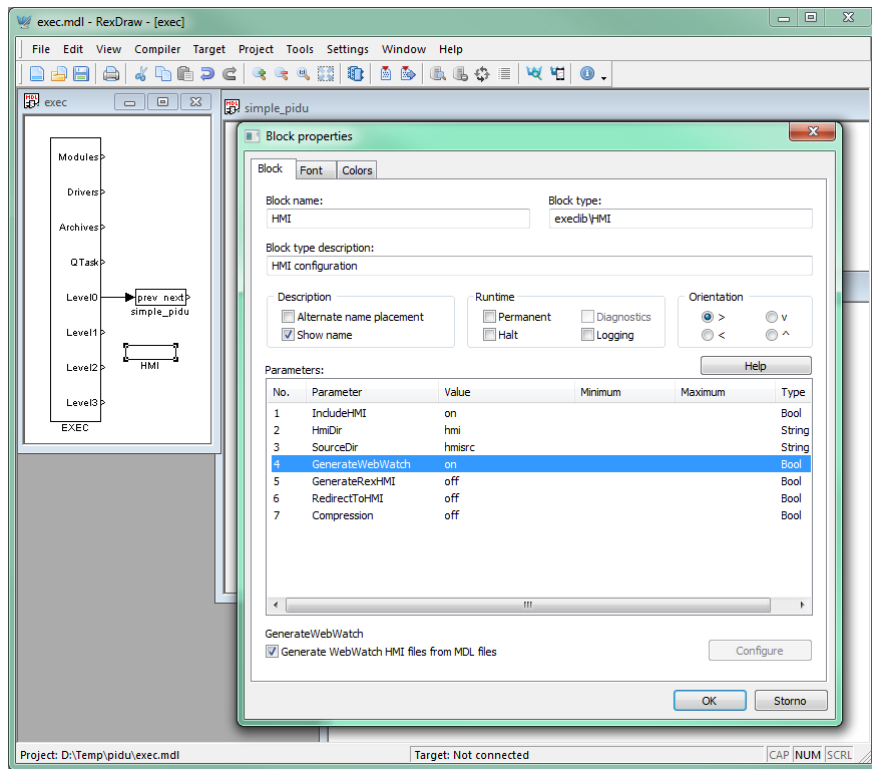


Figure 2.2: Check GenerateWebWatch to create WebWatchHMI.

## 2.2 Advanced Usage

The auto-generated scheme can be adjusted manually. <sup>1</sup>

Edit the `REX.HMI.init = function(){} function` in the selected `*.html` file. There are only few functions for the WebWatch HMI, but you can use all the functions described in chapter 5.

**REX.WebWatch.enableMonitoring(blockIDs)** – Enable monitoring of the selected blocks from the beginning (after web page is loaded).

<sup>1</sup>Note: If the *GenerateWebWatch* in HMI block is enabled the HMI is generate whenever the scheme is compiled. So for manual adjustment disable the *GenerateWebWatch* option in *HMI* block.

| Param    | Type               | Description  |
|----------|--------------------|--|
| blockIDs | Array<br>.<String> | List of all blocks, described by connection string<br>(eg.["task.block1","task.block2"]) |

### Example

```

1 // Enable monitoring of selected blocks
2 REX.HMI.init = function(){
3     REX.WebWatch.enableMonitoring(["pidcontrol_control.CNR_sp",
4     pidcontrol_control.PIDU"]);
5 }

```

**REX.WebWatch.disableHint()** – Disable hint after page is loaded

**REX.WebWatch.showHint()** – Show hint



## Chapter 3

# RexHMI Designer

### 3.1 A Graphical Designer of Web HMI

RexHMI Designer<sup>1</sup> is tool for designing the custom visualizations using predefined components. The whole HMI is stored in the SVG (Scalable Vector Graphics) file with REX specific extensions.

When the visualization is ready the HTML page with all necessary libraries is generated. Such a webpage can be easily downloaded to the target device from `./hmi` directory using *Compile and Download* function of the RexDraw development tool.

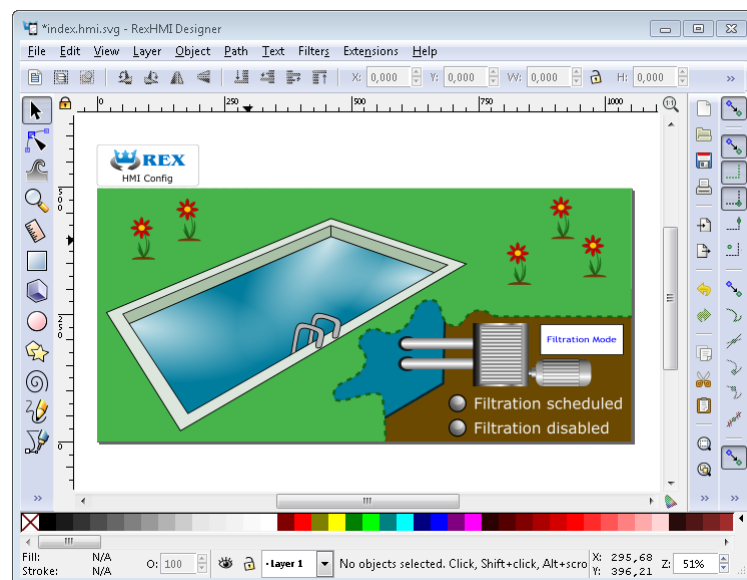


Figure 3.1: The example of RexHMI Designer visualization

<sup>1</sup>The RexHMI Designer is build on the well-known SVG editor Inkscape<sup>TM</sup> <https://inkscape.org/en/>.

Each HMI scheme composes from several components which are connected to the signals from the control system. These components are organized in libraries. The example HMI on fig. 3.1 contain one *Button*, two *LEDs* and several *GeneralComponents* for pool and flower animations.

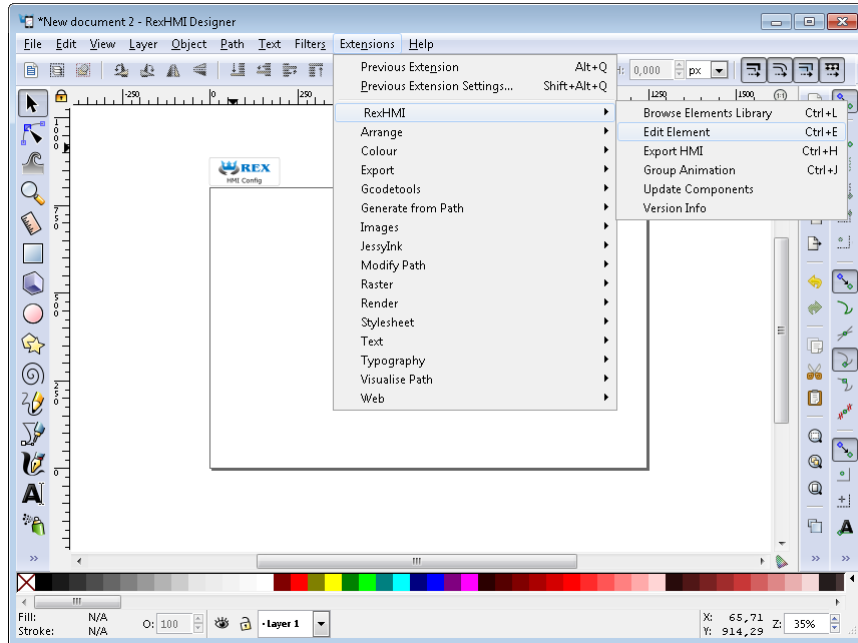


Figure 3.2: The list of RexHMI extensions

The HMI components are controlled via extensions (on fig. 3.2) which edit the RexHMI specific parameters and allow the export of the final HMI.

- **Browse Elements Library** (Ctrl + L) – Open the folder with all available components / elements. Each component is stored as a SVG file which can be drag&dropped to the current HMI
- **Edit element** (Ctrl + E) – Open edit window for the selected component or for the whole project if nothing is selected.
- **Export HMI** (Ctrl + H) – Exports the content of the SVG file to the HTML5 webpage with all necessary libraries. This extension is configured via parameters of the whole project.
- **Group Animation** (Ctrl + J) – Enable animation of the transformations (translation, rotation, scale), opacity or color of the selected object (SVG group) based on the values from the control system.
- **Update Components** – Extension for updating the schema to new version

- **Version Info** – Show current version of the RexHMI Designer and RexHMI tools.

Follow the tutorials on [www.rexcontrol.com](http://www.rexcontrol.com) to know how to use the extensions for creating your custom HMIs.

## Chapter 4

# WebBuDi

### 4.1 Simple Buttons and Displays on the Web

WebBuDi is an acronym for Web Buttons and Displays, is a simple JavaScript file with several declarative blocks that describe data points which the HMI is connected to and assemble a table in which all the data is presented. It provides a textual interaction with selected signals and is suitable for system developers and integrators or may serve as a fall-back mode HMI for non-standard situations.

WebBuDi is composed from several **rows** (a graphical components with pre-defined function and look) connected to a single item in the control system (specified by an **alias** or a **cstring** property). There are different rows according to the type they are changing (for boolean, numbers, dates, etc.). All **rows** are organized in *sections* (colored blocks which can have a heading). The sections are then organized in several columns.

The configuration is done using JavaScript objects. See [4.1](#) for more details.

#### Example

```
1 REX.HMI.init = function(){
2 // Simple PID controller example
3
4 // Optional - Add items first
5 REX.WebBuDi.addItem([
6 {alias: 'PID_MAN', cstring: 'pidcontrol_control.CNB_MAN:YCN', write: true
7   }
8 ]);
9
10 // Add WebBuDi section
11 REX.WebBuDi.addSection({
12   column: 1,
13   title: 'User controls',
14   rows: [
15     {alias: 'PID_MAN', desc: 'Controller mode', type: 'DW', label_false: 'AUT',
16       label_true: 'MAN'},
17     {type: 'ES'},
18     {alias: 'SP_AW', cstring: 'pidcontrol_control.CNR_sp:ycn', desc: 'Setpoint',
19       type: 'AW'}
20   ]
21 }
```

```

18 });
19
20 // Show graph from TRND block
21 REX.HMI.Graph.addSignal({cstring: 'pidcontrol_control.TRND_PIDU', labels:
    ['Process value', 'Manipulated variable', 'Setpoint']}));
22
23 // Set different target address
24 // REX.HMI.setTargetUrl('ws://127.0.0.1:8008/rex');
25
26 // Set refresh rate (Default: 500 ms)
27 REX.HMI.setRefreshRate(100);
28
29 // Change title of the page
30 REX.HMI.setTitle('Simple PID controller');
31 }

```

**.WebBuDi** : object

- **.addSection(opt)** => REX.HMI.WebBuDi
- **.add()** => REX.HMI.WebBuDi
- **.addItems(items)** => Array.<REX.WS.Item>

**REX.WebBuDi.addSection(opt)** => **REX.HMI.WebBuDi** The addSection adds new rows / HMI components to the web page.

The section contains all components defined in **rows** array. It can have **title** shown in header. The whole section can be controlled via **disable\_by** and **hide\_by** item. Finally the section is placed to the **column** (index based).

The function calls can be chained or called via alias **REX.WebBuDi.add()**.

**Kind:** static method of **WebBuDi**

| Param                  | Type              | Default | Description   |
|------------------------|-------------------|---------|---|
| opt                    |                   |         | The main configuration object   |
| opt.rows               | Array.<RowOption> |         | Definition of all HMI components / rows. See the list for more details  |
| [opt.title]            | String            |         | Title of the section shown in the header.   |
| [opt.column]           | number            | 1       | Index of the column (starts from 1).  |
| [opt.background_color] | String            |         | Custom background color of the section.   |
| [opt.text_color]       | String            |         | Custom text color of the section header   |
| [opt.disable_by]       | String   Object   |         | If defined by 'alias' or object {alias:" <b>XXX</b> ", cstring:" <b>XXX</b> ", reverse_meaning: <b>false</b> } the state of the component changes (enabled / disabled). |
| [opt.hide_by]          | String   Object   |         | If defined by 'alias' or object {alias:" <b>XXX</b> ", cstring:" <b>XXX</b> ", reverse_meaning: <b>false</b> } the visibility of the row changes.                       |

| Param             | Type   | Default   | Description  |
|-------------------|--------|-----------|--|
| [opt.customDivID] | String | "content" | The ID of the element where all the columns / sections will be appended. |

### Example

```

1 // Simple HMI for PIDU_Simple_PID_Controller
2
3 REX.WebBuDi.addSection({
4   column: 1,
5   title: 'Controls',
6   rows: [
7     // Digital write
8     {alias: 'PID_MAN', desc:'Controller mode', type: 'DW', label_false: 'AUT',
9       label_true: 'MAN'},
9     // Analog write
10    {alias: 'SP', desc:'Setpoint', cstring: 'pidcontrol_control.CNR_sp:ycn',
11      type: 'AW'},
12    {type: 'ES'}, // Empty space
13  ],
14  hide_by:"",
15  disable_by:""
16 });

```

**REX.WebBuDi.add()** => **REX.HMI.WebBuDi** Shortcut for [REX.WebBudi.addSection](#) function.

**Kind:** static method of [WebBuDi](#)

**REX.WebBuDi.addItems(items)** => **Array.<REX.WS.Item>** Add several items at once. This is useful way how to define aliases and use them in various rows

**Kind:** static method of [WebBuDi](#)

| Param         | Type               | Description  |
|---------------|--------------------|--|
| items         | Array.<br><Object> | Array of items to register. Shortcut for <code>REX.HMI.addItems()</code> . |
| items.alias   | String             |  |
| items.cstring | String             |  |
| items.write   | boolean            | Set true if item is writable   |

### Example

```

1 REX.WebBuDi.addItems([
2   {alias: 'PID_MAN', cstring: 'pidcontrol_control.CNB_MAN:YCN', write: true},
3   {alias: 'SP_AW', cstring: 'pidcontrol_control.CNR_sp:ycn', write: true},

```

```

4 {alias: 'HV', cstring: 'pidcontrol_control.PIDU:hv'}
5  });

```

## 4.2 Available Rows and Components

- AnalogLookupTable ('ALT')
- Analog Write ('AW')
- Digital Read ('DR')
- Digital Write ('DW')
- Empty Space ('ES')
- Link Button ('LINK')
- Manual Pulse ('MP')
- Push Button ('PB')

**General row options** - Every row is configured with common and row-specific properties. The following object represents the common part.

### Properties

| Name         | Type            | Description   |
|--------------|-----------------|---|
| type         | String          | The type of the component / row (eg. "AR")  |
| alias        | alias           | UNIQUE identification of the item from control system (eg. 'SP'). Must NOT contain spaces or diacritics   |
| desc         | String          | Name of the component shown in the description (eg. "Set point"). If the <code>alias</code> is not defined the <code>desc</code> is used instead. The <code>desc</code> is converted to lowercase, spaces are substitute with underscore and all non-ascii letters omitted or replaced with ascii equivalent. |
| cstring      | String          | Connection string which contain whole path to the target device (eg. "task.CNR:ycn")  |
| disable_by   | String   Object | If defined by 'alias' or object {alias:"XXX", cstring:"XXX", reverse_meaning:false} the state of the component changes (enabled / disabled).  |
| hide_by      | String   Object | If defined by 'alias' or object {alias:"XXX", cstring:"XXX", reverse_meaning:false} the visibility of the row changes.  |
| refresh_from | String          | If defined by 'alias' or object {alias:"XXX", cstring:"XXX"} the value for the item is read from different location than written. It is applicable on for WRITE components  |

```

1 // All properties
2 {alias:"SP", desc:"Set point", cstring:"task.CNR:ycn", disable_by:null,
  hide_by:null refresh_from:null}

```

**AnalogLookupTable ('ALT')** - Select with list of options. Used for user define enums.

| Param          | Type      | Default | Description   |
|----------------|-----------|---------|---|
| opt            | RowOption |         | General configuration for row extended with following properties  |
| [opt.show_key] | boolean   | false   | Append keys to the list of options  |
| opt.values     | Object    | {}      | List of all values which can be selected. It is a Object with key-value pairs (e.g. {"1":"Options 1", "2":"Options 2"}) * |

### Example

```
1 {type:'ALT', values:{"1":"Options 1", "2":"Options 2"}, show_key: false}
```

**Analog Read ('AR')** - Periodical reading of selected value (date,time,datetime,text,number).

date, time, datetime - Show date calculated from seconds from REX Epoch timestamp.

text - Show the value without any transformation (suitable for string values).

number - Show number transformed by scale, offset and round to number of decimals.

| Param          | Type      | Default | Description  |
|----------------|-----------|---------|--|
| opt            | RowOption |         | General configuration for row extended with following properties   |
| [opt.format]   | boolean   | number  | One of the following date,time,datetime,text,number  |
| [opt.scale]    | number    | 1       | Scale factor   |
| [opt.offset]   | number    | 0       | Offset for the displayed value   |
| [opt.decimals] | number    | 4       | Number of decimals   |
| [opt.convert]  | function  |         | If defined, the format='number' value is transformed using convert function eg. convert=function(val){return val+1;} |

### Example

```
1 // Show number rounded to 4 decimal places
2 {type:'AR'}
3
4 // Show date
5 {type:'AR', format:'date'}
6
7 // Show string values
8 {type:'AR', format:'text'}
9
10 // Convert radians to degrees
11 {type:'AR', format:'number', scale:(Math.PI/180), offset=0, decimals=0}
```



**Analog Write ('AW')** - Set `date,time,datetime,text` or `number` to the control system.  
`date, time, datetime` - R/W date calculated from seconds from REX Epoch timestamp  
`text` - Write value without any transformation (suitable for string values)  
`number` - R/W number transformed by `scale, offset` and rounded to number of `decimals`.

| Param                          | Type      | Default | Description   |
|--------------------------------|-----------|---------|---|
| <code>opt</code>               | RowOption |         | General configuration for row extended with following properties  |
| <code>[opt.format]</code>      | boolean   | number  | One of the following <code>date,time,datetime,text,number</code>  |
| <code>[opt.scale]</code>       | number    | 1       | Scale factor  |
| <code>[opt.offset]</code>      | number    | 0       | Offset for the displayed value  |
| <code>[opt.min]</code>         | number    |         | Limit the minimum value   |
| <code>[opt.max]</code>         | number    |         | Limit the maximum value   |
| <code>[opt.decimals]</code>    | number    | 4       | Number of decimals  |
| <code>[opt.set_on_blur]</code> | boolean   | false   | If set the value is written when the input is blurred. Use <i>ESC</i> to cancel changes   |
| <code>[opt.convert]</code>     | function  |         | If defined, the <code>format='number'</code> value is transformed using convert function eg. <code>convert=function(val){return val+1;}</code>            |
| <code>[opt.convertW]</code>    | function  |         | If defined, the <code>format='number'</code> value is transformed before write by convert function eg. <code>convertW=function(val){return val+1;}</code> |

### Example

```

1 // Change number rounded to 4 decimal places
2 {type: 'AW'}
3
4 // Set date
5 {type: 'AW', format: 'date'}
6
7 // Set string values
8 {type: 'AW', format: 'text'}
9
10 // Display degrees, read and write as radians
11 {type: 'AW', format: 'number', scale: (Math.PI/180), offset=0, decimals=0}
12
13 // Default options
14 {type: 'AW', format: 'number',
15 scale:1, offset:0, min: -Number.MAX_VALUE, max: Number.MAX_VALUE,
16 set_on_blur:false, convert:null, convertW:null}
```

**Digital Read ('DR')** - Periodical reading of boolean value.

| Param                 | Type      | Default | Description  |
|-----------------------|-----------|---------|--|
| opt                   | RowOption |         | General configuration for row extended with following properties |
| [opt.label_false]     | String    | "OFF"   | Label for the false / off / disable value.                       |
| [opt.label_true]      | String    | "ON"    | Label for the true / on / enable value.                          |
| [opt.reverse_meaning] | boolean   | false   | If set the '0' (zero) means enables / ON and '1' disabled / OFF  |
| [opt.color_false]     | String    | ""      | Change color of FALSE button when active                         |
| [opt.color_true]      | String    | ""      | Change color of TRUE button when active                          |

### Example

```

1 {alias: 'DR', desc: 'Controller mode', type: 'DR', label_false: 'AUT',
  label_true: 'MAN'}
2
3 // All options
4 {type: 'DR', label_false: 'OFF(0)', label_true: 'ON(1)', reverse_meaning:
  false, color_false: '', color_true: ''}

```

### Digital Write ('DW') - Set boolean value

| Param                 | Type      | Default | Description  |
|-----------------------|-----------|---------|--|
| opt                   | RowOption |         | General configuration for row extended with following properties |
| [opt.label_false]     | String    | "OFF"   | Label for the false / off / disable value.                       |
| [opt.label_true]      | String    | "ON"    | Label for the true / on / enable value.                          |
| [opt.reverse_meaning] | boolean   | false   | If set the '0' (zero) means enables / ON and '1' disabled / OFF  |
| [opt.color_false]     | String    | ""      | Change color of FALSE button when active                         |
| [opt.color_true]      | String    | ""      | Change color of TRUE button when active                          |
| [opt.flip]            | boolean   | false   | If set the position of the TRUE/FALSE buttons is flipped.        |

### Example

```

1 {alias: 'DW', desc: 'Controller mode', type: 'DW', label_false: 'AUT',
  label_true: 'MAN'}
2
3 // All options
4 {type: 'DW', label_false: 'OFF(0)', label_true: 'ON(1)', reverse_meaning:
  false, color_false: '', color_true: '', flip: false}

```

### Empty Space ('ES') - Creates empty row to fill gaps

**Link Button ('LINK')** - Create link to different page

| Param            | Type   | Default | Description                     |
|------------------|--------|---------|---------------------------------|
| opt              | Object |         | LINK configuration object       |
| [opt.target_url] | String | ""      | URL to which the link leads to. |
| [opt.desc]       | String | ""      | Description of the link         |
| [opt.label]      | String | ""      | Button label                    |

### Example

```
1 {alias: 'LINK', target_url: '/hmi/index.html', desc: 'Go to index page ...', label: 'To index'}
```

### Manual Pulse ('MP') - Manual Pulse controller (for MP block)

| Param | Type      | Description                         |
|-------|-----------|-------------------------------------|
| opt   | RowOption | General configuration for row apply |

### Example

```
1 {alias: 'MP_RUN', cstring: "task.MP_RUN:BSTATE" type: 'MP'}
```

### Push Button ('PB') - One button for setting different values on press and release.

| Param                 | Type            | Default | Description  |
|-----------------------|-----------------|---------|--|
| opt                   | RowOption       |         | General configuration for row extended with following properties             |
| [opt.label_false]     | String          | "OFF"   | Label for the false / off / disable value.                                   |
| [opt.label_true]      | String          | "ON"    | Label for the true / on / enable value.                                      |
| [opt.reverse_meaning] | boolean         | false   | If set the '0' (zero) means enables / ON and '1' disabled / OFF              |
| [opt.color_false]     | String          | ""      | Change color of FALSE button when active                                     |
| [opt.color_true]      | String          | ""      | Change color of TRUE button when active                                      |
| [opt.value_release]   | number   String | 0       | Set the value which should be set on release (reverse_meaning do not apply). |
| [opt.value_press]     | number   String | 1       | Set the value which should be set on press (reverse_meaning do not apply).   |

### Example

```
1 // Write 'true' on press and 'false' on release
2 {type: 'PB'}
3
4 // Write float value
5 {type: 'PB', value_release: 0, value_press: 0.1}
```

```
6
7 // All options
8 {type: 'PB', label_false: 'OFF(0)', label_true: 'ON(1)', reverse_meaning:
  false,
9 color_false:'', color_true:'', flip:false, value_release: 0, value_press:
  1}}s
```

## Chapter 5

# REX.HMI library

### 5.1 How to Use the Library

The user can program the HMI over REX.HMI interface. It contain several public methods for reading and writing items from REX targets. To create and build your custom HMI based on the RexHMI library follow these steps:

1. Create project directory with *exec.mdl*. Add the *HMI* block to the executive (see the [1] for more details)
2. Create *\*.hmi.js* file in *./hmisrc* directory with `REX.HMI.init = function(){} function` inside.
3. Start writing your script

When the script is ready, enable generation of RexHMI using *GenerateRexHMI* parameter in the *HMI* block of the executive. Then run *Compile and Download* function in the *RexDraw*. The content of the *\*.hmi.js* file is inserted to the RexHMI template with all the necessary scripts and libraries, also all other content of the *hmisrc* directory is copied to the target *./hmi* folder. The HMI is then downloaded to the target and available from the internal webserver.

The `REX.HMI` interface is described in following sections. Each function contain short example with common parameters.

```
1 // Common usage
2 // Append this function to the HTML document or use predefined
  placeholder in the template
3 REX.HMI.init = function(){
4 // Change some basic settings
5
6 }
```

## 5.2 Reference Guide for REX.HMI

The main entry-point for all RexHMI visualization. This class exposes all methods necessary for reading and writing variables in control scheme over WebSockets.

**Emits:** event:time, event:online, event:offline

### Properties

| Name                      | Type    | Description   |
|---------------------------|---------|---|
| REX.HMI.kioskMode         | boolean | Set true to enable kioskMode of the HMI   |
| REX.HMI.disableAutoReload | boolean | Set to true if autoreload of the web page should be disabled Autoreload is call when the REX executive or HMI changes. This can be useful if general signals are read to Graph and one do not want to loose history |

### List of all available functions

- `.init()`
- `.connect() =>Promise`
- `.disconnect()`
- `.getTarget(url) =>WSTarget`
- `.setTargetUrl(url, force) =>Promise`
- `.setRefreshRate(period)`
- `.addItem(opt) =>Item`
- `.addItems(items) =>Array.<Item>`
- `.removeItem(alias) =>Promise`
- `.get(alias) =>Item`
- `.$i(alias) =>Item`
- `.addGroup(g)`
- `.removeGroup(g)`
- `.addTrend(t)`
- `.removeTrend(t)`
- `.writeValues(aliases, values) =>Promise`

- `.setTitle(title)`
- `.setHeaderTitle(title)`
- `.showHeartBeatClock(show)`
- `.useClientTime(use)`
- `.getItemsEventSynchronizer(aliases, events) =>EventSync`

**REX.HMI.init()** This method can be override by the user. The *init* is called when the websocket connection is opened and one can add own items for RW operations. The method can be called either synchronously or asynchronously with callback. Example of the *REX.HMI.init* function is part of the HTML template

#### Example

```
1 // Synchronous version
2 REX.HMI.init = function(){
3 REX.HMI.addItem({alias:"SP", cstring:"task.block:param"});
4 }
5
6 // Init with callback
7 REX.HMI.init = function(done){
8 done();
9 }
```

**REX.HMI.connect()** => **Promise** Connect the RexHMI to the target. This function is called automatically

**REX.HMI.disconnect()** Disconnect all items from the target. Stop reading and dispose connections to all targets.

**REX.HMI.getTarget(url)** => **WSTarget** Return REX target base on the given URL. If URL is null (the most common case) then it returns the default target.

| Param | Type   | Description                 |
|-------|--------|-----------------------------|
| url   | String | URL of the requested target |

#### Example

```
1 // Retrieve version of the default target
2 REX.HMI.getTarget().getVersion().then((data)=>{console.log(data)})
```

**REX.HMI.setTargetUrl(url, force)** => **Promise** Sets the new default target URL. When the page is served from server (not localhost) and the `location.hostname` is set the



*setTargetURL* function sets NULL to use default targetURL. So the target connects to the location which the page is served from.

This behaviour can be changed by setting the `force` parameter to true. Then the 'url' will be used on any occasion.

| Param | Description  |
|-------|--|
| url   | URL of the target  |
| force | set true if the URL should be set even run from server with hostname |

### Example

```

1 // The most common usage
2 // If run locally from file:// connect to IP, when uploaded to server (
  RexCore)
3 // then connect to location.hostname
4 REX.HMI.setTargetUrl("ws://192.168.0.100:8008");
5
6 // Always connect to the localhost
7 REX.HMI.setTargetUrl("ws://127.0.0.1:8008", true);
8
9 // Always use secure WebSocket
10 REX.HMI.setTargetUrl("wss://192.168.0.100:8008", true);

```

**REX.HMI.setRefreshRate(period)** Change the default refresh rate (how fast the data from RexCore will be read)

| Param  | Type   | Default | Description             |
|--------|--------|---------|-------------------------|
| period | number | 500     | New refresh period [ms] |

### Example

```

1 // Change default refresh period to 1000 ms (1s)
2 REX.HMI.setRefreshRate(1000);

```

**REX.HMI.addItem(opt) =>Item** Register new *Item* defined by `alias` and `cstring` for periodical reading and asynchronous writing.

**Returns:** Item - - Registered item

| Param       | Type   | Description                                    |
|-------------|--------|--|
| opt         | Object | Main configuration object                      |
| opt.alias   | string | Alias for the connection string                |
| opt.cstring | string | Connection string of the signal from REX       |
| [opt.url]   | string | URL of the target, if NULL the default is used |

| Param              | Type   | Description   |
|--------------------|--------|---|
| [opt.period]       | number | Item refresh period [ms]  |
| [opt.writeCString] | string | If defined the value of the item will be written to this <i>cstring</i> |

### Example

```

1 // The most common usage
2 var sp = REX.HMI.addItem({alias:"SP", cstring:"task.block:param"});
3 sp.on('change',(data)=>{console.log(data)});
4
5 // Different location of reading and writing (eg. Write before saturation
   and read after)
6 REX.HMI.addItem({alias:"SP", cstring:"task.SAT:y", writeCString:"task.CNR
   :ycn"});

```

**REX.HMI.addItem(items) => Array.<Item>** Add several Items at once. See [HMI#addItem](#) for more details

**Returns:** Array.<Item> - - Return array of added items

| Param | Type            | Description       |
|-------|-----------------|-------------------|
| items | Array. <Object> | An array of items |

**REX.HMI.removeItem(alias) => Promise** Remove an Item based on its *alias*

| Param | Description                                 |
|-------|---|
| alias | Items <i>alias</i> used during registration |

**REX.HMI.get(alias) => Item** Find Item using its *alias*

| Param | Description                                 |
|-------|---|
| alias | Items <i>alias</i> used during registration |

**REX.HMI.\$i(alias) => Item** Find Item using its *alias*

| Param | Description                                 |
|-------|---|
| alias | Items <i>alias</i> used during registration |

**REX.HMI.addGroup(g)** Register custom group of items for R/W operations

| Param | Type         | Description           |
|-------|--------------|-----------------------|
| g     | REX.WS.Group | Group for registering |

### Example

```

1 // Create group see REX.WS.Group
2 var g = new REX.WS.Group({id:"group1", period:100, url:""});
3
4 // Add some items
5 g.addItem(new REX.WS.Item({id:"ITEM-1", cstring:"task.block:param", url:"
  "}));
6
7 // Register events
8 g.on('read',(data)=>{console.log(data)});
9
10 // Register group
11 REX.HMI.addGroup(g);

```

**REX.HMI.removeGroup(g)** Unregister custom group

| Param | Type         | Description                                  |
|-------|--------------|--|
| g     | REX.WS.Group | Instance of group which will be unregistered |

**REX.HMI.addTrend(t)** Unlike REX.WS.Group the Trend reads data from TRND\* blocks. These blocks store several signal with buffered data. Once registered the user can process the data from TRND\* blocks.

| Param | Type         | Description                    |
|-------|--------------|--------------------------------|
| t     | REX.WS.Trend | Trend which will be registered |

### Example

```

1 // Create new trend
2 var trend = REX.WS.Trend({cstring:"task.TRND", id:"TRND-1", period:500,
  readWholeBuffer:true});
3
4 // Register
5 REX.HMI.addTrend(trend);
6 // Add event handlers
7 trend.on('read',(data)=>{console.log(data)});

```

**REX.HMI.removeTrend(t)** Unregister trend

| Param | Type         |
|-------|--------------|
| t     | REX.WS.Trend |

**REX.HMI.writeValues(aliases, values) =>Promise** Write one or several values to the control system. Using already registered items (aliases).

| Param   | Type               | Description                            |
|---------|--------------------|--|
| aliases | Array.<br><String> | An array of already registered aliases |
| values  | Array              | An array of values to be written       |

### Example

```

1 // Register some items
2 REX.HMI.addItem([{alias:"A1",cstring:"task.A1:ycn"},{alias:"A2",cstring:
   "task.A2:ycn"}]);
3
4 // Write values
5 REX.HMI.writeValues(["A1","A2"],[0.5, 0.7]);

```

**REX.HMI.setTitle(title)** Change bot title in header and title of the webpage

| Param | Type   | Description                      |
|-------|--------|----------------------------------|
| title | String | New title for header and webpage |

### Example

```

1 REX.HMI.setTitle("My HMI");

```

**REX.HMI.setHeaderTitle(title)** Change the header title only

| Param | Type   | Description              |
|-------|--------|--------------------------|
| title | String | New title for the header |

**REX.HMI.showHeartBeatClock(show)** If true, the template will display CLOCK in upper right corner of the main screen. When the update of the time stops, the default target is disconnected and the HMI is not updated

| Param | Type    | Description            |
|-------|---------|------------------------|
| show  | Boolean | True to show the clock |

**REX.HMI.useClientTime(use)** Set to True if the time should be displayed in client time not target time. When the target is not able to synchronize with some time server is it possible to use and display times in a client time

| Param | Type    | Description                                   |
|-------|---------|---|
| use   | Boolean | True to use client time instead of target one |

**REX.HMI.getItemsEventSynchronizer(aliases, events) =>EventSync** Return an EventSync object which emits events when all registered items have emitted the same event.

| Param   | Type            | Description                                    |
|---------|-----------------|--|
| aliases | Array<br>String | Array of item aliases or one alias as a string |
| events  | Array<br>String | Array of events which will be monitored        |

### Example

```

1 / Register some items
2 REX.HMI.addItem({alias:"A1",cstring:"task.A1:ycn"},{alias:"A2",cstring:
  "task.A2:ycn"}]);
3
4 var sync = REX.HMI.getItemsEventSynchronizer(["A1","A2"],["read"]);
5
6 sync.on("read",()=>{console.log("All read events emitted")});

```

## 5.3 Reference Guide for REX.HMI.Graph

Time-based graph component which is shown on the bottom of the web page. Graph can read arbitrary signal connected via `alias` and `cstring` or all signals from TRND\* blocks. The Graph is shown when first signal is added over `addSignal` or `addTrend` function.

The REX.HMI.Graph has following functions:

- `.resume()`
- `.pause()`
- `.show()`
- `.hide()`
- `.addSignal(opt)`
- `.addTrend(opt)`
- `.setSize(size)`
- `.setRefreshRate(period)`
- `.hideLegend()`

**REX.HMI.Graph.resume()** Resume redrawing the graph

**Example**

```
1 REX.HMI.Graph.resume()
```

**REX.HMI.Graph.pause()** Pause redrawing the graph

**Example**

```
1 REX.HMI.Graph.pause()
```

**REX.HMI.Graph.show()** Show graph

**Example**

```
1 REX.HMI.Graph.show()
```

**REX.HMI.Graph.hide()** Hide graph

**Example**

```
1 REX.HMI.Graph.hide()
```

**REX.HMI.Graph.addSignal(opt)** Add arbitrary signal from the REX executive to the trend. Warning! The data are stored inside the web page once refreshed all the data will be lost.

| Param         | Type               | Description                              |
|---------------|--------------------|--|
| opt           | Object             | Main configuration object                |
| opt.alias     | string             | Alias for the connection                 |
| [opt.cstring] | string             | Connection string of the signal from REX |
| [opt.desc]    | Array.<br><string> | Signal's description                     |
| [opt.period]  | number             | Signal refresh period [ms]               |

**Example**

```
1 REX.HMI.Graph.addSignal({alias:"Signal-1", cstring:"task.CNR:ycn", desc:"  
Set point", period:500});
```

**REX.HMI.Graph.addTrend(opt)** Add signals from TRND\* blocks to the common graph in HMI

| Param       | Type   | Description                       |
|-------------|--------|-----------------------------------|
| opt         | Object | Main configuration object         |
| opt.cstring | string | Connection string for TRND* block |

| Param        | Type               | Description            |
|--------------|--------------------|------------------------|
| [opt.labels] | Array.<br><string> | Array of signal labels |
| [opt.period] | number             | Graph redraw period    |

### Example

```
1 REX.HMI.Graph.addTrend({cstring:"task.TRND", labels:["signal-1","signal-2"], period:500});
```

**REX.HMI.Graph.setSize(size)** Change size of the graph. The size is in percent of the page.

| Param | Type   | Default | Description                         |
|-------|--------|---------|-------------------------------------|
| size  | number | 0.39    | Size of the graph in percents <0;1> |

### Example

```
1 REX.HMI.Graph.setSize(0.39); // Default
```

**REX.HMI.Graph.setRefreshRate(period)** Change refresh rate of all signals and trends.

| Param  | Description         |
|--------|---------------------|
| period | Refresh period [ms] |

### Example

```
1 REX.HMI.Graph.setRefreshRate(250);
```

**REX.HMI.Graph.hideLegend()** Hide legend of the graph

### Example

```
1 REX.HMI.Graph.hideLegend();
```

# Bibliography

- [1] REX Controls s.r.o.. *Function blocks of the REX Control System – reference manual*, 2016.