| Name | Description | Parameter | Return |
|---|---|---|---|
| **Manage instances of the reader/writer modules** | | | |
| AddJointReader | In this method an instance of the joint reader is added to the interface. | ***name:*** Name for the joint reader instance, later used to identify it. (*string*) | - |
| AddJointWriter | In this method an instance of the joint writer is added to the interface. | ***name:*** Name for the joint writer instance, later used to identify it. (*string*) | - |
| AddSkinReader | In this method an instance of the skin reader is added to the interface. | ***name:*** Name for the skin reader instance, later used to identify it. (*string*) | - |
| AddVisualReader | In this method the instance of the visual reader is added to the interface. | - | - |
| RemoveVisualReader | In this method the instance of the visual reader is removed from the interface. | - | - |
| | | | |
| **Joint reader member functions** | | | |
| JointRInit | This method initializes the joint reader instance with the given parameters. | ***name:*** Name of the joint reader instance to identify it. (*string)*<br>***part:*** iCub part which should be controlled with this instance. (*string)*<br>***sigma:*** Standard deviation of the Gaussian encoding for the population coded joint angles. (*double*)<br>***n_pop:*** Set the Population size for the joint angle population coding. (*int*)<br>***degr_per_neuron:*** neuron degree resolution for the population coding; default value 0.0: if non-zero the population size for each joint is computed with the given resolution and the joint angle range. (*double)* | A failure indication value, being true for a successful execution. (*bool*) |
| JointRClose | This method closes the joint reader with a cleanup. | ***name:*** Name of the joint reader instance to identify it. (*string*) | - |
| JointRGetJointCount | This method returns the number of | ***name:*** Name of the joint reader instance to identify it. | Amount of joints associated to the |

| Name | Description | Parameter | Return |
|---|---|---|---|
| | joints controlled by the joint reader. | (*string*) | joint reader (*int*) |
| JointRGetJointsDegRes | This method returns the resolution of the populations encoding the joint angles. | ***name:*** Name of the joint reader instance to identify it. (*string*) | Resolution of the joints in degree. (*vector[double]*) |
| JointRGetNeuronsPerJoint | This method returns the size of the populations encoding the joint angles | ***name:*** Name of the joint reader instance to identify it. (*string*) | Neurons in the populations per associated joint. (*vector[int]*) |
| JointRReadDouble | This method reads the angle of one joint and return the joint angle directly as a double value. | ***name:*** Name of the joint reader instance to identify it. (*string*)<br>*joint:* number of the selected joint (*int*) | Joint angle in degree. (*double*) |
| JointRReadPopAll | This method reads the angles of all joints and returns the joint angles encoded in vectors. | ***name:*** Name of the joint reader instance to identify it. (*string*) | The joint angles of all joints encoded in populations. (*vector[vector[double]]*) |
| JointRReadPopOne | This method reads the angle of one joint and return the joint angle encoded in a vector | ***name:*** Name of the joint reader instance to identify it. (*string*)<br>***joint:*** number of the selected joint (*int*) | The joint angles of the selected joint encoded in populations. (*vector[double]*) |
| | | | |
| **Joint writer member functions** | | | |
| JointWInit | This method initializes the joint writer with the given parameters. | ***name:*** Name of the joint writer instance to identify it. *(string)*<br>***part:*** iCub part which should be controlled with this instance. *(string)*<br>***n_pop:*** Set the Population size for the joint angle population coding. (int)<br>***degr_per_neuron:*** neuron degree resolution for the population coding; default value 0.0: if non-zero the population size for each joint is computed with the given resolution and the joint angle range. (*double)*<br>***speed:*** Joint velocity value set for all joints. (*double*) | A failure indication value, being true for a successful execution. (*bool*) |

| Name | Description | Parameter | Return |
|---|---|---|---|
| JointWClose | This method closes the joint writer with a cleanup. | ***name:*** Name of the joint writer instance to identify it. (*string*) | - |
| JointWGetJointCount | This method returns the number of joints controlled by the joint writer. | ***name:*** Name of the joint writer instance to identify it. (*string*) | Amount of joints associated to the joint writer. (*int*) |
| JointWGetJointsDegRes | This method returns the resolution of the populations encoding the joint angles. | ***name:*** Name of the joint writer instance to identify it. (*string*) | Resolution of the joints in degree (*vector[double])* |
| JointWGetNeuronsPerJoint | This method returns the size of the populations encoding the joint angles | ***name:*** Name of the joint writer instance to identify it. (*string*) | Neurons in the populations per associated joint. (*vector[int]*) |
| JointWSetJointVelocity | In this method the joint velocity is set. Two modes are possible, set the velocity for one selected joint or set the velocity for all joints. | ***name:*** Name of the joint writer instance to identify it. (*string*)<br>***speed:*** The velocity value which should be set. (*double*)<br>***joint:*** number of the selected joint, -1 to set all joints (*int*) | A failure indication value, being true for a successful execution. (*bool*) |
| JointWWriteDoubleAll | This method write all joints with double values. The joint motion can be executed in a non-blocking or a blocking mode. | ***name:*** Name of the joint writer instance to identify it. (*string*)<br>***position:*** joint angle values for all joints (*vector[double]*)<br>***blocking:*** Control the blocking execution of the joint movement. Default value is true. (*bool*) | A failure indication value, being true for a successful execution. (*bool*) |
| JointWWriteDouble | This method write one joint with a double value. The joint motion can be executed in a non-blocking or a blocking mode. | ***name:*** Name of the joint writer instance to identify it. (*string*)<br>***position:*** joint angle value for the selected joint (double)<br>***joint:*** number of the selected joint (*int*)<br>***blocking:*** Control the blocking execution of the joint movement. Default value is true. (*bool*) | A failure indication value, being true for a successful execution. (*bool*) |
| JointWWritePopAll | This method write all joints with joint angles encoded in populations. | ***name:*** Name of the joint writer instance to identify it. (*string*) | A failure indication value, being true for a successful execution. (*bool*) |

| Name | Description | Parameter | Return |
|---|---|---|---|
| | | ***position_pops:*** The joint angles for all joints encoded in populations. (*vector[vector[double]]*)<br>***blocking:*** Control the blocking execution of the joint movement. Default value is true. (*bool*) | |
| JointWWritePopOne | This method write one selected joint with the joint angle encoded in a population. | ***name:*** Name of the joint writer instance to identify it. (*string*)<br>***position_pop:*** The joint angles for all joints encoded in populations. (*vector[double]*)<br>***joint:*** number of the selected joint (*int*)<br>***blocking:*** Control the blocking execution of the joint movement. Default value is true. (*bool*) | A failure indication value, being true for a successful execution. (*bool*) |
| | | | |
| **Skin reader member functions** | | | |
| SkinRInit | This method initialize the skin reader with the given parameters. | ***name:*** Name of the skin reader instance to identify it. (*string*)<br>***arm:*** A parameter selecting the skin side. Right ‘R’/’r’ or left ‘L’/’l’ is possible . (*char*)<br>***norm:*** A bool parameter to control if the tactile sensor data is returned normalized. The default value is true. (*bool*) | A failure indication value, being true for a successful execution. (*bool*) |
| SkinRClose | This method closes the skin reader and cleans up the module. | ***name:*** Name of the skin reader instance to identify it. (*string*) | - |
| SkinRGetTactileArm | Returns the tactile data of the upper arm skin sensors. | ***name:*** Name of the skin reader instance to identify it. (*string*) | Tactile data for the upper arm skin section, divided in steps (first dim: step, second dim: sensor data non-normalized: [0..255] or normalized: [0..1]). (*C++: vector[vector[double]]; Python: numpy array*) |
| SkinRGetTactileForearm | Returns the tactile data of the forearm | ***name:*** Name of the skin reader instance to identify it. | Tactile data for the forearm skin |

| Name | Description | Parameter | Return |
|---|---|---|---|
| | skin sensors. | (*string*) | section, divided in steps (first dim: step, second dim: sensor data non-normalized: [0..255] or normalized: [0..1]). (*C++: vector[vector[double]]; Python: numpy array*) |
| SkinRGetTactileHand | Returns the tactile data for hand skin sensors. | ***name:*** Name of the skin reader instance to identify it. (*string*) | Tactile data for the hand skin section, divided in steps (first dim: step, second dim: sensor data non-normalized: [0..255] or normalized: [0..1]). (*C++: vector[vector[double]]; Python: numpy array*) |
| SkinRGetTaxelPos | Returns the taxel positions, given by the simulator ini-files, for the selected skin section. | ***name:*** Name of the skin reader instance to identify it. (*string*)<br>***skin_part:*** Selection of the skin section. Possible are: arm, forearm and hand.<br>(*string*) | Taxel positions for the different skin sections in relation to the kinematic links. (*C++: vector[vector[double]]; Python: numpy array*) |
| SkinRReadTactile | In this method the complete sensor data is read and splitted in the different skin sections: upper arm, forearm and hand. | ***name:*** Name of the skin reader instance to identify it. (*string*) | A failure indication value, being true for a successful execution. (*bool*) |
| | | | |
| **Visual reader member functions** | | | |
| VisualRInit | Initializes the visual reader with given parameters. Returns a failure indication flag. | ***eye:*** Select the associated eye for the visual reader, 'L'/'l' for the left eye and 'R'/'r' for the right eye. (*char*)<br>***fov_width:*** Choose the width of the output field of view (fov). The iCub fov width is 60°. The default value is the 60°. (*double*)<br>***fov_height:*** Choose the height of the output field of view (fov). The iCub fov height is 48°. The default value is the 48°. (*double*) | A failure indication value, being true for a successful execution. (*bool*) |

| Name | Description | Parameter | Return |
|---|---|---|---|
| | | ***img_width:*** Select the out image width in pixel. The default simulator image output width is 320 px, therefore the default value is 320 px. (*int*)<br>***img_height:*** Select the out image height in pixel. The default simulator image output height is 240 px, therefore the default value is 240 px. (*int*)<br>***fast_filter:*** A flag to select the filter for the image up-scaling. In the case of true a faster filter is selected with a slightly worse quality (cv::INTER_LINEAR). If the flag is not set, a filter with a higher quality is used, which is slower (cv::INTER_CUBIC). The default value for this parameter is true. (*bool*) | |
| VisualRReadFromBuf | This method returns an image vector from the image buffer and removes it from the buffer | - | The camera image of the selected eye. The image is normalized and flattened from 2-D to 1-D (*C++: vector[double]; Python: numpy array*) |
| VisualRStart | This method starts YARP-RF-Module, which reads the images from the iCub, normalizes and flatten the images and finally stores them in a buffer. | - | - |
| VisualRStop | This stop the RF-Module, reading the images from the iCub and terminates it. | - | - |