The Gostai Standard Robotics API

Version 2.3

Gostai

September 30, 2010

Contents

Chapter 1

Gostai Standard Robotics API

This section aims at clarifying the naming conventions in Urbi Engines for standard hardware/-software devices and components implemented as UObject and the corresponding methods/at-tributes/events to access them. The list of available hardware types and software component is increasing and this document will be updated accordingly. Please contact us directly, should you be working on a component not described or closely related to one described here:

standard@gostai.com

Any implementation of an Urbi server must comply with the latest version of this standard to get the "Urbi Ready" certification from Gostai S.A.S.

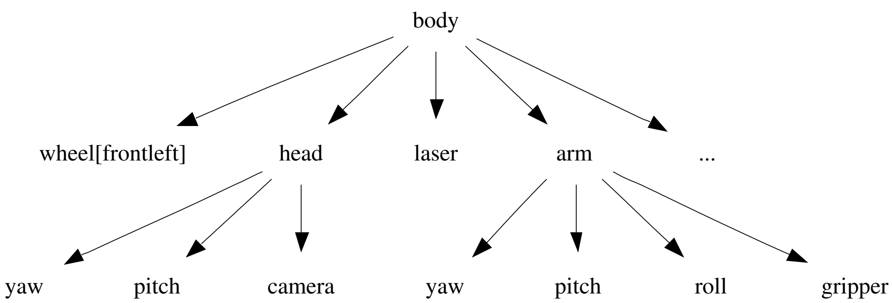
Gostai S.A.S. is currently the only authority which has the ability to deliver an "Urbi Ready" certification.

"Urbi Ready" and the associated logo are trademarks of Gostai S.A.S. and should not be used or displayed in any way without an explicit written agreement from Gostai.

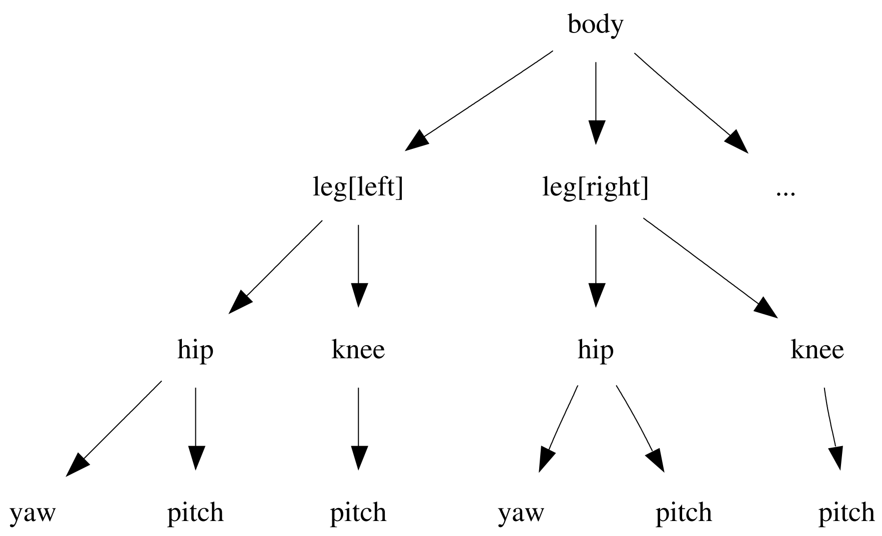
Chapter 2

The Structure Tree

The robot will be described as a set of *components* organized in a hierarchical structure called the *structure tree*. The relationship between a component and a sub-component in the tree is a 'part-of' inclusion relationship. From the point of view of Urbi, each component in the tree is an object, and it contains attributes pointing to its sub-components. Here is an example illustrating a part of a hierarchy that could be found with a wheeled robot with a gripper:



And here is another example for an humanoid robot:



The leaves of the tree are called *devices*, and they usually match physical devices in the robot: motors, sensors, lights, camera, etc. Inside Urbi, the various objects corresponding to the tree components are accessed by following the path of objects inclusions, like in the example below (shortcuts will be described later):

```
body.leg[right].hip.tilt;
body.arm.grip;
body.laser;
// ...
```

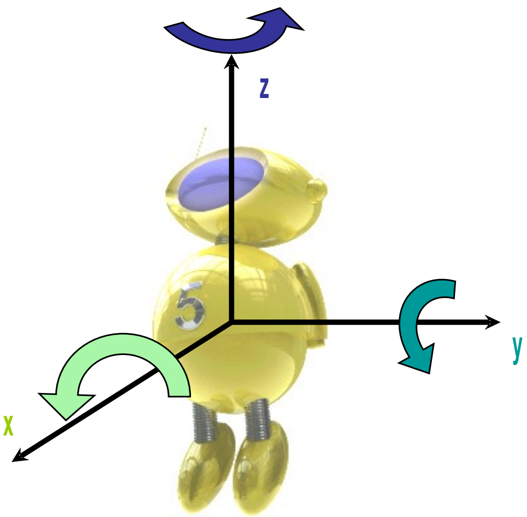
The structure tree should not be mistaken for a representation of the kinematics chain of the robot. The kinematics chain is built from a subset of the devices corresponding to motor devices, and it represents spatial connections between them. Except for these motor devices, the structure tree components do not have a direct counterpart in the kinematics chain, or, if they do, it is as a subset of the kinematics chain (for example, `leg[right]` is a subset of the whole kinematics chain).

The goal of this standard is to provide guidelines on how to define the components and the structure tree, knowing the kinematics chain of the robot.

Chapter 3

Frame of Reference

In many cases, it will be necessary to refer to an absolute frame of reference attached to the robot body. To avoid ambiguities, the standard frame of reference will have the following definition:



Origin the center of mass of the robot

X axis oriented towards the front of the robot. If there is a camera, the front is defined by the default direction of the camera, otherwise the front will be seen as the natural frontal orientation for a mobile robot (the direction of "forward" movement). If the robot is not naturally oriented, the X axis will be chosen to match the main axis of symmetry of the robot body and it will be oriented towards the smallest side, typically the top of a cone for example. In case of a perfectly symmetrical body, the X axis can be chosen arbitrarily but a clear mark should be made visible on the robot body to indicate it.

Z axis oriented in the opposite direction from the gravity. If there is no gravity or natural up/down orientation in the environment or normal operation mode of the robot, the Z axis should be chosen in the direction of the main axis of symmetry in the orthogonal plane defined by the X axis, oriented towards the smallest side. In case of a perfectly symmetrical plane, the Z axis can be chosen arbitrarily but a clear mark should be made visible on the robot body to indicate it.

Y axis oriented to make a right-handed coordinate system.

The axes are oriented in a counter-clockwise direction, as depicted in the illustration above.

Chapter 4

Component naming

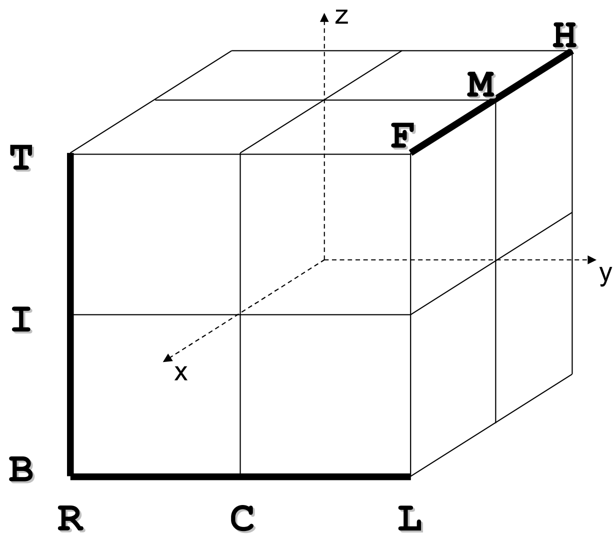
Each component A, which is a sub-component of component B has a name, distinct from the name of all the other components at the same level. This name is a generic designation of what A represents, such as “leg” ,“head”, or “finger”.

Using the correct name for each component is a critical part of this standard. No formal rule can be given to find this name for any possible robot configuration. However, this document includes a table covering many different possible cases. We recommend that robot manufacturers pick from this table the name that fits the most the description of their component.

Chapter 5

Localization

When two identical components A1 and A2, such as the two legs of an humanoid robots, are present in the same sub-component B, an extra node is inserted in the hierarchy to differentiate them. This node is of the `Localizer` type, and provides a `[]` operator, taking a `Localization` argument, used to access each of the identical sub-components. The Urbi SDK provides an implementation for the `Localizer` and `Localization` classes. When possible, localization should be simple geometrical qualifier like *right*/*center*/*left*, *front*/*middle*/*back* or *up*/*in-between*/*down*. Note that "right" or "front" are understood here from the point of view of a man standing and looking in the direction of the X-axis of the robot, and *up/pown* matches the Z-axis, as depicted in the figure below:

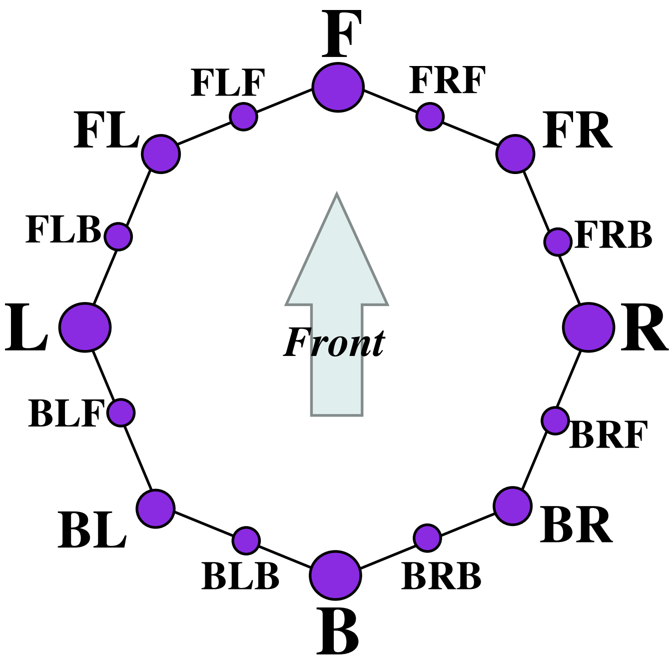


Several geometric qualifiers can be used at the same time to further refine the position. In this case, multiple Localizer nodes are used. As a convention, height information (U/I/D) comes first, followed by depth information (F/M/H), and then side information (R/C/L).

```
// Front-left wheel of a four-wheeled robot:
robot.body.wheel[front][left];
```

```
// Front laser of a robot equipped with multiple sonars:
robot.body.laser[front];
// Left camera from a robot with a stereo camera at the end of an arm:
robot.body.arm.camera[left];
// Top-left LED of the left eye.
robot.body.head.eye[left].led[up][left].val = 1;
// Touch sensor at the end of the front-left leg of a four-legged robot:
robot.body.leg[front][left].foot.touch;
```

You can further qualify a side+depth localization with an additional F/B side information. This can be used in the typical layout below:



This dual positioning using side+depth can also be used to combine side+height or height+depth information.

Layouts with a sequence of three or more identical components can use numbers as their Localization, starting from 0. The smaller the number, the closer to the front, up, or left. For instance, an insectoid robot with 3 legs on each side will use `robot.body.leg[left][0]` to address the frontleft leg.

Layouts with identical components arranged in a circle can also use numeric localization. The component with index 0 should be the uppermost, or front-most if non applicable. Index should increase counterclockwise.

Some components like spines or tails are highly articulated with a set of identical sub-components. When talking about these sub-components, the above localization should be replaced by an array with a numbering starting at 0. The smaller the number, the closer the sub-component is to the robot main body. For surface-like sub-components, like skin touch sensors, the array can be two dimensional.

Other possible localization for sensors are the X, Y and Z axis themselves, like for example for an accelerometer or a gyro sensor, available in each of the three directions.

```
robot.body.accel[x]; // accelerometer in the x direction
```

Examples of component names including localization:

```
leg[right], leg[left];
finger[right], finger[center], finger[left];      // three-finger hand
joint[0], joint[1] ... joint[5] // from tail
touch[478][124]                 // from skin
accel[x], accel[y], accel[z];         // typical accelerometer
gyro[x], gyro[y], gyro[z];            // typical gyro sensor
```

Chapter 6

Interface

An “interface” describes some aspects of a type of device, by specifying the slots and methods that implementations must provide. Each child node of the component hierarchy should implement at least one interface.

For example, for a joint, we can have a “Swivel” interface, used to define patella joints. For the robot body itself, we have a “Mobile” interface describing mobile robots, which includes some standard way of requesting a move forward, a turn, etc.

In short, interfaces are standard Urbi objects that components can inherit from to declare that they have some functionalities.

The following pages describe a few of the most standard interfaces. Each device in the component hierarchy which falls within the category of an interface should implement it.

Each interface defines slots, which can be functions, events or plain data. Some of those slots are optional: they are grayed and put around square brackets in the following table.

6.1 Identity

Contains information about the robot identity.

- `type`
  This describes the robot category among: humanoid, four-legged, wheeled, industrial arm. It gives a general idea of the robot family, but does not replace a more systematic probe of available services by investigating the list of attributes of the object.

- `name`
  Name of the robot.

- `model`
  Model of the robot.

- `serial`
  Serial number (if available).

6.2 Network

Contains information about the network identification of the robot.

- `IP`
  IP address of the robot.

6.3 Motor

This interface is used to describe a generic motor controller.

- `val`
  This slot is a generic pointer to a more specific slot describing the motor position, like `position` or `angle`, depending on the type of motor. It is mandatory in the Urbi Ready standard as a universal proxy to control an actuator. The more specific slot is described in a subclass of `Motor`.

- `PGain?`
  Controls the P gain of the PID controller.

- `IGain?`
  Controls the I gain of the PID controller.

- `DGain?`
  Controls the D gain of the PID controller.

6.4 LinearMotor (subclass of Motor)

This interface is used to describe a linear motor controller.

A wheel can fall in this category, if the reported position is the distance traveled by a point at the surface of the wheel.

- `position`
  Position of the motor in meters. Pointed to by the `val` slot.

- `force`
  Intensity of the measured or estimated force applied on a linear motor.

6.5 LinearSpeedMotor (subclass of Motor)

Motor similar to `LinearMotor`, but controlled by its translation speed.

- `speed`
  Translation speed in meters per second. Pointed to by the `val` slot.

6.6 RotationalMotor (subclass of Motor)

This interface is used to describe a position-controlled rotational motor controller.

- `angle`
  Angle of the motor in radian, modulo 2π. Pointed to by the `val` slot.
- `turn`
  Absolute angular position of the motor, expressed in number of turns.
- `torque`
  Intensity of the measured or estimated torque applied on the motor.

6.7 RotationalSpeedMotor (subclass of Motor)

Interface describing a motor similar to `RotationalMotor` controlled by its rotation speed.

- `speed`
  Rotation speed in radians per second.

6.8 Sensor

This interface is used to describe a generic sensor.

- `val`
  This slot is a generic pointer to a more specific slot describing the sensor value, like `distance` or `temperature`, depending on the type of sensor. It is mandatory in the Urbi Ready standard as a universal proxy to read a sensor. The more specific slot is described in a subclass of `Sensor`.

6.9 DistanceSensor (subclass of Sensor)

This interface is used to describe a distance sensor (infrared, laser, ultrasonic...).

- `distance`
  Measured distance expressed in meters. Pointed to by the `val` slot.

6.10 TouchSensor (subclass of Sensor)

This interface is used to describe a touch pressure sensor (contact, induction,...).

- `pressure`
  Intensity of the pressure put on the touch sensor. Can be 0/1 for simple buttons or expressed in Pascal units. Pointed to by the `val` slot.

6.11 AccelerationSensor (subclass of Sensor)

This interface is used to describe an accelerometer.

- `acceleration`
  Acceleration expressed in m/s^2. Pointed to by the `val` slot.

6.12 GyroSensor (subclass of Sensor)

This interface is used to describe an gyrometer.

- `speed`
  Rotational speed in rad/s. Pointed to by the `val` slot.

6.13 TemperatureSensor (subclass of Sensor)

This interface is used to describe a temperature sensor.

- `temperature`
  Measured temperature in Celsius degrees. Pointed to by the `val` slot.

6.14 Laser (subclass of Sensor)

Interface for a scanning laser rangefinder, or other similar technologies.

- `val`
  Last scan result. Can be either a list of ufloat, or a binary containing a packed array of doubles.

- `angleMin`
  Start scan angle in radians, relative to the front of the device.

- `angleMax`
  End scan angle in radians, relative to the front of the device.

- `resolution`
  Angular resolution of the scan, in radians.

- `distanceMin`
  Minimum measurable distance.

- `distanceMax`
  Maximum measurable distance.

- `rate?`
  Number of scans per second.

Depending on the implementation, some of the parameters can be read-only, or can only accept a few possible values. In that case it is up to the implementer to select the closest possible value to what the user entered. It is the responsibility of the user to read the parameter after setting it to check what value will actually be used by the implementation.

6.15 Mobile

Mobile robots all share this generic interface to provide high order level motion control capabilities.

- `go(`x`)`
  Move approximately x meters forward if x is positive, backward otherwise.

- `turn(`x`)`
  Turn right approximately x radians. x can be a positive or negative value.

6.16 Tracker

Camera-equipped robots can sometimes move the orientation of the field of view horizontally and vertically, which is a very important feature for many applications. In that case, this interface abstracts how such motion can be achieved, whether it is done with a pan/tilt camera or with whole body motion or a combination of both.

- `yaw`
  Rotational articulation around the Z axis in the robot, expressed in radians.

- `pitch`
  Rotational articulation around the Y axis in the robot, expressed in radians.

6.17 VideoIn

The VideoIn interface groups every information relative to cameras or any image sensor.

- `val`
  Image represented as a `Binary` value.

- `xfov`
  The x field of view of the camera expressed in radians.

- `yfov`
  The y field of view of the camera expressed in radians.

- `height`
  Height of the image in the current resolution, expressed in pixels.

- `width`
  Width of the image in the current resolution, expressed in pixels

- `format?`
  Format of the image, expressed as an integer in the enum urbi::UImageFormat. See below for more information.

- `exposure?`
  Exposure duration, expressed in seconds. 0 if non applicable.

- `wb?`
  White balance (expressed with an integer value depending on the camera documentation). 0 if non applicable.

- `gain?`
  Camera gain amplification (expressed as a coefficient between 0 and infinity). 1 if non applicable.

- `resolution?`
  Image resolution, expressed as an integer. 0 corresponds to the maximal resolution of the camera. Successive values correspond to all the supported image sizes in decreasing order. Once modified, the effective resolution in X/Y can be checked with the width and height slots.

- `quality?`
  If the image is in the jpeg format, this slot sets the compression quality, from 0 (best compression, worst quality) to 100 (best quality, bigger image).

The image sensor is expected to use the cheapest way in term of CPU and/or energy consumption to produce images of the requested format. Implementations linked to a physical image sensor do not have to implement all the possible formats. In this case, the format closest to what was requested must be used. A generic image conversion object will be provided. In order to avoid duplicate image conversions when multiple unrelated behaviors need the same format, it is recommended that this object be instantiated in a slot of the VideoIn object named after the format it converts to:

```
if (!robot.body.head.camera.hasSlot("jpeg"))
{
  var robot.body.head.camera.jpeg =
    ImageConversion.new(robot.body.head.camera.getSlot("val"));
  robot.body.head.camera.jpeg.format = 3;
}
```

6.18 AudioOut

The AudioOut interface groups every information relative to speakers.

- `val`
  The speaker value, expressed as a binary, in the format given by the binary header during the assignment.

  Speakers are write-only devices, so there is not much sense in reading the content of this attribute. At best, it returns the remaining sound to be played if it is not over yet, but this is not a requirement.

- `remain`
  The amount of time remaining to play in the speaker sound buffer (expressed in *ms* as a default unit).

- `playing`
  This is a Boolean value which is true when there is a sound currently playing (the buffer is not empty)

- `volume?`
  Volume of the play back, in decibels.

6.19 AudioIn

The AudioIn interface groups every information relative to microphones.

- `val`
  Binary value corresponding to the sound heard, expressed in the current unit (wav, mp3...). The unit can be changed like any other regular unit in Urbi.

  The content is the sound heard by the microphone since the last update event.

- `duration`
  Amount of sound in the val attribute, expressed in *ms*.

- `gain?`
  Microphone gain amplification (expressed between 0 and 1).

6.20 BlobDetector

Ball detectors, marker detectors and various feature-based detectors should all share a similar interface. They extract a part of the image that fits some criteria and define a *blob* accordingly. Here are the typical slots expected:

- `x`
  The x position of the center of the blob in the image

- `y`
  The y position of the center of the blob in the image

- `ratio`
  The size of the blob expressed as a normalized image size: 1 = full image, 0 = nothing.
- `visible`
  A Boolean expressing whether there is a blob in the image or not (see `threshold`).
- `threshold`
  The minimum value of ratio to decide that the blob is visible.
- `orientation?`
  Angle of the main ellipsoid axis of the blob (0 = horizontal), expressed in radians.
- `elongation?`
  Ratio between the main and the second diameter of the blob enveloping ellipse.

6.21 TextToSpeech

Text to speech allows to read text using a speech synthesizer. Default implementations should use the `speaker` component (or alias) as their default sound output.

- `lang?`
  The language used, in international notation (fr, en, it...): ISO 639
- `speed?`
  How fast the voice should go. A positive number, with 1 standing for “regular speed”.
- `pitch?`
  Voice pitch. A positive number, with 1 standing the regular pitch.
- `gender?`
  Gender of the speaker (0:male/1:female).
- `age?`
  Age of the speaker, if applicable.
- `voice?`
  Most TTS engines propose several voices, this attribute allows picking one. It’s a string identifier specific to the TTS developer.
- `say(s)`
  Speak the sentence given in parameter *s*.
- `voicexml?(s)`
  Speak the text *s* expressed as a VoiceXML string.
- `script?(s)`
  Speak the text *s* augmented by script markups (see specific Gostai documentation) to generate Urbi events.

6.22 SpeechRecognizer

Speech recognition allows to transform a stream of sound into a text using various speech recognition algorithms. Implementations should use the `micro` component as their default sound input.

- `lang?`
  The language used, in international notation (fr, en, it...): ISO 639.
- `hear(s)`
  This event has one parameter which is the string describing what the speech engine has recognized (can be a word or a sentence).

6.23 Led

Simple uni-color Led.

- `val`
  Led intensity between 0 and 1.

6.24 RGBLed (subclass of Led)

Tri-color led.

- `r`
  Intensity of the red component, between 0 and 1.
- `g`
  Intensity of the green component, between 0 and 1.
- `b`
  Intensity of the blue component, between 0 and 1.

6.25 Battery

Power source of any kind.

- `remain`
  Estimation of the remaining energy between 0 and 1.
- `capacity`
  Storage capacity in Amp.Hour.
- `current`
  Current current consumption in Amp.

- `voltage`
  Current voltage in Volt.

Chapter 7

Standard Components

Standard components correspond to components typically found in wheeled robots, humanoid or animaloid robots, or in industrial arms. This section provides a list of such components. Whenever possible, robots compatible with the Gostai Standard Robotics API should name all the components in the hierarchy using this list.

7.1 Yaw/Pitch/Roll orientation

Note that Gostai Standard Robotics API considers the orientation to be a component name, and not a localizer. So one would write `head.yaw` and not `head.joint[yaw]` to refer to a rotational articulation of the head around the Z axis.

It is not always clear which rotational direction corresponds to the yaw, pitch or roll components (listed in the table below). This is a quick guideline to help determine the proper association.

Let us consider the robot in its resting, most prototypical position, like "standing" on two or four legs for a humanoid or animaloid, and let all members "naturally" fall under gravity. When gravity has no effect on a certain joint (because it is in the orthogonal plan to Z, for example), the medium position between rangemin and rangemax should be used. The body position achieved will be considered as a reference. Then for each component that is described in terms of yaw/pitch/roll sub-decomposition, the association will be as follow:

yaw rotational articulation around the Z axis in the robot.

pitch rotational articulation around the Y axis in the robot.

roll rotational articulation around the X axis in the robot.

When there is no exact match with the X/Y/Z axis, the closest match, or the default remaining available axis, should be selected to determine the yaw/pitch/roll meaning.

7.2 Standard Component List

The following table summarizes the currently referenced standard components, with a description of potential components that they could be sub-component of, a description of potential components they may contain, and a list of relevant interfaces. This table should be seen as a quick reference guide to identify available components in a given robot.

Name	**Description**	**Sub. of**	**Contains**	**Facets**
`robot`	This is the main component that represents an abstraction of the robot, and the root node of the whole component hierarchy.		`body`	`Identity` `Network` `Mobile` `Tracker`
`body`	This is the main component that contains every piece of hardware in the robot. This includes all primary kinematics sub-chains (arms, legs, neck, head, etc) and non-localized sensor arrays, typically body skin or heat detectors. Localized sensors, like fingertips touch sensors, will typically be found attached to the finger component they belong and not directly to the body.	`robot`	`wheel` `arm leg` `neck head` `wheel` `tail skin` `torso ...`	
`wheel`	Wheel attached to its parent component.	`body`		`Rotational-Motor`
`leg`	Legs are found in humanoid or animaloid robots and correspond to part of the kinematics chain that are attached to the main body by one extremity only and which do touch the ground in normal operation mode (unlike arms). A typical configuration for humanoids contains a hip, a knee and an ankle. If the leg is more segmented, the leg can be described with a simple array of joints.	`body`	`hip knee` `ankle` `foot` `joint`	

Name	Description	Sub. of	Contains	Facets
`arm`	Unlike legs, an arm's extremity does not always touch the ground in normal operating mode. This applies to humanoid robots or single-arm industrial robots. Arms supersede legs in the nomenclature: if a body part behaves alternatively like an arm and like a leg, it will be considered as an arm.	`body`	`shoulder` `elbow` `wrist` `hand` `grip` `joint`	
`shoulder`	The shoulder is the upper part of the arm. It can have one, two or three degrees of freedom and is the closest part of the arm relative to the body.	`arm`	`yaw` `pitch` `roll`	
`elbow`	Separates the upper arm and the lower arm, this is usually a single rotational axis.	`arm`	`pitch`	
`wrist`	Connects the hand and the lower part of the arm. Usually three degrees of freedom axis.	`arm`	`yaw` `pitch` `roll`	
`hand`	The hand is an extension of the arm that usually holds fingers. It's not the wrist, which is articulated and between the arm and the hand.	`arm`	`finger`	
`finger`	Fingers are a series of articulated motors at the extremity of the arm, and connected to the hand. They are usually localized with arrays and/or lateral localization respective to the hand.	`hand`	`touch`	`Motor`
`grip`	Simple two-fingers system.	`arm` `hand`	`touch`	`Motor`
`hip`	The hip is the upper part of the leg and connects it to the main body. It can have one, two or three degrees of freedom.	`leg`	`yaw` `pitch` `roll`	
`knee`	Separates the upper leg and the lower leg, this is usually a single rotational axis.	`leg`	`pitch`	
`ankle`	Connects the foot and the lower part of the leg. Usually three degrees of freedom axis.	`leg`	`yaw` `pitch` `roll`	
`foot`	The foot is an extension of the leg that usually holds toes. It's not the ankle, which is articulated and between the leg and the foot. The foot can also contain touch sensors in simple configurations.	`leg`	`touch`	

Name	Description	Sub. of	Contains	Facets
`toe`	Like fingers, but attached to the foot.	`foot`	`touch`	`Motor`
`neck`	The neck corresponds to a degree of freedom not part of the head, but relative to the rigid connection between the head and the main body.	`body`	`yaw pitch roll`	
`tail`	A tail is a series of articulated motors at the back of the robot.	`body`	`joint`	
`head`	The head main pivotal axis.	`body neck`	`camera mouth ear lip eye eyebrow`	
`mouth`	The robot mouth (open/close)	`head`	`lip`	`Motor`
`ear`	Ears may have degrees of freedom in certain robots.	`head`		`Motor`
`joint`	Generic articulation in the robot.	`tail arm leg lip`		`Motor`
`yaw`	Rotational articulation around the Z axis in the robot. See Section 7.1.	`body neck knee ankle shoulder elbow wrist torso`		`Rotational-Motor Rotational-Speed-Motor`
`pitch`	Rotational articulation around the Y axis in the robot. See Section 7.1.	`body neck knee ankle shoulder elbow wrist torso`		`Rotational-Motor Rotational-Speed-Motor`
`roll`	Rotational articulation around the X axis in the robot. See Section 7.1.	`body neck knee ankle shoulder elbow wrist torso`		`Rotational-Motor Rotational-Speed-Motor`

Name	Description	Sub. of	Contains	Facets
`x`	Translational movement along the X axis.	`body arm`		`Linear-Motor Linear-Speed-Motor`
`y`	Translational movement along the Y axis.	`body arm`		`Linear-Motor Linear-Speed-Motor`
`z`	Translational movement along the Z axis.	`body arm`		`Linear-Motor Linear-Speed-Motor`
`lip`	Corresponds to animated lips.	`mouth`	`joint`	`Motor`
`eye`	Corresponds to the eyeball pivotal axis.	`head`	`camera`	
`eyebrow`	Some robots will have eyebrows with generally one or several degrees of freedom.	`head`	`joint`	`Motor`
`torso`	This corresponds to a pivotal or rotational axis in the middle of the main body.	`body`	`yaw pitch roll`	
`spine`	This is a more elaborated version of "torso", with a series of articulations to give several degrees of freedom in the back of the robot.	`torso`	`joint`	
`clavicle`	This is not to be mixed up with the "top of the arm" body part. It is an independent degree of freedom that can be used to bring the two arms closer in a sort of "shoulder raising" movement.	`body`		`Motor`
`touch`	Touch sensor.	`finger grip foot toe`		`TouchSensor`
`gyro`	Gyrometer sensor.	`body`		`GyroSensor`
`accel`	Accelerometer sensor.	`body`		`Accel-eration-Sensor`

Name	**Description**	**Sub. of**	**Contains**	**Facets**
`camera`	Camera sensor. If several cameras are available, localization shall apply; however there must always be an alias from `camera` to one of the effective cameras (like `cameraR` or `cameraL`).	`head body`		`VideoIn`
`speaker`	Speaker device. If several speakers are available, localization shall apply; however there must always be an alias from `speaker` to one of the effective speakers (like `speakerR` or `speakerL`).	`head body`		`AudioIn`
`micro`	Microphone devices. If several microphones are available, localization shall apply; however there must always be an alias from `micro` to one of the effective microphones (like `microR` or `microL`).	`head body`		`AudioOut`
`speech`	Speech recognition component.	`robot`		`Speech-Recognizer`
`voice`	Voice synthesis component.	`robot`		`TextTo-Speech`

Chapter 8

Compact notation

Components are usually identified with their full-length name, which is the path to access them inside the structure tree. For convenience and backward compatibility with pre-2.0 versions of Urbi, there is also a compact notation available. We will describe here how to construct the compact notation starting from the full name and the structure tree.

Full name	Compact name
`robot.body.armR.elbow`	`elbowR`
`robot.body.head.yaw`	`headYaw`
`robot.body.legL.knee.pitch`	`kneeL`
`robot.body.armR.hand.finger[3][2]`	`fingerR[3][2]`
`robot.body.armL.hand.fingerR`	`fingerLR`

The rule is to move every localization qualifier at the end of the compact notation, in the order where they appear in the full-length name. The remaining component names should then be considered one by one to see if they are needed to remove ambiguities. If they are not, like typically the robot or body components which are shared with almost every other full-length name, they can be ignored. If finally several component names have to be kept, they should be separated by using upper case letters for the first character instead of a dot, like in Java-style notation.

Example 1 (`robot.body.armL.hand.fingerR`)

1. *Move all localization at the end: `robot.body.arm.hand.fingerLR`*
2. *The full name remaining is: `robot.body.arm.hand.finger`*
3. *`finger` should be kept, `hand`, `arm`, `body` and `robot` are not necessary since every finger component will always be attached only to a hand, itself attached to an arm and a body and a robot.*
4. *The result is `fingerLR`*

Example 2 (`robot.body.head.yaw`)

1. *No localization to move*
2. `yaw` *must be kept because* `head` *also have a* `pitch` *sub-component and*
3. `head` *must also be kept to avoid ambiguity with other components having a* `yaw` *sub-component.*
4. *The result is* `headYaw`

Example 3 (`robot.body.legL.knee.pitch`)

1. *Move all localization at the end:* `robot.body.leg.knee.pitchL`
2. `pitch` *is not necessary because* `knee` *has only a* `pitch`*, so* `knee` *will be kept only*
3. *The result is* `kneeL`

Chapter 9

Support classes

The Urbi SDK provides a few support urbiscript classes to help you build the component hierarchy. You can access to those classes by including the files '`urbi/naming-standard.u`' and '`urbi/component.h`'.

9.0.1 Interface

The `Interface` class contains urbiscript objects for all the interfaces defined in this document. Implementations must inherit from the correct interface.

```
// Instantiate a camera.
var cam = myCamera.new();
// Make it inherit from VideoIn.
cam.addProto(Interface.VideoIn);
```

The `Interface.interfaces` method can be called to get a list of all the interfaces an object implements.

9.0.2 Component

The `Component` class can be used to create intermediate nodes of the hierarchy. It provides the following methods:

- `addComponent(`*`name`*`)`
  Add a new sub-component to the current component. *`name`* can be the name of the new component to create, or an instance of `Component`.

- `addDevice(`*`name`*`, `*`value`*`)`
  Add device *`value`* as sub-component, under the name *`name`*. The device must inherit from at least one Interface.

- `makeCompactNames`
  This function must be called once on the root node (`robot`) after the hierarchy is completed. It automatically computes the short name of all the devices, and insert them as slots of the `Global` object.

- `dump`
  Display a hierarchical view of the component hierarchy.

- `flatDump`
  Display all the devices in the hierarchy, sorted by the Interface they implement.

9.0.3 Localizer

The `Localizer` class is a special type of `Component` that stores other components based on their localization. It provides a `[]` operator that takes a `Localization`, such as `top,left,front`, and that can be used to set and get the `Component` or device associated with that `Localization`.

Note that the `[]` function is using a mechanisms to automatically look for its argument as a slot of `Localizer`. As a consequence, you cannot pass a variable to this function, but only one of the constant `Localization`. To pass a variable, use the `get(loc)` or the `set(loc, value)` function.

The following example illustrates a typical instantiation sequence:

```
// Create the top-level node.
var Global.robot = Component.new("robot");
robot.addComponent("head");
var cam = MyCamera.new;
cam.addProto(Interface.VideoIn);
robot.head.addDevice("camera", cam);
// Add two wheels
robot.addComponent(Localizer.new("wheel"));
robot.wheel[left] = MyWheel.new(0).addProto(Interface.RotationalMotor);
robot.wheel[right] = MyWheel.new(1).addProto(Interface.RotationalMotor);
// Implement the Mobile facet in urbiscript:
var robot.go = function(d)
{
  robot.wheel.val = robot.wheel.val + d / wheelRadius adaptive:1
};
var robot.turn = function(r)
{
  var v = r * wheelDistance / wheelRadius;
  robot.wheel[left].val = robot.wheel[left].val + v adaptive:1 &
  robot.wheel[right].val = robot.wheel[right].val - v adaptive:1
};
robot.addProto(Interface.Mobile);
robot.makeCompactNames;
// Let us see the result:
robot.flatDump;
[00010130] *** Mobile: robot
[00010130] *** RotationalMotor: wheelL wheelR
[00010130] *** VideoIn: camera
```

Chapter 10

Index