API Documentation for Modules of Smart Home Environments

Bachelor's Thesis

Gianluca Andrea Vinzens
08-910-606
vinzensg@student.ethz.ch

Distributed Systems Group
Bachelor's Thesis
ETH Zürich

Supervisors:
Dipl.-Ing. Matthias Kovatsch,
Prof. Dr. Friedemann Mattern

July 26, 2012

Contents

Chapter 1

Persisting Service

1.1 Description

The persisting service can be used to store historical data about different device sources. A persisting task specifies a source resource and the persisting service collects its data and stores it in a database.
Two data types are possible, i.e. strings and numbers. Data can be retrieved from the database through a RESTful interface. Time dependent data retrieval can be used for both data types. Numbers also support different aggregation forms.

1.2 Setup

Initially only the /**persistingservice** resource, together with the top resource /**general** is available. Persisting tasks can then be created and bundled to logical entities using top resources (Figure 1.2). Usually general persisting tasks for the /**general** top resource are created, when they serve a universal purpose. Persisting tasks belonging to an application are bundled using a unique top resource, preferably with the application name.

1.3 Create new Task

In order to collect data, a persisting task has to be created. This can be accomplished through sending a **POST request** on the /**tasks** resource with a specific payload. Initially the persisting task does not start collecting data from the source device. It first has to be started by sending a **PUT request** on the /**running** resource (see **Use**).

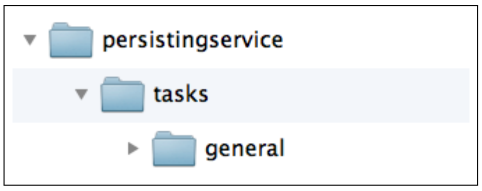


Figure 1.1: Initial Setup

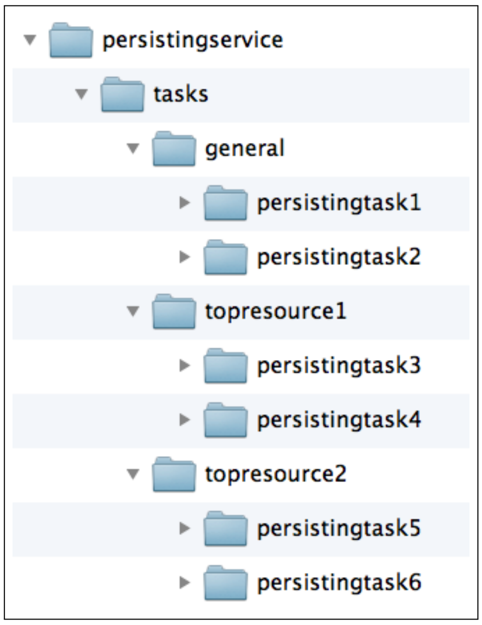


Figure 1.2: Persisting Tasks

Parameter	Content
topid =	The top resource identification is used to bundle persisting resources. **Info:** It is also used as the name for the top resource. To bundle different persisting resource together, the same topid has to be specified. The persisting service then automatically puts them into the same top resource. The same data source resource can be specified for multiple top resources. Data retrieval distinguishes those different persisting resources. /**general** is a reserved top resource identification to put the persisting tasks, which don't belong to a specific application, but can be accessed from all sorts of apps.
resid =	The resource identification for the new persisting resource. **Info:** The resource identification is also used for the name of the persisting task.

Parameter		Content	
deviceroot	=	The path to the device root of the source resource.	
		Example:	coap://localhost:5685/thermostat/temperature device root: coap://localhost:5685/thermostat
deviceres	=	The path from the device root to the actual source device resource	
		Example:	coap://localhost:5685/thermostat/temperature device resource: /temperature
options (opt)	=	The options used by the persisting task to retrieve data from a source device resource.	
		Info:	Data for the same data source resource with different options are distinguished by the persisting service.
		Example:	OPTION1=OPT1&OPTION2=OPT2
type	=	The type of data to be stored in the database.	
		Values:	string | number
		Info:	In general a number value can be stored as either string or number. Only numbers support aggregated value retrieval from the database.

A created persisting task offers a set of resources to access information about the task and interact with it (Figure 1.3). The exact meaning of each resource and their possible requests are explained in the **Use** section.

1.4 Use

1.4.1 General Requests

The persisting resource offers general requests, which are always possible to perform as long as the service is running.

URI: /persistingservice

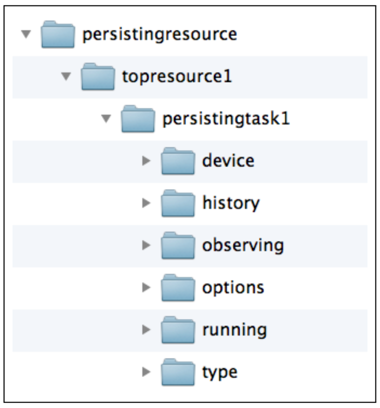


Figure 1.3: Persisting Task

URI	Request	Notes
/tasks	GET	Returns all general persisting service tasks and top resources.

URI: /persistingservice/tasks

URI	Request	Notes
/topresource1	GET	Returns a list of all persisting tasks belonging to this top resource.
/topresource1/ persistingtask1	DELETE	Removes the persisting task from the persisting service. **Info:** The data remains in the database. It is possible to later on create the same persisting task again and still have access to all values recorded. If this is the only persisting resource for the top resource, it will also be removed.

1.4.2 Task Requests

Created tasks offer an interface for interaction and access of general information about them (Figure 1.3).

URI: /persistingservice/tasks/topresource1/persistingtask1

URI	Request	Notes
/deviceinfo	GET	Returns the source device resource's .well-known core information. **Values:** unknown: if .well-known/core is not available for any reason.
/devicepath	GET	Returns the source device resource, from where the data is collected from.
/observing	GET	Returns the observing status. **Values:** true: observing false: polling
/options	GET	Returns the options, if specified at creation.
/type	GET	Returns the type of data being stored. **Values:** string | number
/running	GET	Returns the running status. **Values:** true: running false: not running **Info:** Data can be retrieved from the database even if it is not running.
/running	PUT	Changes the running status. **Payload:** true: persisting task starts collecting data from the source. false: collecting data from the source stops. false;withstorage: collecting data from the source stops, but the persisting service fetches the data one last time and stores it in the database before stopping. **Info:** Data can be retrieved from the database even if it is not running.

1.4.3 History Requests

Data can be retrieved from the database through the **/history** resource. Depending on the data type, more or less retrieval variants are possible. For strings only time dependent data retrieval is possible (Figure 1.4), whereas numbers offer a range of aggregations with each time dependent retrieval option (Figure 1.5). Data retrieved from the database is always dependent on the source device, the top resource and the options. The possible **time dependent requests** are:

URI: /persistingservice/tasks/topresource1/persistingtask1/history

URI	Request	Notes
/all	GET	Returns a list of all the values ever stored for the device.
/last	GET	Returns a list of the last X values stored in the database. **Options:** limit=<1-1000> withddate=true (opt) **Values:** withdate=true: VALUE;yyyy/MM/dd-HH:mm:ss
/newest	GET	Returns the newest (most recent) value stored in the database.
/onday	GET	Returns all the values stored on the specified day. **Options:** date=yyyy/MM/dd-HH:mm:ss withdate=true (opt) **Values:** withdate=true: VALUE;yyyy/MM/dd-HH:mm:ss
/since	GET	Returns all the values stored since some specified date. **Options:** date=yyyy/MM/dd-HH:mm:ss withdate=true (opt) **Values:** withdate=true: VALUE;yyyy/MM/dd-HH:mm:ss

URI	Request	Notes
/timerange	GET	Returns all the values stored between two specified dates. **Options:** startdate=yyyy/MM/dd-HH:mm:ss enddate=yyyy/MM/dd-HH:mm:ss withdate=true (opt) **Values:** withdate=true: VALUE;yyyy/MM/dd-HH:mm:ss

Depending on the type, a persisting task also supports aggregations of values. The aggregation is performed over the list of values returned by the time constraint.

URI: /persistingservice/tasks/topresource1/persistingtask1/history/all
/persistingservice/tasks/topresource1/persistingtask1/history/last
/persistingservice/tasks/topresource1/persistingtask1/history/onday
/persistingservice/tasks/topresource1/persistingtask1/history/since
/persistingservice/tasks/topresource1/persistingtask1/history/timerange

URI	Request	Notes
/sum	GET	Returns the sum of the list of values.
/avg	GET	Returns the average of the list of values. **Options:** weighted=true (opt) **Info:** The weighted average takes into account the time a value was valid, i.e. until a new one was stored for the same source device. The last value stored is always considered to be valid for only one millisecond. This makes sense, when an observer registered on this resource.
/max	GET	Returns the maximum of the list of values.
/min	GET	Returns the minimum of the list of values.

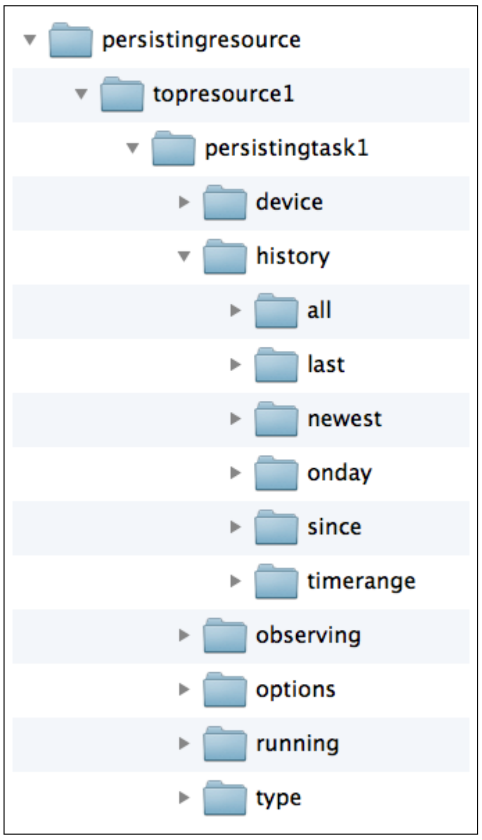


Figure 1.4: String

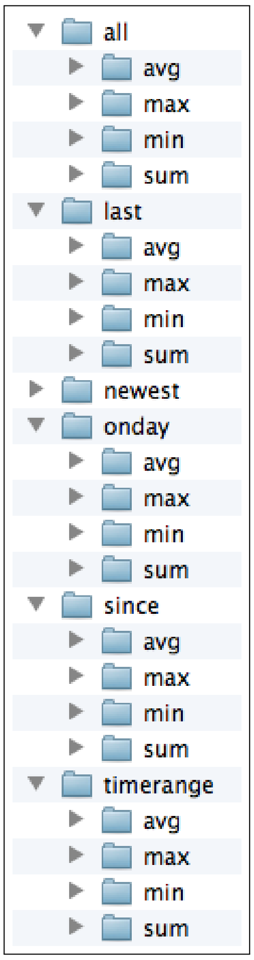


Figure 1.5: Number

Chapter 2
Timed Action

2.1 Description

This module can be used to create timed actions. A timed action has a specific time associated to it, which defines the time it will be executed. The execution time can be specified through a date or a delay. POST, PUT and DELETE requests are possible.

2.2 Setup

Initially the timed action module only contains the resource /**tasks**, which is still empty (Figure 2.1). All timed action tasks will be put into that resource one after the other (Figure 2.2).

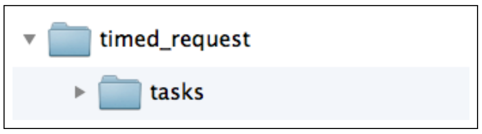


Figure 2.1: Initial Setup

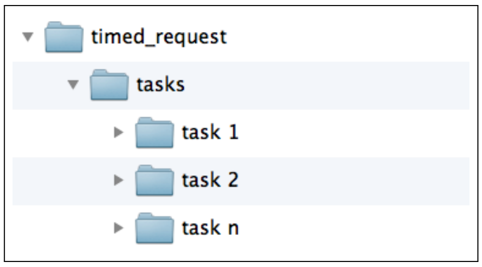


Figure 2.2: Tasks

2.3 Create new Task

In order to create a new task of a timed action, a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter		Content
resid	=	The resource identification for the new task. **Info:** The resource identification will also be used as the name of the task.
target	=	The target device for the timed task.
operation	=	The operation to be performed once the execution time has been reached. **Values:** PUT | POST | DELETE
datetime	=	The date and time when the timed action should be executed. **Values:** yyyy/MM/dd-HH:mm:ss | yyyy/MM/dd | HH:mm:ss **Info:** yyyy/MM/dd uses the time 00:00:00 HH:mm:ss uses the current date

payload (opt)	=	The payload that will be sent with the periodic request
		
		Values: Multiline payloads are possible:
		payload = ;;MULTILINE_PAYLOAD;;
		
		Info: If inside the multiline payload ;; is needed, then use **;:;** , which is replaced by ;; .
		
		Example: payload = ;;
		resid = test
		payload = **;:;**MULTILINE_PAYLOAD**;:;**
		other = ...;;
		
		Payload:
		resid = test
		payload = **;;**MULTILINE_PAYLOAD**;;**
		other = ...

The created task instance provides an interface for access to information about it and interaction with it (Figure 2.3).

2.4 Use

The timed action module supports different requests for its resources. This section gives a detailed overview of all the possible requests for each resource.

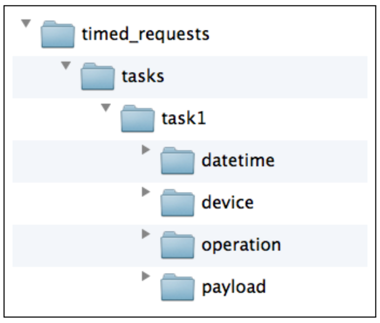


Figure 2.3: Timed Action Task

2.4.1 General Requests

General requests can always be performed, as long as the module is running.

URI: /timed_request

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this module.
/tasks/task1	DELETE	Removes the task from the module.

2.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /timed_request/tasks/task1

URI	Request	Notes
/datetime	GET	Returns the date and time of the planned request execution.
/datetime	PUT	Changes the date and time of the planned execution. **Payload:** yyyy/MM/dd-HH:mm:ss | yyyy/MM/dd | HH:mm:ss **Info:** yyyy/MM/dd uses the time 00:00:00 HH:mm:ss uses the current date
/target	GET	Returns the target device for the planned request.

URI	Request	Notes
/operation	GET	Returns the operation of the planned request. **Values:** PUT | POST | DELETE
/payload	GET	Returns the payload of the planned request
/payload	PUT	Changes the payload of the planned request **Payload:** Any string.

Chapter 3

Periodic Action

3.1 Description

This module can be used to create periodic actions. A periodic action periodically executes requests on a target device. Both the period and the payload can be changed while the periodic task is running. POST, PUT and DELETE requests are possible.

3.2 Setup

Initially the periodic action module only contains the resource /**tasks**, which is still empty (Figure 3.1). All periodic action tasks will be put into that resource one after the other (Figure 3.2).

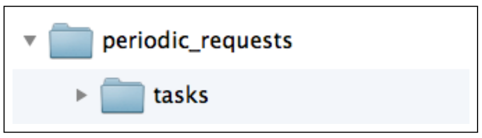


Figure 3.1: Initial Setup

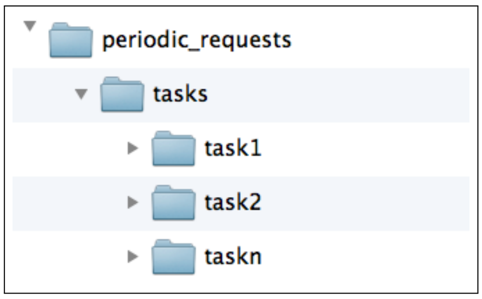


Figure 3.2: Tasks

3.3 Create new Task

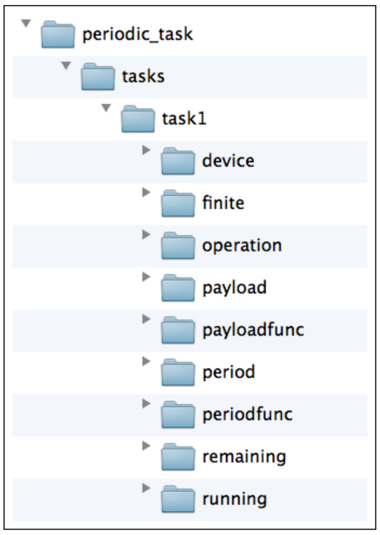
In order to create a new task of a periodic action a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter		Content
resid	=	The resource identification for the new task. **Info:** The resource identification will also be used as the name of the task.
target	=	The target device for the periodic task.
operation	=	The operation to be performed for each periodic action. **Values:** PUT | POST | DELETE
period	=	The period defines the interval between two requests.
periodfunc (opt)	=	The period function is used to set the period. The period can be variable and change after each interval. **Values:** inc;;START;;STEP;;END set;;VALUE1;;VALUE2;;...;;VALUEn own;;FUNCTION;; **Info:** If period is defined this parameter will be ignored For more information on the possible period functions see the **Predefined Functions** section.
finite	=	A fixed number, which defines the number of repetitions for the periodic action. **Info:** When finite is defined the periodic action becomes finite and stops after executing the fixed number of requests.

payload (opt)	=	The payload that will be sent with the periodic action
		
		Values: Multiline payloads are possible:
		payload = ;;MULTILINE_PAYLOAD;;
		
		Info: If inside the multiline payload ;; is needed, then use **;:;** , which is replaced by ;; .
		
		Example: payload = ;;
		resid = test
		payload = **;:;**MULTILINE_PAYLOAD**;:;**
		other = ...;;
		
		Payload:
		resid = test
		payload = **;;**MULTILINE_PAYLOAD**;;**
		other = ...

Parameter	Content
payloadfunc = (opt)	The payload function is used to set the payload. The payload can change after each period. **Values:** inc;;START;;STEP;;END set;;VALUE1;;VALUE2;;...;;VALUEn own;;FUNCTION;; **Info:** If payload is defined this parameter will be ignored. For more information on the possible payload functions see the **Predefined Functions** section.

The created task instance offers a set of resources to access information about the task and interact with it (Figure 3.3).



```
periodic_task
  tasks
    task1
      device
      finite
      operation
      payload
      payloadfunc
      period
      periodfunc
      remaining
      running
```

Figure 3.3: Periodic Action Task

3.4 Use

3.4.1 General Requests

General requests can always be performed, as long as the module is running.

URI: /periodic_request

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this module.
/tasks/task1	DELETE	Removes the task from the module.

3.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /periodic_request/tasks/task1

URI	Request	Notes
/target	GET	Returns the target device for the request.
/finite	GET	Returns the a fixed number of repetitions, if specified at creation.
/operation	GET	Returns the operation of the request. **Values:** PUT | POST | DELETE
/payload	GET	Returns the payload.
/payload	PUT	Changes the payload. **Payload:** Any string
/payloadfunc	GET	Returns the payload function, if specified at creation.
/payloadfunc	PUT	Changes the payload function. **Payload:** inc;;START;;STEP;;END set;;VALUE1;;VALUE2;;...;;VALUEn own;;FUNCTION;; remove **Info:** When passing **remove** as payload, the function is removed and the constant payload becomes active. For more information on the possible payload functions see the **Predefined Functions** section.
/period	GET	Returns the interval between two requests.

URI	Request	Notes
/period	PUT	Changes the period. **Payload:** Any number
/periodfunc	GET	Returns the period function, if specified at creation.
/periodfunc	PUT	Changes the period function. **Payload:** inc;;START;;STEP;;END set;;VALUE1;;VALUE2;;...;;VALUEn own;;FUNCTION;; remove **Info:** When passing **remove** as payload, the function is removed and the constant period becomes active. For more information on the possible period functions see the **Predefined Functions** section.
/remaining	GET	Returns the remaining repetitions, if finite was specified at creation.
/running	GET	Returns the running status. **Values:** true: running false: not running
/running	PUT	Changes the running status. **Payload:** true: starts the periodic action from the beginning true;continue: continues the periodic action false: stopps the periodic action.

3.5 Predefined Functions

For the period function and the payload function some predefined functions are implemented for easy use. In addition, the user has the alternative to define his or her own period and payload functions using JavaScript.

Function	Use
Increaser	This function increases its value after each cycle. **Call:** inc;;START;;STEP;;END **Parameters:** START: The start value. STEP: The amount to increase. END: The end value. **Info:** When the end value is reached, the increaser wraps around and starts from the start again. The increaser can also work as a decreaser, when the start value is chosen larger than the end value and the step is a negative value.
Set	This function uses a set of values where one after the other is chosen. **Call:** set;;VALUE1;;VALUE2;;...;;VALUEn **Parameters:** VALUE1,...VALUEn: any number or string. **Info:** When the end of the set is reached, it starts over again at the beginning.

Own	The user can define any function in valid JavaScript code.
	
	Call: own;;FUNCTION;;
	
	Parameters: FUNCTION: Valid JavaScript code.
	
	Info: The value to return can directly be stored into **ret**, a predefined variable. When values need to be stored for later reuse, they can be put in a predefined array **storage**.
	No function declaration is required.
	
	Example: if (!storage[0]) {
	    storage[0] = 0;
	} else {
	    storage[0] = storage[0] + 10;
	    ret = storage[0];
	}

Chapter 4

Multicast

4.1 Description

This module multiplies a single request for a collection of target resources. A multicast task is created for multicasts, which are often used. A single multicast can be performed using a simpler mechanism where the target resources are passed with the request.

4.2 Setup

Initially the multicast module only contains the resource /**tasks**, which is still empty (Figure 4.1). All multicast tasks will be put into that resource one after the other (Figure 4.2).

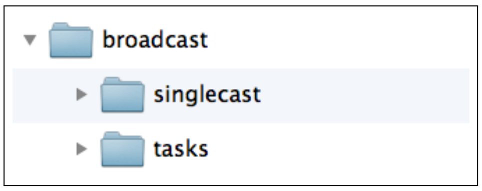


Figure 4.1: Initial Setup

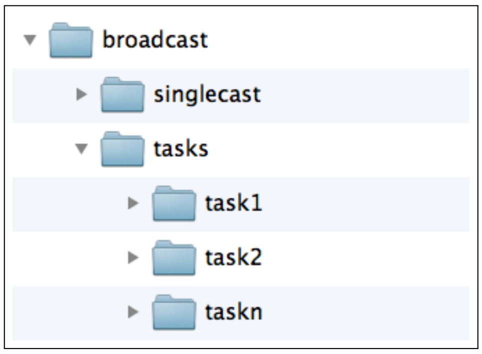


Figure 4.2: Tasks

4.3 Create new Task

In order to create a new task of a multicast a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter	Content
resid =	The resource identification for the new task. It will also be used as the name of the task.
targetX (many) =	A collection of targets, which become the target of the multicast. **Info:** The X is replaced by increasing numbers. **Example:** target1 = The first target. target2 = The second target. targetn = Up to n targets.

The created task instance offers a set of resources to access information about the task and interact with it (Figure 4.3).

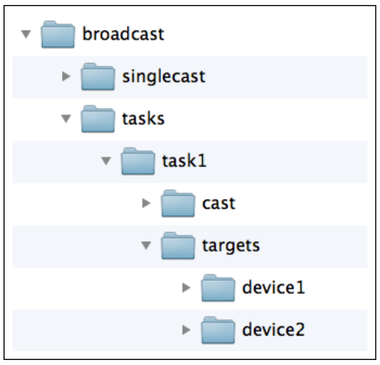


Figure 4.3: Multicast Task

4.4 Use

4.4.1 General Requests

General requests can always be performed, as long as the module is running.

URI: /multicast

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this module.
/tasks/task1	DELETE	Removes the task from the module.
/single	POST	The POST request will be multicast to the targets specified in the payload. **Payload:** target1 = The first target. target2 = The second target. targetn = Up to n targets. payload = The payload to be sent with the multicast. **Info:** The payload holds information about the targets and an optional payload.
/single	PUT	The PUT request will be multicast to the targets specified in the payload. **Payload:** target1 = The first target. target2 = The second target. targetn = Up to n targets. payload = The payload to be sent with the multicast. **Info:** The payload holds information about the targets and an optional payload.
/single	DELETE	The DELETE request will be multicast to the targets specified in the payload. **Payload:** target1 = The first target. target2 = The second target. targetn = Up to n targets. payload = The payload to be sent with the multicast. **Info:** The payload holds information about the targets and an optional payload.

4.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /broadcast/tasks/task1

URI	Request	Notes
/targets	GET	Returns a list of all target devices.
/targets/deviceX	GET	Returns the target device with the number X.
/cast	POST	The POST request is multicasted to the target devices. **Payload:** Any string. **Info:** The payload specified in the request is passed to the broadcast.
/cast	PUT	The PUT request is multicasted to target devices. **Payload:** Any string. **Info:** The payload specified in the request is passed to the broadcast.
/cast	DELETE	The DELETE request is multicasted to target devices. **Payload:** Any string. **Info:** The payload specified in the request is passed to the multicast.
/targetdecisions	GET	Returns the decision functions for all targets. **Values:** 1;DECISION_FUNCTION 2;DECISION_FUNCTION n;DECISION_FUNCTION **Info:** If there is no decision function specified for a target, DECISION_FUNCTION is the empty string.
/targetdecisions/ decisionX	GET	Returns the decision function, if specified.

URI	Request	Notes
/targetdecisions/ decisionX	PUT	Changes the decision function. **Payload:** equal;;NUMBER | STRING notequal;;NUMBER | STRING greater;;NUMBER greaterequal;;NUMBER less;;NUMBER lessequal;;NUMBER contains;;STRING prefix;;STRING suffix;;STRING own;;BOOLEAN_EXPRESSION;; remove **Info:** When passing **remove** as payload, the function is removed. For more information on the possible decision functions see the **Predefined Functions** section.

4.5 Predefined Functions

For the decision functions some predefined functions are implemented. In addition, the user has the alternative to define his or her own decision function using JavaScript.

Function	Use
Equal	The source value is checked to be equal to an other value. **Call:** equal;;VALUE **Parameters:** VALUE: any number or string. **Info:** The function checks for: incoming value == VALUE

Function	Use
Not Equal	The source value is checked to be not equal to an other value. **Call:** notequal;;VALUE **Parameters:** VALUE: any number or string. **Info:** The function checks for: incoming value $!=$ VALUE
Greater	The source value is checked to be greater than an other value. **Call:** greater;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value $>$ NUMBER
GreaterEqual	The source value is checked to be greater or equal than an other value. **Call:** greaterequal;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value $>=$ NUMBER
Less	The source value is checked to be less than an other value. **Call:** less;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value $<$ NUMBER

Function	Use
LessEqual	The source value is checked to be less or equal than an other value. **Call:** less equal;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value <= NUMBER
Contains	The source value is checked to contain a string value. **Call:** contains;;STRING **Parameters:** STRING: any string.
Prefix	The source value is checked to have a prefix of a string value. **Call:** contains;;STRING **Parameters:** prefix: any string.
Suffix	The source value is checked to have a suffix of a string value. **Call:** suffix;;STRING **Parameters:** STRING: any string.
Own	The user can specify a boolean expression in valid JavaScript code. **Call:** own;;BOOLEAN_EXPRESSION;; **Parameters:** BOOLEAN_EXPRESSION: Valid JavaScript code. **Info:** The incoming value can be accessed through a variable called **value**. **Example:** (value>0 && value<100) || (value==1000)

Chapter 5

Multiple Aggregate

5.1 Description

The multiple aggregate module can be used to aggregate values of a collection of source resources. The module registers as observer on each resource. When a new value arrives an aggregated value is computed. Some aggregate functions are predefined in the module, others can be defined by the user.

5.2 Setup

Initially the multiple aggregate module only contains the resource /**tasks**, which is still empty (Figure 5.1). All broadcast tasks will be put into that resource one after the other (Figure 5.2).

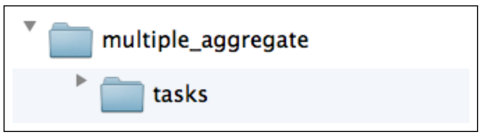


Figure 5.1: Initial Setup

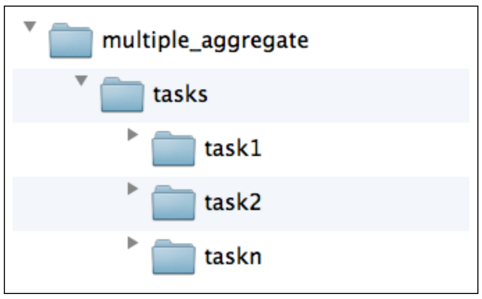


Figure 5.2: Tasks

5.3 Create new Task

In order to create a new task of a broadcast a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter		Content
resid	=	The resource identification for the new task. It will also be used as the name of the task.
sourceX (many)	=	A collection of sources to collect the data from. **Info:** The X is replaced by increasing numbers. **Example:** source1 = The first source. source2 = The second source. sourcen = Up to n sources.
aggregatefunc	=	The aggregation function aggregates the incoming values. **Values:** sum avg max min add subtract multiply divide modulo prefix suffix newest own;;FUNCTION;; **Info:** For more information on the possible aggregation functions see the **Predefined Functions** section.

The created task instance offers a set of resources to access information about the task and interact with it (Figure 5.3).

5.4 Use

5.4.1 General Requests

General requests can always be performed, as long as the module is running.

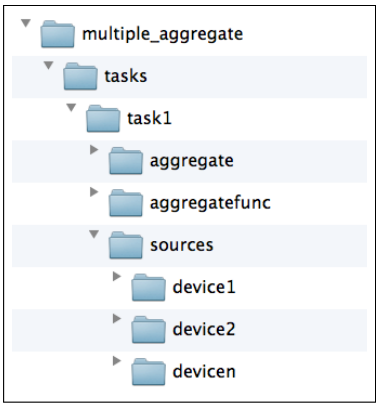


Figure 5.3: Multiple Aggregate Task

URI: /multiple_aggregate

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this module.
/tasks/task1	DELETE	Removes the task from the module.

5.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /broadcast/tasks/task1

URI	Request	Notes
/sources	GET	Returns a list of all sources the data is collected from.
/sources/deviceX	GET	Returns the source device with the number X.
/aggregate	GET	Returns the aggregated value computed from the incoming data. **Options:** withdevice=true (opt) **Values:** withdevice=true: VALUE;DEVICE_NR Returns the value and the device number of the last incoming data. **Info:** This resource is **observable**. Any application can register as observer.
/aggregatefunc	GET	Returns the aggregation function used to aggregate the values.

URI	Request	Notes
/aggregatefunc	PUT	Changes the aggregation function. **Payload:** sum avg max min add subtract multiply divide modulo prefix suffix newest own;;FUNCTION;; **Info:** Temporary data stored for an aggregated value is lost.

5.5 Predefined Functions

For the aggregation function some predefined functions are implemented. In addition, the user has the alternative to define his or her own aggregation function using JavaScript.

Function	Use
Sum	The sum of all incoming values is recorded. **Call:** sum
Avg	The average of all incoming values is recorded. **Call:** avg
Max	The maximum of all incoming values is recorded. **Call:** max
Min	The minimum of all incoming values is recorded. **Call:** min

Function	Use
Add	A specified value is added to each incoming source value. **Call:** add;;NUMBER **Parameters:** NUMBER: Any number.
Subract	A specified value is subtracted from each incoming source value. **Call:** subtract;;NUMBER **Parameters:** NUMBER: Any number.
Multiply	Each incoming source value is multiplied with a specified value. **Call:** multiply;;NUMBER **Parameters:** NUMBER: Any number.
Divide	Each incoming source value is divided by a specified value. **Call:** divide;;NUMBER **Parameters:** NUMBER: Any number.
Modulo	For each incoming source value modulo of a specified value is commuted. **Call:** modulo;;NUMBER **Parameters:** NUMBER: Any number.
Prefix	A prefix is appended at the beginning of every incoming source value. **Call:** prefix;;STRING **Parameters:** STRING: Any string.
Suffix	A suffix is appended at the end of every incoming source value. **Call:** suffix;;STRING **Parameters:** STRING: Any string.

Function	Use
Newest	Every new incoming value is being recorded. **Call:** newest

Own	The user can define any function in valid JavaScript code.
	
	Call: own;;FUNCTION;;
	
	Parameters: FUNCTION: Valid JavaScript code.
	
	Info: The value is passed through a variable called **value**. With every value, its source device can be accessed through the variable **device**. The result to return can directly be stored into **ret**, a predefined variable. When values need to be stored for later reuse, they can be put in a predefined array **storage**. No function declaration is required.
	
	Example:
	`if (!storage[0] && storage[1]) {`
	`    storage[0] = 0;`
	`    storage[1] = 0;`
	`}`
	`if (device==1) {`
	`    storage[0] += value;`
	`    ret = storage[0];`
	`} else if (device==2) {`
	`    storage[1] += value;`
	`    ret = storage[1];`
	`}`

Chapter 6

Control loop

6.1 Description

The conrol loop module implements a simple parametrizable control loop. The value is read from one source resource, altered to a new value, and finally sent to a target resource. For the altering mechanism a set of predefined functions are available. In addition the user can choose to set a decision function which allows simple if-then-else branching for the modification.

6.2 Setup

Initially the control loop module only contains the resource **tasks**, which is still empty (Figure 6.1). All control loop tasks will be put into that resource one after the other (Figure 6.2).

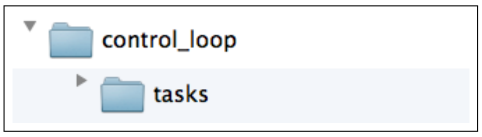


Figure 6.1: Initial Setup

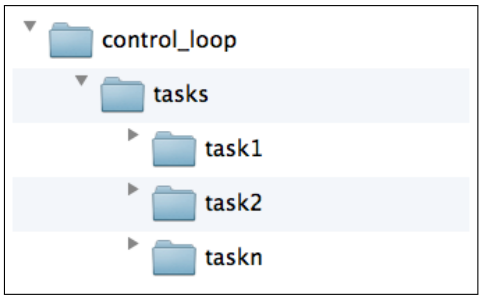


Figure 6.2: Tasks

6.3 Create new Task

In order to create a new task of a control loop a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter		Content
resid	=	The resource identification for the new task. It will also be used as the name of the task.
source	=	The source resource, where the value is read from.
target	=	The target resource, where the modified value is sent to.
targetoperation	=	The operation used for the request on the target resource. **Values:** PUT | POST | DELETE
decisionfunc (opt)	=	The decision function is used to create a possible branch for the modification function. **Values:** equal;;NUMBER | STRING notequal;;NUMBER | STRING greater;;NUMBER greaterequal;;NUMBER less;;NUMBER lessequal;;NUMBER contains;;STRING prefix;;STRING suffix;;STRING own;;BOOLEAN_EXPRESSION;; **Info:** With the decision function, both a simple modification function or a modification function plus a else-modification function can be defined. In the first case only those values passing the decision function will be sent to the target. The others are lost. For more information on the possible decision functions see the **Predefined Functions** section.

Parameter	Content
modifyfunc =	The modification function is used to modify the incoming value from the source. **Values:** sum avg max min add;;NUMBER subtract;;NUMBER multiply;;NUMBER divide;;NUMBER modulo;;NUMBER prefix;;STRING suffix;;STRING own;;FUNCTION;; **Info:** For more information on the possible modification functions see the **Predefined Functions** section.
modifyfuncelse = (opt)	The else-modification function is used for the else branch of the decision function. It modifies the value the same way as the modify function does. **Values:** sum avg max min add;;NUMBER subtract;;NUMBER multiply;;NUMBER divide;;NUMBER modulo;;NUMBER prefix;;STRING suffix;;STRING own;;FUNCTION;; **Info:** The modification function for the else branch of the decision function can only be defined if a decision function is defined. For more information on the possible else-modification functions see the **Predefined Functions** section.

The created task instance offers a set of resources to access information about the task and interact with it (Figure 6.3).

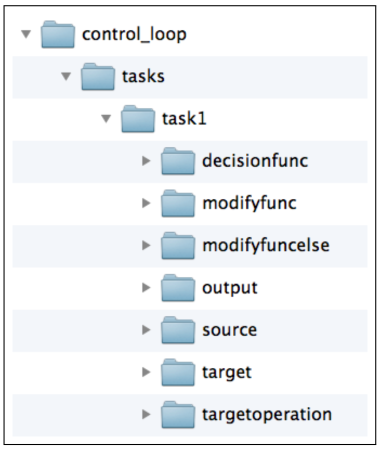


Figure 6.3: control loop Task

6.4 Use

6.4.1 General Requests

General requests can always be performed, as long as the module is running.

URI: /control_loop

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this module.
/tasks/task1	DELETE	Removes the task from the module.

6.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /control_loop/tasks/task1

URI	Request	Notes
/source	GET	Returns the source resource for the control loop.
/target	GET	Returns the target resource for the control loop.

URI	Request	Notes
/targetoperation	GET	Returns the operation for the request on the target. **Values:** PUT | POST | DELETE
/output	GET	Returns the control loop output after the modification. **Info:** This resource is **observable**. Any application can register as observer.
/decisionfunc	GET	Returns the decision function, if specified at creation.
/decisionfunc	PUT	Changes the decision function **Payload:** equal;;NUMBER | STRING notequal;;NUMBER | STRING greater;;NUMBER greaterequal;;NUMBER less;;NUMBER lessequal;;NUMBER contains;;STRING prefix;;STRING suffix;;STRING own;;BOOLEAN_EXPRESSION;; remove **Info:** When passing **remove** as payload, the function is removed. For more information on the possible decision functions see the **Predefined Functions** section.
/modifyfunc	GET	Returns the modification function.

URI	Request	Notes
/modifyfunc	PUT	Changes the modification function. **Payload:** sum avg max min add;;NUMBER subtract;;NUMBER multiply;;NUMBER divide;;NUMBER modulo;;NUMBER prefix;;STRING suffix;;STRING own;;FUNCTION;; remove **Info:** When passing **remove** as payload, the function is removed. For more information on the possible modification functions see the **Predefined Functions** section.
/modifyfuncelse	GET	Returns the modification function of the else branch, if specified at creation.

URI	Request	Notes
/modifyfuncelse	PUT	Changes the modification function for the else branch. **Payload:** sum avg max min add;;NUMBER subtract;;NUMBER multiply;;NUMBER divide;;NUMBER modulo;;NUMBER prefix;;STRING suffix;;STRING own;;FUNCTION;; remove **Info:** When passing **remove** as payload, the function is removed. Changing the modification function for the else branch only has an effect if the decision function is defined. Otherwise the created modification function will just be ignored. For more information on the possible else-modification functions see the **Predefined Functions** section.

6.5 Predefined Functions

The control loop module defines a large set of predefined functions. Different functions serve as decision functions or modification functions.

6.5.1 Decision Functions

The alternative for the decision function are implementations of boolean expressions. In addition the user has the alternative to define his or her own decision function using JavaScript.

Function	Use
Equal	The source value is checked to be equal to an other value. **Call:** equal;;VALUE **Parameters:** VALUE: any number or string. **Info:** The function checks for: incoming value == VALUE
Not Equal	The source value is checked to be not equal to an other value. **Call:** notequal;;VALUE **Parameters:** VALUE: any number or string. **Info:** The function checks for: incoming value ! = VALUE
Greater	The source value is checked to be greater than an other value. **Call:** greater;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value > NUMBER
GreaterEqual	The source value is checked to be greater or equal than an other value. **Call:** greaterequal;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value >= NUMBER

Function	Use
Less	The source value is checked to be less than an other value. **Call:** less;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value $<$ NUMBER
LessEqual	The source value is checked to be less or equal than an other value. **Call:** less equal;;NUMBER **Parameters:** NUMBER: any number. **Info:** The function checks for: incoming value $<=$ NUMBER
Contains	The source value is checked to contain a string value. **Call:** contains;;STRING **Parameters:** STRING: any string.
Prefix	The source value is checked to have a prefix of a string value. **Call:** contains;;STRING **Parameters:** prefix: any string.
Suffix	The source value is checked to have a suffix of a string value. **Call:** suffix;;STRING **Parameters:** STRING: any string.

Function	Use
Own	The user can specify a boolean expression in valid JavaScript code. **Call:** own;;BOOLEAN_EXPRESSION;; **Parameters:** BOOLEAN_EXPRESSION: Valid JavaScript code. **Info:** The incoming value can be accessed through a variable called **value**. **Example:** (value>0 && value<100) || (value==1000)

6.5.2 Modification Functions

The modification functions take the incoming value and alter it. In addition, the user has the alternative to define his or her own decision function using JavaScript.

Function	Use
Sum	The incoming source value is summed up. **Call:** sum **Info:** After changing the modification function the temporary sum is lost.
Average	The average of all incoming values is calculated. **Call:** avg **Info:** After changing the modification function the temporary average is lost.
Maximum	The maximum of all incoming values is returned. **Call:** max **Info:** After changing the modification function the temporary maximum is lost.

Function	Use
Minimum	The minimum of all incoming values is returned. **Call:** min **Info:** After changing the modification function the temporary minimum is lost.
Add	A specified value is added to each incoming source value. **Call:** add;;NUMBER **Parameters:** NUMBER: Any number.
Subract	A specified value is subtracted from each incoming source value. **Call:** subtract;;NUMBER **Parameters:** NUMBER: Any number.
Multiply	Each incoming source value is multiplied with a specified value. **Call:** multiply;;NUMBER **Parameters:** NUMBER: Any number.
Divide	Each incoming source value is divided by a specified value. **Call:** divide;;NUMBER **Parameters:** NUMBER: Any number.
Modulo	For each incoming source value modulo of a specified value is commuted. **Call:** modulo;;NUMBER **Parameters:** NUMBER: Any number.
Prefix	A prefix is appended at the beginning of every incoming source value. **Call:** prefix;;STRING **Parameters:** STRING: Any string.

Function	Use
Suffix	A suffix is appended at the end of every incoming source value. **Call:** suffix;;STRING **Parameters:** STRING: Any string.
None	The incoming value is not changed at all. **Call:** none

Own	The user can define any function in valid JavaScript code.
	
	Call: own;;FUNCTION;;
	
	Parameters: FUNCTION: Valid JavaScript code.
	
	Info: The incoming value can be accessed through a variable called **value**. The result to return can directly be stored into **ret**, a predefined variable. When values need to be stored for later reuse, they can be put in a predefined array **storage**.
	No function declaration is required.
	
	Example:
	if (!storage[0]) {
	    storage[0] = 0;
	}
	storage[0] += value;
	ret = storage[0];

Chapter 7

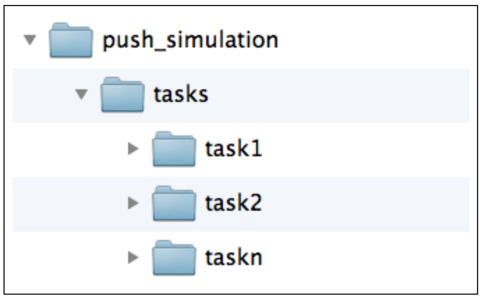
Push Simulation

7.1 Description

This app can be used to make a non-observable resource observable. A push simulation task polls on the resource and offers a observable resource to register. Only changed values are pushed to the observing applications.

7.2 Setup

Initially the push simulation only contains the resource /**tasks**, which is still empty (Figure 7.1). All push simulation tasks will be put into that resource one after the other (Figure 7.2). s



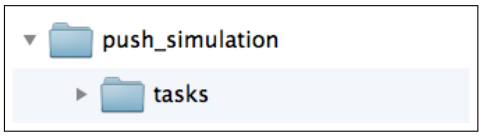


Figure 7.1: Initial Setup

Figure 7.2: Tasks

7.3 Create new Task

In order to create a new task of a push simulation a **POST request** on the /**tasks** resource has to be performed. The POST request requires a specific payload:

Parameter		Content
resid	=	The resource identification for the new task. It will also be used as the name of the task.
source	=	The source device, which is not observable.
poll (opt)	=	The polling rate to retrieve data from the device. **Info:** The **default** value, if not specified is **1000 ms**.
options (opt)	=	The options to be sent with the GET request. **Values:** OPTION1=OPT1&...&OPTIONn=OPTn

The created task instance provides an interface for access to information about it and interaction with it (Figure 7.3).

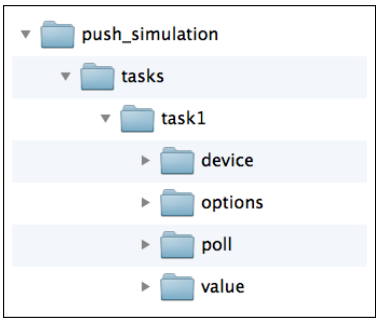


Figure 7.3: Push Simulation Task

7.4 Use

7.4.1 General Requests

General requests can always be performed, as long as the app is running.

URI: /push_simulation

URI	Request	Notes
/tasks	GET	Returns the list of tasks running on this app.
/tasks/task1	DELETE	Removes the task from the app.

7.4.2 Task Requests

Requests that can be executed on created task instances.

URI: /timed_requests/tasks/task1

URI	Request	Notes
/source	GET	Returns the source resource, which is not observable.
/options	GET	Returns the options for the polling GET request on the source.
/poll	GET	Returns the polling interval used to fetch the data from the source.
/poll	PUT	Changes the polling interval. **Payload:** Any number (ms)
/value	GET	Returns the value fetched from the source resource. This resource is used to simulate the observation mechanism. **Info:** This resource is **observable**. Any application can register as observer.