KREO HMI
Javascript guidelines

Software

Connect
Ideas.
Shape
solutions.

Table of contents

Scope

This document contains the description of the public API of the integrated JavaScript engine (RTsvg).
This engine let the user JavaScript interact with the Kreo HMI project components.

System variables

The client application exposes a sets of system variables managed as client-only tags.
You can bind them to properties as any other tags.
You can also easily access their values from JavaScript if necessary.

SYS_UIUserName	Name of the current user. This is the empty string when no useris logged in.
SYS_UIUserLevelInteract	Integer ranging [0,10] indicating the security of the current levelfor interacting with controls. The lower the level, the highest theprivileges.
SYS_UILanguageName	Name of the language used to translate displayed texts.
SYS_UILanguageId	Index of the language used to translate displayed texts
SYS_UIClientId	Unique identifier of the client connection. This identifier is unique for all the connections from a given client.
SYS_UIClientIP	IPv4 address of the client as seen by the server. This address maydiffer from the real address when the server is accessed thru NATor proxy.
SYS_UIClientName	Human-readable name of the client as seen as by the server. Thisname is the one assigned to the IPv4 address in Kreo HMI.

RTsvg public API

Functions that can be invoked from user-defined JavaScript scripts without any restriction.

esa.RTdata.setTag (id, val)	RTdata.setTag(*id*, *val*) sets the whole value of the tag *id* to *value*.The function takes care of handling all the special cases where the tag is a client-only tag, a PLC-direct tag, or a standard server tag. If *id* is NOT an integer, the identifier of the tag is searched for using the string representation of *id* within the collection of the tags marked in Kreo HMI as “use in scripts”.
esa.RTdata.getTag(id)	RTdata.getTag(*id*) gets the whole value of the tag *id*. The function takes care of handling all the special cases where the tagis a client-only or standard server tag. If *id* is NOT an integer, the identifier of the tag is searched for using the string representation of *id* within the collection of the tags marked in Crew as “use in scripts”. NOTE: the function does not trigger a PLC action and it returns the last value received from the server or the last written for client-only tags. Thus, this function is guaranteed to return a valid value when the tag is referenced to a currently visible page and the page has been updated by the server. If you need to retrieve a value from the PLC without bothering about the current state of the application, you should use a server-side ST script (see the examples at the end of this document).
esa.RTdata.subscribe	RTdata.subscribe(*tagId*,*owner*,*callback*) registers the callback*callback* of the object *owner* to variations of the tag identifier by its numerical identifier *tagId*. The callback is invoked on the context of its owner (that is *this* matches *owner*) and has as singleparameter a tag object.

esa.RTdata.unsubscribe	RTdata.unsubscribe(*tagId*,*owner*) unregisters the callback previously associated to a call to RTdata.subscribe for the given *tagId* and *owner*. When *tagId* is not supplied, that is invoked as RTdata.unscribe(owner), then all callbacks associated to owner are removed.
esa.RTdata.getTagId	RTdata.getTagId(*tagName, cb*) returns the internal identifier of atag with the given name. The callback cb (tagName, id) will be invoked with the name and associated identifier, -1 in case of failure. This function allows to manipulate tags by name withoutany need to mark the tags with “use in scripts”.
esa.RTdata.getReadTagId	RTdata.getReadTagId(*tagId*) returns the internal identifier of a tag associated to the server-side id. This function must be called when it is required to determine the client-side identifier of a tagfollowing a server-notification of a change to that tag.
esa.RTdata.getWriteTagId	RTdata.getWriteTagId(*tagId*) returns the external identifier of a tag associated to the client-side id. This function must be called when it is required to change the value of a tag on the server fromthe client.
esa.RTdata.getUseInScriptTagId	RTdata.getUseInScriptTagId() returns the internal identifier of a tag from name. Is valid only for tag with flag “UseInScript”. *NB: For client tags use also getWriteTagId for get correct server tagid*

Scripting

The RTprj.scripts namespace/object exposes various client-side functions that may be useful to accessvarious elements from a JavaScript code.

findWidgetByName(name) findWidgetByName(name,popup)	Searches for an SVG widget identified by its name. In the first form, the widget is searched into the current page. In the second form, the widget is searched onto the popup with the given name.The returned value is null when no widget is found or no popup with the given name is opened.

Page Widget

Functions of page widget objects (“findWidgetByName”)

IsVisible()	Check if widget is visible. Check "visibility" attribute of svg element. This method check also parent visibility (Group or template).

Actions

The RTprj.actions namespace/object exposes various client-side functions that trigger actions and specifically intended for JavaScript scripts; these actions are executed on the client side, their effects may also affect the runtime (for instance when loading recipes). The actions are grouped by categories, not all them available to all applications as they may not be relevant in all contexts.

RTprjCallbacks

The `RTprj` namespace/object contains low-level functions related to the applications. You must include the file RT.app.core.js to get access to the namespace.

onDialogActivityListener(key,callback)	Registers or unregisters a callback that will be invoked whenever a window activity occurs on a system dialog. The key is used to identified the callback; when the callback is missing, the key is used to unregister the callback of the previous call. This method is quite useful when you want to show or hide data on the screen when a system dialog is presented or hidden. Take care that system dialogs are working like a stack with many system dialogs opened one on top of the other (though only one may have the focus). See the examples paragraph for a sample usage.

RTprj.actions.apps

The `RTprj.actions.apps` namespace/object contains functions related to the applications, typicallysending a message to the master application to switch application or communicate with another application. You must include the file RT.app.core.js to get access to the namespace.

showApp(appId) showApp(appId, args)	Opens the application indicated to by the given application. Identifier. The application identifier must be one of those registered within the `/config/my_apps.js` file, with the followingconstants recommended for known system ones (with `RTprj.actions.apps` as namespace): • APP_RTSVG • APP_ALARMS • APP_TRENDS • APP_RECIPES • APP_USERS • APP_FDA

RTprj.actions.pages

The RTprj.actions.pages namespace contains functions related to the RTsvg pages (both full screen and pop_up). You must include the file RT.app.svg.core.js to get access to the namespace.

selectPage(pageId)	Presents the page with the given page identifier. If no page exists for the given identifier then nothing happens. If *pageId* is an integer, then the page with the given identifier is presented, either regular or popup. If a regular page and popup page have the same identifier, the regular page is used. If *pageId* is a string, then the page with the given name is presented, either regular or popup. If a regular page and popup page have the same name, the regular page is used. If *pageId* is not given, then the previously displayed page is presented. The implicit navigation stack created by the*selectPage*, *nextPage*, and *previousPage* functions is limited

	to an undefined number of entries, and no assumption should be maderegarding its maximum size (though the maximum is about 10).
nextPage(abs)	Presents the next page in the circular list of the pages making up the current sequence. If there is no sequence or the *abs* parameteris true, then the list of the pages is assumed; in this case the implicit order is determined upon compilation with the pages ordered by pageNo ascending.
previousPage(abs)	Presents the previous page in the circular list of the pages making up the current sequence. If there is no sequence or the *abs* parameter is true, then the list of the pages is assumed; in this casethe implicit order is determined upon compilation with the pages ordered by pageNo ascending.
nextPopup()	Presents the next popup page in the circular list of the popup pages. Upon compilation, the popup pages are ordered by pageNo ascending.
previousPopup()	Presents the previous popup page in the circular list of the popup pages. Upon compilation, the popup pages are ordered by pageNo ascending.
closePopup(pageId)	Closes the popup page with the given page identifier. If no suchpopup page is opened then nothing happens. If *pageId* is an integer, then the popup page with the givenidentifier is closed. If *pageId* is a string, then the popup page with the given name isclosed.
closeAllPopups()	Closes all opened popup pages.

openPopupTo (pageNo, x, y, w, h)	Open the popup page with the given page identifier. If no popup exists for the given identifier then nothing happens. If *pageNo* is an integer, then the popup with the given identifier is open. If *pageNo* is a string, then the popup page with the given name is open. *x, y, w, h [Optional]* position and dimension of popup. If not present are used project default value
setPopupLeft(pname, value) setPopupTop(pname, value) setPopupWidth(pname, value) setPopupHeight(pname, value) setPopupPosition(pname, x, y, w, h)	Set position and dimension of open popup. *pname*: Name of open popup *value*: position/dimension value
openPopupUnder(pname, widget)	Open popup under a specified widget *pname*: Name of open popup *widget*: Javascript widget object data (see `findWidgetByName`)

RTprj.actions.timers

The `RTprj.actions.timers` namespace/object contains functions to command client timers.

start(name)	Starts the timer with the given name
startId(id)	Starts the timer with the given id
stop(name)	Stop the timer with the given name
stopId(id)	Stops the timer with the given id
suspend(name)	Suspend the timer with the given name
suspendId(id)	Suspend the timer with the given id
setCounter(timerName, value)	Set the counter value of a timer with given name
setCounterId(id, value)	Set the counter value of a timer with given id

RTprj.actions.security

The `RTprj.actions.security` namespace/object contains functions related to the identification of the user interacting with the client application. They are mainly limited to login and log off functions that are activating security levels, thus allowing or preventing the access to widgets, pages, or actions. You must include the file RT.app.core.js to get access to the namespace.

login(user, password, cb)	Logs in with the given credentials If user or password are not passed during the call, then its presents the login screen where the user must enter his/her credentials. The optional callback `cb` with prototype `cb(err, name)` is invoked as a result of the call with `err` indicating the status (0 for success, anything else for an error code) and `name` the name of the logged user (if `err` is 0).
logoff()	Logs off the current user and switches back to the previous one, usually the default user account.
getGroupPermission(authName)	Returns permission for actual user group and authorization name
getGeoPermission(authName)	Returns permission for actual security client and authorization name

RTprj.actions.system

The `RTprj.actions.system` namespace/object contains system functions, typically changing heavily the behaviour of the client or server application. You must include the file RT.app.core.js to get access to the namespace.

exitToSystem()	On the terminal, the function closes the client and runtime applications and goes back to the default terminal application. When executed from a remote client, the action acts as if executedlocally on the terminal, however the side-effect of the hosting browser is obviously undefined as one consequence of the action is to terminate the web server.
execFunction(id, argv)	Executes the runtime function id with the given array of parameters, each parameter is presented as 2 entries, the first one indicating the type and the second one the value; the argument types are: • RTcommon.SYS_FUNC_ARGTYPE_CONST (0), a simple constant value (string or number). • RTcommon.SYS_FUNC_ARGTYPE_TAG_REF (2), the value will be read from the value of the tag which identifieris the parameter. For example, writing the value 3 to a tag which identifier is 42 ismade using the call: ExecFunction(1000, 0, 42, 0, 3). Add 5 to the tag 42 is made using the call ExecFunction(1001, 0, 42, 0, 5).
execSTScript(name,args…, cb)	Executes a server-hosted ST script with the given name. The optional callback is invoked with the result of the call, thus allowing retrieving a returned value from the executed script. Thefunction accepts a variable number of parameters to pass to the ST script along with a callback (last parameter) that is invoked with the result of the execution. See the examples then after for further details. Note that the call is not blocking, thus the usage of the callback `cb` to handle the returned code of the ST script.

RTprj.actions.langs

The `RTprj.actions.langs` namespace/object contains languages functions.

selectLanguage (langId)	Activate a specific language
nextLanguage()	Activate the next language
previousLanguage()	Activate the previous language

RTprj.actions.interaction

The `RTprj.actions.interaction` namespace/object contains functions for enable/disable user interaction.

enable	Enable user interaction (only single client)
disable	Disable user interaction (only single client)

Gadgets (external javascript)

The `RTprj.gadgets` is the container for the user gadgets. This exposes one and only one functions that is used to load gadgets. Typically, this container is used by gadgets to add their own requirements or registrations.

loadGadget(arg)	Loads a set of files. The general interface is:
	`esa.RTprj.gadgets.loadGadget({`
	`files : [...list of files...],`
	`onloaded : callback,`
	`Onerror : callback`
	`});`
	
	The files properties is an array of resources JavaScript (.js) and Stylesheet (.css) files to load. When supplied, the `onloaded` function is called whenever a file is loaded. When supplied, the `onerror` function is called when an error occurred whilst loading a file; see the HTML/JavaScript documentation for error handlers on load error for further details.
	
	Note that the files are registered to load in the indicated order butthe actual load order is not guaranteed because under the control of the browser among other things. See the official HTML/JavaScript documentation for details.
	
	In order to access resources located in the installed directory of theapplication, you must use the `/$RESOURCES` prefix in the URLs of the files.

```
esa.RTprj.gadgets.loadGadget({
    files :[
        '/$RESOURCES/navbar_103.js',
        '/$RESOURCES/navbar_003.css'
    ],
    onloaded : function(ev, index, file)
    {
        alert('NAVBAR 102 LOADED: [' + index + ']' + file);
    },
    onerror : function(ev, index, file)
```

```
    {
        alert('FILE NOT EXIST ERROR');
    }
});
```

Examples

Server script with returned value

Sample 1

ST script:

```
FUNCTION FC_Read_String : STRING[100]

    FC_Read_StringAlt := 'Hello World';
END_FUNCTION;
```

JavaScript code to execute the function and read the value:

```
RTprj.actions.system.execSTScript('FC_Read_String', function(err, val)

{

    if (err)

    {

        alert('an error occured: ' + err);
        return;

    }
```

Sample 2

ST script

```
FUNCTION FC_Read_StringAlt
    VAR_OUTPUT

        result : STRING[100];
    END_VAR;

    result := 'Bonjour le monde';
```

JavaScript code to execute the function and read the value:

```
RTprj.actions.system.execSTScript('FC_Read_String', function(err, val)
{
    if (err)
    {
        alert('an error occured: ' + err);
        return;
    }

    alert('script response: ' + val);
});
```

Changing page from server data

The JavaScript requests a value from a server ST script and changes page based on it.

ST script:

```
FUNCTION FC_Read_PageId : INT
    FC_Read_PageId := 2;
END_FUNCTION;
```

JavaScript code to execute the function and change page:

```
RTprj.actions.system.execSTScript('FC_Read_PageId', function(err, val)
{
    if (err)
    {
        alert('an error occured: ' + err);
        return;
    }

    // ensure integer
    const pageId = parseInt(val);
    if (pageId > 0)
    {
        RTprj.actions.pages.selectPage(pageId);
    }
});
```

Reading the value of a Tag

The JavaScript requests the value of a tag using a server ST script. Note that parameters are passed fromJavaScript to ST.

ST script:

```
FUNCTION FC_Read_Tag : INT
    VAR_INPUT
        tagName : WSTRING[100];
    END_VAR;
    FC_Read_Tag := TAG_READVALUE(tagName);
END_FUNCTION;
```

JavaScript code to read the value of a tag with a given name:

```
RTprj.actions.system.execSTScript('FC_Read_Tag', 'Tag1', function(err, val)
{
    if (err)
    {
        alert('an error occured: ' + err);
        return;
    }

    alert('script response: the value of the tag is ' + val);
});
```

Changing the text of a label

The following code change the text of a label named $gadget.title to match the title of the current page:

```
const wtitle = RTprj.scripts.findWidgetByName('$gadget.title');
if (wtitle)
{
    wtitle.content(RTprj.pages[RTprj.pageIndex].title);
}
```

Connect
ideas.
shape
solutions.