

KUKA Ethernet KRL Interface

The *rc_cube* provides an Ethernet KRL Interface (EKI Bridge), which allows communicating with the *rc_cube* from KUKA KRL via KUKA.EthernetKRL XML.

! Note

The component is optional and requires a separate Roboception's EKIBridge [license](#) to be purchased.

! Note

The KUKA.EthernetKRL add-on software package version 2.2 or newer must be activated on the robot controller to use this component.

The EKI Bridge can be used to programmatically

- do service calls, e.g. to start and stop individual computational nodes, or to use offered services such as the hand-eye calibration or the computation of grasp poses;
- set and get run-time parameters of computation nodes, e.g. of the camera, or disparity calculation.

Ethernet connection configuration

The EKI Bridge listens on port 7000 for EKI XML messages and transparently bridges the *rc_cube's* [REST-API](#). The received EKI messages are transformed to JSON and forwarded to the *rc_cube's* REST-API. The response from the REST-API is transformed back to EKI XML.

The EKI Bridge gives access to run-time parameters and offered services of all computational nodes described in [Software components](#) and [Optional software components](#).

The Ethernet connection to the *rc_cube* on the robot controller is configured using XML configuration files. The EKI XML configuration files of all nodes running on the *rc_cube* are listed at [EKI XML configuration files](#).

Each node offering run-time parameters has an XML configuration file for setting and getting its parameters. These are named following the scheme `<node_name>-parameters.xml`. Each node's service has its own XML configuration file. These are named following the scheme `<node_name>-<service_name>.xml`.

All elements in the XML files are preset, except for the IP of the *rc_cube* in the network.

These files must be stored in the directory `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL` of the robot controller and they are read in when a connection is initialized.

As an example, an Ethernet connection to configure the `rc_stereomatching` parameters is established with the following KRL code.

```
DECL EKI_Status RET
RET = EKI_INIT("rc_stereomatching-parameters")
RET = EKI_Open("rc_stereomatching-parameters")

; ----- Desired operation -----

RET = EKI_Close("rc_stereomatching-parameters")
```

! Note

The EKI Bridge automatically terminates the connection to the client if the received XML telegram is invalid.

Generic XML structure

For data transmission, the EKI Bridge uses `<req>` as root XML element (short for request).

The root tag always includes the following elements.

- `<node>`. This includes a child XML element used by the EKI Bridge to identify the target node. The node name is already included in the XML configuration file.
- `<end_of_request>`. End of request flag that triggers the request.

The following listing shows the generic XML structure for data transmission.

```

<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>

```

For data reception, the EKI Bridge uses `<res>` as root XML element (short for response). The root tag always includes a `<return_code>` child element.

```

<RECEIVE>
  <XML>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>

```

❗ Note

By default the XML configuration files uses 998 as flag to notify KRL that the response data record has been received. If this value is already in use, it should be changed in the corresponding XML configuration file.

Return code

The `<return_code>` element consists of a `value` and a `message` attribute.

As for all other components, a successful request returns with a `res/return_code/@value` of 0. Negative values indicate that the request failed. The error message is contained in `res/return_code/@message`. Positive values indicate that the request succeeded with additional information, contained in `res/return_code/@message` as well.

The following codes can be issued by the EKI Bridge component.

Table 42 Return codes of the EKI Bridge component

Code	Description
0	Success
-1	Parsing error in the conversion from XML to JSON

Code	Description
-2	Internal error
-9	Missing or invalid license for EKI Bridge component
-11	Connection error from the REST-API

Note

The EKI Bridge can also return return code values specific to individual nodes. They are documented in the respective [software component](#).

Note

Due to limitations in KRL, the maximum length of a string returned by the EKI Bridge is 512 characters. All messages larger than this value are truncated.

Services

For the nodes' services, the XML schema is generated from the service's arguments and response in JavaScript Object Notation (JSON) described in [Software components](#) and [Optional software components](#). The conversion is done transparently, except for the conversion rules described below.

Conversions of poses:

A pose is a JSON object that includes `position` and `orientation` keys.

```
{
  "pose": {
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
    },
    "orientation": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
      "w": "float64",
    }
  }
}
```

This JSON object is converted to a KRL `FRAME` in the XML message.

```
<pose X="..." Y="..." Z="..." A="..." B="..." C="..."></pose>
```

Positions are converted from meters to millimeters and orientations are converted from quaternions to KUKA ABC (in degrees).

❗ Note

No other unit conversions are included in the EKI Bridge. All dimensions and 3D coordinates that don't belong to a pose are expected and returned in meters.

Arrays:

Arrays are identified by adding the child element `<le>` (short for list element) to the list name. As an example, the JSON object

```
{
  "rectangles": [
    {
      "x": "float64",
      "y": "float64"
    }
  ]
}
```

is converted to the XML fragment

```
<rectangles>
  <le>
    <x>...</x>
    <y>...</y>
  </le>
</rectangles>
```

Use of XML attributes:

All JSON keys whose values are a primitive data type and don't belong to an array are stored in attributes. As an example, the JSON object

```
{
  "item": {
    "uuid": "string",
    "confidence": "float64",
    "rectangle": {
      "x": "float64",
      "y": "float64"
    }
  }
}
```

is converted to the XML fragment

```
<item uuid="..." confidence="...">
  <rectangle x="..." y="...">
  </rectangle>
</item>
```

Request XML structure

The `<SEND>` element in the XML configuration file for a generic service follows the specification below.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/service/<service_name>" Type="STRING"/>
    <ELEMENT Tag="req/args/<argX>" Type="<argX_type>"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

The `<service>` element includes a child XML element that is used by the EKI Bridge to identify the target service from the XML telegram. The service name is already included in the configuration file.

The `<args>` element includes the service arguments and should be configured with `EKI_Set<Type>` KRL instructions.

As an example, the `<SEND>` element of the `rc_itepick`'s `get_load_carriers` service (see [ItemPick and BoxPick](#)) is:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/rc_itepick" Type="STRING"/>
    <ELEMENT Tag="req/service/get_load_carriers" Type="STRING"/>
    <ELEMENT Tag="req/args/load_carrier_ids/le" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

The `<end_of_request>` element allows to have arrays in the request. For configuring an array, the request is split into as many packages as the size of the array. The last telegram contains all tags, including the `<end_of_request>` flag, while all other telegrams contain one array element each.

As an example, for requesting two load carrier models to the `rc_itepick`'s `get_load_carriers` service, the user needs to send two XML messages. The first XML telegram is:

```
<req>
  <args>
    <load_carrier_ids>
      <le>load_carrier1</le>
    </load_carrier_ids>
  </args>
</req>
```

This telegram can be sent from KRL with the `EKI_Send` command, by specifying the list element as path:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_itepick-get_load_carriers", "req/args/load_carrier_ids/le", "load_carrier1")
RET = EKI_Send("rc_itepick-get_load_carriers", "req/args/load_carrier_ids/le")
```

The second telegram includes all tags and triggers the request to the `rc_itepick` node:


```

<req>
  <node>
    <rc_itempick></rc_itempick>
  </node>
  <service>
    <get_load_carriers></get_load_carriers>
  </service>
  <args>
    <load_carrier_ids>
      <le>load_carrier2</le>
    </load_carrier_ids>
  </args>
  <end_of_request></end_of_request>
</req>

```

This telegram can be sent from KRL by specifying `req` as path for `EKI_Send`:

```

DECL EKI_STATUS RET
RET = EKI_SetString("rc_itempick-get_load_carriers", "req/args/load_carrier_ids/le", "load_carrier2")
RET = EKI_Send("rc_itempick-get_load_carriers", "req")

```

Response XML structure

The `<RECEIVE>` element in the XML configuration file for a generic service follows the specification below:

```

<RECEIVE>
  <XML>
    <ELEMENT Tag="res/<resX>" Type="<resX_type>"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>

```

As an example, the `<RECEIVE>` element of the `rc_april_tag_detect`'s `detect` service (see [TagDetect](#)) is:

```

<RECEIVE>
  <XML>
    <ELEMENT Tag="res/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose_frame" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose/@X" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Y" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Z" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@A" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@B" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@C" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/instance_id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/size" Type="REAL"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>

```

For arrays, the response includes multiple instances of the same XML element. Each element is written into a separate buffer within EKI and can be read from the buffer with KRL instructions. The number of instances can be requested with `EKI_CheckBuffer` and each instance can then be read by calling `EKI_Get<Type>`.

As an example, the tag poses received after a call to the `rc_april_tag_detect`'s `detect` service can be read in KRL using the following code:

```

DECL EKI_STATUS RET
DECL INT i
DECL INT num_instances
DECL FRAME poses[32]

DECL FRAME pose = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}

RET = EKI_CheckBuffer("rc_april_tag_detect-detect", "res/tags/le/pose")
num_instances = RET.Buff
for i=1 to num_instances
  RET = EKI_GetFrame("rc_april_tag_detect-detect", "res/tags/le/pose", pose)
  poses[i] = pose
endfor
RET = EKI_ClearBuffer("rc_april_tag_detect-detect", "res")

```

Note

Before each request from EKI to the *rc_cube*, all buffers should be cleared in order to store only the current response in the EKI buffers.

Parameters

All nodes' parameters can be set and queried from the EKI Bridge. The XML configuration file for a generic node follows the specification below:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="req/parameters/<parameter_y>/@value" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@default" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@min" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@max" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@value" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@default" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@min" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@max" Type="REAL"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

The request is interpreted as a *get* request if all parameter's `value` attributes are empty. If any `value` attribute is non-empty, it is interpreted as *set* request of the non-empty parameters.

As an example, the current value of all parameters of `rc_stereomatching` can be queried using the XML telegram:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters></parameters>
  <end_of_request></end_of_request>
</req>
```

This XML telegram can be sent out with Ethernet KRL using:

```
DECL EKI_STATUS RET
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

The response from the EKI Bridge contains all parameters:

```
<res>
  <parameters>
    <acquisition_mode default="Continuous" max="" min="" value="Continuous"/>
    <quality default="High" max="" min="" value="High"/>
    <static_scene default="0" max="1" min="0" value="0"/>
    <seg default="200" max="4000" min="0" value="200"/>
    <smooth default="1" max="1" min="0" value="1"/>
    <fill default="3" max="4" min="0" value="3"/>
    <minconf default="0.5" max="1.0" min="0.5" value="0.5"/>
    <mindepth default="0.1" max="100.0" min="0.1" value="0.1"/>
    <maxdepth default="100.0" max="100.0" min="0.1" value="100.0"/>
    <maxdeptherr default="100.0" max="100.0" min="0.01" value="100.0"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

The `quality` parameter of `rc_stereomatching` can be set to `Low` by the XML telegram:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters>
    <quality value="Low"></quality>
  </parameters>
  <end_of_request></end_of_request>
</req>
```

This XML telegram can be sent out with Ethernet KRL using:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_stereomatching-parameters", "req/parameters/quality/@value", "Low")
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

In this case, only the applied value of `quality` is returned by the EKI Bridge:

```
<res>
  <parameters>
    <quality default="High" max="" min="" value="Low"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

EKI XML configuration files

This section contains EKI XML configuration files for all nodes running on the *rc_cube*. They are also available for download in this [ZIP archive](#).






















rc_april_tag_detect

- [Parameters](#)
- Services:
 - [detect](#)
 - [reset_defaults](#)
 - [restart](#)
 - [save_parameters](#)
 - [start](#)
 - [stop](#)













rc_boxpick

- [Parameters](#)
- Services:
 - [compute_grasps](#)
 - [delete_load_carriers](#)
 - [delete_regions_of_interest](#)
 - [detect_filling_level](#)
 - [detect_items](#)
 - [detect_load_carriers](#)
 - [get_load_carriers](#)
 - [get_regions_of_interest](#)
 - [reset_defaults](#)
 - [save_parameters](#)
 - [set_load_carrier](#)
 - [set_region_of_interest](#)
 - [start](#)
 - [stop](#)











rc_cadmatch

-  Parameters
- Services:
 -  delete_grasps
 -  delete_load_carriers
 -  delete_regions_of_interest
 -  detect_filling_level
 -  detect_load_carriers
 -  detect_object
 -  get_grasps
 -  get_load_carriers
 -  get_preferred_orientation
 -  get_regions_of_interest
 -  get_symmetric_grasps
 -  reset_defaults
 -  save_parameters
 -  set_all_grasps
 -  set_grasp
 -  set_load_carrier
 -  set_preferred_orientation
 -  set_region_of_interest
 -  start
 -  stop




rc_collision_check

-  Parameters
- Services:
 -  check_collisions
 -  delete_grippers
 -  get_grippers
 -  get_grippers_schmalz_vac
 -  get_grippers_schunk
 -  reset_defaults
 -  save_parameters
 -  set_gripper
 -  set_gripper_schmalz_vac
 -  set_gripper_schunk
 -  set_stroke_schunk















rc_hand_eye_calibration

-  Parameters
- Services:
 -  calibrate
 -  get_calibration
 -  remove_calibration
 -  reset_calibration
 -  reset_defaults
 -  save_calibration
 -  save_parameters
 -  set_calibration
 -  set_pose








rc_iocontrol

-  Parameters
- Services:
 -  get_io_values
 -  reset_defaults
 -  save_parameters























rc_itempick

-  Parameters
- Services:
 -  compute_grasps
 -  delete_load_carriers
 -  delete_regions_of_interest
 -  detect_filling_level
 -  detect_load_carriers
 -  get_load_carriers
 -  get_regions_of_interest
 -  reset_defaults
 -  save_parameters
 -  set_load_carrier
 -  set_region_of_interest
 -  start
 -  stop




rc_qr_code_detect

-  Parameters
- Services:
 -  detect
 -  reset_defaults
 -  restart
 -  save_parameters
 -  start
 -  stop

rc_silhouettematch

-  Parameters
- Services:
 -  calibrate_base_plane
 -  delete_base_plane_calibration
 -  delete_grasps
 -  delete_load_carriers
 -  delete_regions_of_interest_2d
 -  detect_filling_level
 -  detect_load_carriers
 -  detect_object
 -  get_base_plane_calibration
 -  get_grasps
 -  get_load_carriers
 -  get_preferred_orientation
 -  get_regions_of_interest_2d
 -  get_symmetric_grasps
 -  reset_defaults
 -  save_parameters
 -  set_all_grasps
 -  set_grasp
 -  set_load_carrier
 -  set_preferred_orientation
 -  set_region_of_interest_2d

rc_stereocamera

-  Parameters
- Services:
 -  reset_defaults
 -  save_parameters

rc_stereomatching

-  Parameters
- Services:
 -  acquisition_trigger
 -  reset_defaults
 -  save_parameters