TSC Schema
Revision 0.42 – Draft Issue

Introduction

Note: Currently TSC is not an adopted xAP specification. Once adopted then messages will change from TSC.<type> to xAPTSC.<type>, first adopters should be aware of this.

The xAP TSC schema is designed for the reporting and control of 'telemetry' or sensor types of devices. Examples include weather sensors, temperature sensors (thermistors, thermostats, one-wire, etc), pressure sensors, flow sensors, watt hour meters, etc.

The schema allows devices to be discovered, including to some extent their capabilities, to report their status and, for those that have the facility, to be controlled.

Control can be over devices such as heating, pumps, radiators, where the xAP BSC schema does not provide the control required.

The primary difference between BSC and TSC is that with TSC devices the status and control uses real world units – a temperature sensor reports its value in Celsius (or Fahrenheit or Kelvin) and the units it is using are included in the message so there is no ambiguity. In addition, control is extendable to include such things as high and low limits/alarms, setpoints, etc.

Not all devices will be able to be calibrated against a recognised standard. As an example, an LDR(light dependent resistor) may be used to measure outside light levels but to calibrate it in Lumens is too onerous. The 'Arbitrary' unit can then be used and the consuming xAP application knows, for example, that at a value below 100 *for this specific device* it should turn on the lights (or whatever).

TSC has been designed to cater for both centralised and distributed device and system topologies. For example, a 'thermostat' on a wall in a room may simply be a temperature sensor, such as a one-wire DS18B20, or it could be a sophisticated unit with temperature and humidity sensors and user interaction controlling temperature and humidity setpoints by hour by day and logging maximum and minimum values. Whilst the latter has significant capabilities, these do not have to be used and a central application could ignore or override them. Better still, TSC allows both user interaction and centrally updated control.

TSC is not designed for binary (on/off) devices, again BSC provides an adequate method of control and reporting. And a device which has TSC endpoints may also have a BSC endpoint for on/off control. An example being a complete home heating system.

Whilst many real world measurement and control (in the Home Automation realm anyway) have been catered for, the schemas are readily extensible.

Note: There is a current plan to increase the UID length and this would affect the specification, simplifying some areas.

Device Types

All TSC devices are considered as complex, the specific details of which will depend on the implementation and the stimuli being measured or controlled. Even simple devices such as a temperature sensor may respond to a command, for example to change a low limit or high limit variable. Other devices may fall into the 'black box' category with a number of measurement values being reported and, potentially, have a number of outputs.

Addressing

Each device will have one or more endpoints (examples are temperature sensor, voltmeter, valve, etc). Each endpoint will have associated with it a sub-address which meets the xAP specification and a unique device ID. The range of device ID's is 01 to FE (subject to being extended in the future). Where the device is generating a message (a source device) then the UID will be mapped to the device ID with a one to one relationship. This mapping must persist past restarts/power cycling of the device and not be a dynamic assignment/re-assignment.

The device must support a level of wildcard addressing such that device discovery can take place and as a minimum this should be *.*.*:> and >:>

It is recommended that a logical approach is taken to sub-addresses such that the flexibility inherent in xAP is available. For example a device with multiple sensors should support interrogation by location or by sensor/control type using subaddresses such as:

```
acme.controller.instance:outdoor.temperature
acme.controller.instance:outdoor.humidity
acme.controller.instance:outdoor.sunsensor
acme.controller.instance:indoor.temperature
acme.controller.instance:indoor.humidity
```

However, any legal sub-addressing scheme may be used since the context of the message is determined in the body.

Schema

There are five classes which relate to the TSC schema, these are.

TSC.cmd
TSC.query
TSC.event
TSC.info
TSC.capability

The latter has no equivalent in xAP BSC.

TSC.cmd

Messages of this class are used to control a device, which may either alter the state of the device or update a process variable in the device.

The class= header field must be set to TSC.cmd by the sending application – i.e.
`class=TSC.cmd`

A target must be included in the header field. This must address an endpoint or endpoints. It is specifically forbidden to target a TSC.cmd message at the base device.

Each TSC.cmd message will consist of a header and one or more body sections. Each body section will be titled `cmd.<sensor/control type>`, where <sensor/control type> is the type of device endpoint.

The <sensor/control type> is used to indicate the context of the section following the title. See the section on sensor and control types.

The body section contains a number of key/value pairs appropriate to the device being controlled or the process variable being changed. For each <sensor/control type> there are some mandatory key/value pairs and other optional items.

One mandatory pair for all `cmd` body sections is that of the Device ID. This is a two digit hexadecimal field. The form of this pair is ID=xx, example:
`ID=5A`

A device must check that the Device ID corresponds to the sub-address in the header target field and that the context is correct for the endpoint type. If these tests fail then no further action should be taken.

TSC does not support wildcarded Device ID's for simplicity and safety.

If the `TSC.cmd` message causes a change in the output state of the endpoint, either by direct action or by the change of a process variable, then a `TSC.event` message must be sent. If the `TSC.cmd` message does not change the endpoint status and that pair/value(s) is supported then a `TSC.info` message must be sent.

If all of the name/value pairs are not supported then no `TSC.info` message should be sent. Where only a subset is supported then the `TSC.info` message should only reflect those pairs supported.

The following are examples of `TSC.cmd` messages:

Example:
```
{
v=12
hop=1
```

```
UID=FF123400
class=TSC.cmd
source=acme.controller.instance1
target=johndoe.dcpowersuppy.instance1:1
}
cmd.voltage
{
ID=01
unit=v
value=22.5
}
```

The above would change the current value of the power supply to 22.5V

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=acme.heatingcontroller.instance1
target=acme.radiatorcontrol.instance1:>
}
cmd.position
{
ID=01
unit=Arb
value=50
}
cmd.position
{
ID=02
unit=Arb
value=50
}
```

This would cause the radiator valves (as ID=01 and 02) linked to the upperfloor controller to go to a position determined by the endpoint. It may be that this is 50% open, 50 turns in or 50% PWM or some other position. Clearly the sending application must have knowledge of what the action expected is of this command.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=acme.heatingcontroller.instance1
target=acme.radiatorcontrol.instance1:1
}
cmd.rpm
```

```
{
ID=01
unit=rpm
value=800
}
```

Since from the previous example Device ID=01 is a positional endpoint then this command will cause no action and no response message

Setpoints

Many TSC devices will support a setpoint, i.e. a target value for the device to maintain and could be an input (such as a temperature measuring device) or an output (such as a tank heater).

Devices are controlled through the use of a `setpoint.<device>`

The minimum value pairs required are:

ID=(endpoint of device to be updated)
---mandatory
unit=(measurement unit)
---mandatory
value=(value of the setpoint)
---mandatory
apply=(timeframe to apply setpoint, 'current' is minimum to be supported. Device specific
---mandatory

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=acme.heatingcontroller.instance1
target=acme.thermostat.bedroom:2
}
setpoint.humidity
{
ID=02
unit=RH
value=80
apply=current
}
```

The thermostat will therefore update the current RH setpoint value to 80% RH.

Example:

```
{
```

```
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=acme.heatingcontroller.instance1
target=acme.thermostat.bedroom:1
}
setpoint.temperature
{
ID=01
unit=C
value=16
apply=night
}
```

The above sets the device specific 'night' setpoint to 18C and does not affect the current setpoint unless the device is currently in 'night' mode. Note: A device does not need to support more than one setpoint.

Resets and counters

Within a device there may be cumulative counters which can be reset, as well as non-resettable counters. This is device specific and an optional feature. A counter is an endpoint or endpoints within a device.

To reset a counter then an `TSC.cmd` message is sent with body of the form `reset`

The response is to send an `TSC.info` message.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=mi4.floorplan.instance1
target=acme.rainguage.shed:counter.level
}
reset
{
ID=9A
}
```

The above would reset the cumulative level counter in the rainguage device.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
```

```
source=mi4.floorplan.instance1
target=acme.weatherstation.shed:counter.*
}
reset
{
}
```

The above would reset all counters in the weatherstation.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.cmd
source=mi4.floorplan.instance1
target=acme.watthour.offpeak:counter.energy
}
reset
{
}
```

The device should record the date and time of the reset if possible for reporting within the `stats` body section. If this is not possible then the device may record the elapsed seconds and report this.

TSC.query

Messages of this class are designed to elicit a status response, in the form of an TSC.info message, from a device. This message has two explicit purposes – for device and capability discovery and for getting device status.

The class= header field must be set to TSC.query by the sending application – i.e. `class=TSC.query`

A target must be included in the header field. This can include the base device or be targeted at an endpoint.

The message must include one or more body sections. Unlike xAP BSC, the title of the body section is significant. This can either consist of 'request.all', 'request.capability' or 'request'.<sensor/control type> For example:

```
request.voltage
```

The body section(s) are empty and are to be ignored.

Where the title is `request.<sensor/control type>` then the device should respond with TSC.info messages for each endpoint which match the target address and have the matching context. For example:

Where the title is `request.all` then the device should respond with TSC.info messages for each endpoint which match the target address.

Where the title is `request.capability` then the device should respond with TSC.capability messages for each endpoint which match the target address.

Example:

```
{
v=12
hop=1
UID=FF456200
class=TSC.query
source=mi4.floor.instance1
target=>:>
}
request.humidity
{
}
```

This is a request for all xAP TSC devices on the network which have endpoints that support humidity to respond with TSC.info messages.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.query
source=acme.heatingcontroller.instance1
target=acme.thermostat.*
}
request.temperature
{
}
```

This is a request to all acme.thermostat devices to report their endpoints that support temperature.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.query
source=acme.heatingcontroller.instance1
target=acme.thermostat.*
}
request.all
{
}
```

The above is a request to all devices to report all their endpoints and their values.

Example:

```
{
v=12
hop=1
UID=FF123400
class=TSC.query
source=acme.heatingcontroller.instance1
target=acme.thermostat.*
}
request.capability
{
}
```

The above is a special format to allow device discovery.

TSC.info

Devices use messages of this class to report the current value of process and/or input values. `TSC.info` messages may be sent periodically, in response to a `TSC.query` message and on device startup. It is recommended that devices support all three.

A device with multiple subdevices/endpoints will send multiple TSC.info messages, one for each subdevice.

`TSC.info` messages do not contain a target address in the message header.

Example:

```
{
v=12
hop=1
UID=FF756601
class=TSC.info
source=acme.watthour.offpeak:1
}
info.energy
{
name=offpeak meter
unit=kwh
value=10.5
DisplayText=10.5kWh Offpeak
delta=0.5
}
stats.energy
{
period=2147
```

```
}
```

This is information from a simple watt hour meter on the off peak supply showing that 10.5 kWh has been used since the counter was reset 2147 seconds ago - this device does not have date and time capabilities. The value change to trigger a message is 0.5 kWh. An optional friendly name and display message is included.

Example:

```
xap-header
{
v=12
hop=1
UID=FF456201
class=TSC.info
source=acme.thermostat.bathroom:1
}
info.temperature
{
name=main bathroom stat
datetime=20060830194500
unit=c
value=22
setpoint=21
}
stats.temperature
{
max=23
maxtime=20060830194500
min=-3
maxtime=20060830034500
}
```

Example:

```
xap-header
{
v=12
hop=1
UID=FF456202
class=TSC.info
source=acme.thermostat.bathroom:2
}
info.humidity
{
name=main bathroom humidistat
datetime=20060830194500
unit=RH
value=72
setpoint=70
}
stats.humidity
```

```
{
max=100
maxtime=20060830074500
min=45
mintime=20060830124600
resettime=20060830000001
}
```

This shows two messages which were either generated as a result of an interval or in response to a TSC.query command that targeted both (or all) endpoints in the device.

Alarms

Many process control systems require the facility to set an alarm. An example may be an oil tank low sensor that reports an alarm when it reaches 10% full. These would be handled in the normal way with an `TSC.event` class message, with a separate `alarmhi.<type>` / `alarmlo.<type>` body containing the alarm parameter causing the alarm. The units in the alarm section must be the same as the event or info section.

The alarm body section cannot be the only body section and must accompany either an info or an event body.

Example:

```
{
v=12
hop=1
UID=FF456402
class=TSC.event
source=acme.tanksensor.oiltank:1
}
event.level
{
datetime=20060930194500
unit=pc
value=9
DisplayText=Tank Low
}
alarmhi.level
{
unit=pc
value=10
}
```

It is, of course, acceptable for a consuming application to handle the raising of alarms and for this not to be sensor generated. Equally, the sensor may generate other xAP class messages to cause a display on networked devices as well as the `DisplayText` value pair. All are implementation details which will depend on the design of the overall automation system, and may change over time as the system gets more complex.

TSC.event

Messages of this class are used when a process and/or input value changes or a device output status changes. The intention is that whenever some action (internal or external) takes place that this is reported in a TSC.event message:

- When a `TSC.cmd` message causes an output to change
- When a `TSC.cmd` message changes a process variable, such as a setpoint or alarm, which then changes the status of the endpoint. For example, for a 'thermostat' type device if the current temperature is 20C and the current setpoint is 22C and a new setpoint of 18C is received then a `TSC.event` message should be sent since the status has changed.
- When an external action causes an output status or process variable to change the status of an endpoint. For example, the user changes the thermostat setpoint on the device.
- When another xAP (or other – e.g. xPL, EIB, etc) schema changes the output status or the status of an endpoint process variable.

`TSC.event` messages do not contain a target address in the message header.

Where a status/stimulii is changing rapidly or by small increments then it is up to the developer to either decide upon the interval or delta value change on which to generate the `TSC.event` message or to support the use of process variables which define the interval and/or delta value. Even if an `TSC.event` message has not been sent but the value has changed, then a `TSC.event` message must be sent in response to the `TSC.query`.

The body consists of a single `event.<sensor/control type>` titled section.

If the device has the capability it should include the date and time of the event within the event section as a `DateTime=` value.

It may also contain a `DisplayText` value which may be used for display purposes and a `Name=` friendly name. Note: The values in `DisplayText=` must not be used as process values.

Example:

```
{
v=12
hop=1
UID=FF456202
class=TSC.event
source=acme.thermostat.bathroom:2
}
event.humidity
{
name=main bathroom humidistat
datetime=20060830194500
unit=RH
```

```
value=72
setpoint=70
}
stats.humidity
{
max=100
maxtime=20060830074500
min=45
mintime=20060830124600
resettime=20060830000001
}
```

This is the humidity endpoint within the bathroom thermostat device. It has a device ID/Sub UID of 02. It is reporting humidity as RH percentage and the current value is 72% RH. The setpoint is 70% RH. The maximum humidity value has been 100% RH at 07:45 on 30th August 2006 and the minimum 45% RH at 12:46 on the same day, both figures since the reset at midnight (00:00:01).

TSC.capability

Without the ability for a device to advertise the type and range of acceptable values controlling applications will be unable to utilise them without considerable configuration.

Within BSC there is a simple method for determining the maximum level a device can control but with the complexity of TSC this is not such a simple task. Consider a boiler thermostat – it may send temperature values back across a range from, say -20 to 130 C but the setpoint may never go above 95C and the alarm maximum safe value may be 110C, so a single range for a device or endpoint is not acceptable.

What is required is that a device can report on all endpoints. Reporting in `TSC.info` messages would not give the full range of available variables, for example alarmhi/lo.

The TSC capability class consists of a header message and one or more body messages and is generated in response to a `TSC.request` class with a `request.capability` body. The minimum body section is `capability.<device>`.

There then follows optional capability messages of the form `<capability type>.<device>`. The current list of capability types is:

alarmlo
alarmhi
stats

This list is expandable by the implementer of a device.

Example:

```
{
v=12
hop=1
UID=FF456400
class=TSC.capability
source=acme.tanksensor.oiltank
}
capability.level
{
ID=01
type=input
unit=pc
maxvalue=100
minvalue=0
}
alarmhi.level
{
ID=01
unit=pc
maxvalue=100
minvalue=0
}
alarmlo.level
{
ID=01
unit=pc
maxvalue=100
minvalue=0
}
capability.temperature
{
ID=02
type=input
unit=c
maxvalue=450
minvalue=-50
}
alarmhi.temperature
{
ID=02
unit=c
maxvalue=150
minvalue=50
}
```

From the message above a controller knows that the tanksensor supports two sensor types, one for level and one for temperature, and the maximum and minimum values possible for each. For the level sensor there are two alarms supported one high and one low and the maximum and minimum values reported. For the temperature sensor only a high temperature alarm is supported, again with allowable maximum and minimum configuration values.

Example:

```
{
v=12
hop=1
UID=FF456600
class=TSC.capability
source=acme.tankstat.boiler
}
capability.temperature
{
ID=01
type=input
unit=C
maxvalue=130
minvalue=-20
setpointmax=95
}
alarmhi.level
{
ID=01
unit=c
maxvalue=110
minvalue=45
}
alarmlo.level
{
ID=01
unit=c
maxvalue=30
minvalue=0
}
```

The above matches the boiler thermostat example from earlier.

Finally, an example of an output.

Example:

```
{
v=12
hop=1
UID=FF12E400
class=TSC.capability
source=johndoe.dcpowersuppy.instance1
}
capability.voltage
{
ID=01
type=output
unit=v
minvalue=0
```

```
maxvalue=30
}
capability.current
{
ID=02
type=output
unit=a
minvalue=0
maxvalue=2
}
alarmhi.current
{
ID=02
unit=a
maxvalue=2
}
```

Sensor and Control Types-Context

Central to the philosophy of TSC is the standardisation of reporting. The xAP BSC schema (and others) have been used to report sensor statuses but there is no common agreement on units and values.

Within the body of each TSC message is the information regarding the units of measurement or control and the section title indicates the endpoint type.

The table below shows the standardised input types and the recommended units.

Note: This table is subject to revision outside of the TSC specification.

Measurement	**Section Title**	**Measurement Units**	**Abbreviation**
Temperature	temperature	Celsius or Fahrenheit or Kelvin	c, f or k
Humidity	humidity	RH%	rh
Voltage	voltage	Volts	v
Current	current	Amps	a
Resistance	resistance	Ohms	ohm
Energy	energy	Joules per second or kWh	js or kwh
Flow	flow	lps	lps
Pressure	pressure	Pascal or mBar	pa or mbar
Rainfall	rainfall	mm per hour	mmph
Power	power	Joules or Watts	j or w
Brightness	illuminance	Lumens	lm
Distance	distance	millimetre or metre or foot	mm or m or ft
Weight	weight	kilogram or pound	kg or lb

Volume	volume	litre, cubic metre, cu.ft	l, m3, cuft
Level	level	%, or linear measurement	pc, mm, m, ft
Position	position	degrees or rel%	deg, pc
Count	count	counts	cnt
Rotational speed	rpm	rpm	rpm
Linear speed	speed	mps (ms^{-1}) or fps	mps or fps
Percentage (generic)	percent	%	pc
Any of the above	Any of the above	Arbritary	arb

The Arbritary unit is used to represent values where either it is not possible to provide calibration or where there is no appropriate unit for the control or measurement.

Values are represented as signed numeric values, with or without a decimal point and to any precision required. The following are all valid values:
value=0
value=0.00
value=-1
value=201131.212121
value=-0.1

The following are not valid:

value=+2
value=+.44444
value=1.133E33
value=-.1
value=undefined
value=unknown

Where a value is not known then it should be sent as ?, e.g.

value=?

Other value/pairs are dependent on the specific sensor/control type.

The following in an incomplete list:

setpoint
max
min
avg
interval
delta
validity
reset
apply
duration
schedule

alarmlo
alarmhi

Datetimes

The normal format of dates and times in xAP is of the form YYYYMMDDHHMMSS. For sensors a finer control may be required and the format may, at the devices discretion, use the form YYYYMMDDHHMMSSsss. If present the milliseconds must be to three digits. All consuming applications should be able to handle either format by inspecting the length.

Note: Since the datetime value is locally produced all xAP devices should try and remain closely synchronised, otherwise the value of the data is degraded.

Not all devices are capable of supporting date and time and therefore would not include the datetime values in any message, rather than `datetime=?`, which is the value which may be sent by a device if the real time clock is not set.

Periods

Where a device does not support datetime then it may support periods, for the basis of counters. Periods are always in whole seconds. Example

```
period=3600
```

TSC.cmd

Change output value (output device)

class=tsc.cmd

cmd.<type>
---mandatory section, <type> is the control type for the device
{
ID=(01 to FE)
---mandatory endpoint identifier
unit=(unit designation)
---mandatory field which defines the output value units
value=(value of the command)
---mandatory output value
}

Change device setpoint

Device specific, devices are not required to support setpoints.

class=tsc.cmd

setpoint.<type>
---mandatory section, <type> is the control type for the device
{
ID=(01 to FE, endpoint subaddress)
---mandatory endpoint identifier
unit=(unit designation)
---mandatory
value=(value of the command)
---mandatory
apply=(device specific setpoint type, for example – current, day, night, occupied, etc)
---mandatory, device must support current as a minimum
}

Change device alarm values

Device specific, devices are not required to support alarms.

class=tsc.cmd

alarmhi.<type> / alarmlo.<type>
---mandatory section, <type> is the control type for the device
{
ID=(01 to FE, endpoint subaddress)
---mandatory endpoint identifier
unit=(unit designation)
---mandatory field which defines the value
value=(value of the alarm threshold)
---mandatory value
}

Reset Counters

Device specific, devices are not required to support counters.

class=tsc.cmd

reset
---mandatory section header
{
ID=(01 to FE, endpoint subaddress)
---optional endpoint identifier, if missing then all counters matching the target address are reset
}

TSC.info

class=tsc.info

info.<type>
---mandatory header, <type> defines the reported measurement type
{
unit=(unit designation)
---mandatory
value=(value of the command)
---mandatory
name=(friendly name of endpoint)
---optional
displaytext=(display message relating to the measurement)
---optional
datetime=(xAP or enhanced xAP datetime value when measurement took place)
---optional
setpoint=(value of the currently applicable setpoint in the same units as 'value')
---optional
alarmhi=(value of the high value alarm in the same units as 'value')
---optional
alarmlo=(value of the low value alarm in the same units as 'value')
---optional
delta=(unit of change required for device to generate an event in the same units as 'value')
---optional
}
stats.<type>
---optional header, <type> as per previous section
{
max=(maximum value reached, units as previous section)
---optional
maxtime=(xAP or enhanced xAP datetime value)
---optional
min=(minimum value reached, units as previous section)
---optional
mintime=(xAP or enhanced xAP datetime value)
---optional
resettime=(xAP or enhanced xAP datetime value of last statistics reset)
---optional, exclusive with period
period=(value, in whole seconds, since statistics were reset)
---optional, exclusive with resettime
}
alarmhi.<type> / alarmlo.<type>
---optional header, the presence of which specifies an alarm situation
{
unit=(unit designation)
---mandatory
value=(value of the alarm threshold)
---mandatory
}

TSC.event

class=tsc.event

event.<type>
---mandatory header, <type> defines the reported measurement type
{
unit=(unit designation)
---mandatory field which defines the value
value=(value of the command)
---mandatory value
name=(friendly name of endpoint)
---optional
displaytext=(display message relating to the measurement)
---optional
datetime=(xAP or enhanced xAP datetime value when measurement took place)
---optional
setpoint=(value of the currently applicable setpoint in the same units as 'value')
---optional
alarmhi=(value of the high value alarm in the same units as 'value')
---optional
alarmlo=(value of the low value alarm in the same units as 'value')
---optional
delta=(unit of change required for device to generate an event in the same units as 'value')
---optional
}
stats.<type>
---optional header, <type> as per previous section
{
max=(maximum value reached, units as previous section)
---optional
maxtime=(xAP or enhanced xAP datetime value)
---optional
min=(minimum value reached, units as previous section)
---optional
mintime=(xAP or enhanced xAP datetime value)
---optional
resettime=(xAP or enhanced xAP datetime value of last statistics reset)
---optional, exclusive with period
period=(value, in whole seconds, since statistics were reset)
---optional, exclusive with resettime
}
alarmhi.<type> / alarmlo.<type>
---optional header, the presence of which specifies an alarm situation
{
unit=(unit designation)
---mandatory field
value=(value of the alarm threshold)
---mandatory
}

TSC.request

Status Request

Specific

Used to elicit a TSC.info/TSC.event response for a specific endpoint type matching the target address.

class=tsc.request

request.<type>
---mandatory section header, where <type> is the sensor/measurement type
{
}

All

Used to elicit a TSC.info/TSC.event response for all endpoints matching the target address.

class=tsc.request

request.all
---mandatory section header
{
}

Capability Request

class=tsc.request

request.capability
---mandatory section header
{
}

TSC.capability

class=tsc.capability

capability.<type>
---mandatory header, <type> is sensor/output/measurement type, repeated for each endpoint
{
ID=(01 to FE, endpoint subaddress)
---mandatory
type=(either input or output. A single endpoint cannot be both)
---mandatory
unit=(measurement unit)
---mandatory
maxvalue=(the maximum legal value that can be reported)
---mandatory
minvalue=(the minimum legal value that can be reported)
---mandatory
setpointmax=(the maximum value of a setpoint threshold)
---optional, if missing then device does not support max setpoints
setpointmin=(the minimum value of a setpoint threshold)
---optional, if missing then device does not support min setpoints
setpointlist=(comma separated list of setpoint types)
---optional, if missing then only 'current' is supported by the device
}
alarmhi.<type>
---mandated if device supports alarms, <type> is sensor/output/measurement type
{
ID=(01 to FE, endpoint subaddress)
---mandatory
unit=(unit designation)
---mandatory
value=(value of maximum input/output alarm threshold, in 'units')
---mandatory
}
alarmlo.<type>
--- mandated if device supports alarms, <type> is sensor/output/measurement type
{
ID=(01 to FE, endpoint subaddress)
---mandatory
unit=(unit designation)
---mandatory
value=(value of minimum input/output alarm threshold, in 'units')
---mandatory
}
stats.<type>
--- mandated if device will generate statistics, <type> is sensor/output/measurement type
{
ID=(01 to FE, endpoint subaddress)
---mandatory
}