Basic Status and Control – Specification

V1.3, 16 July 2004

(a summary of the changes from BSC v1.2 is included at the end of this document)

By Kevin Hawkins, Edward Pearson and Mark Harrison.
Thanks to Mary Harrison, James Traynor, Ian Bird, Patrick Lidstone and Stuart Booth.

Comments to Kevin@xapautomation.org

Overview

Basic Status and Control (BSC) is a set of xAP Schema intended to provide a common interface to "Basic" devices. BSC provides a simple mechanism to allow discovery, control & status reporting and as such provides immediate interoperability across compliant devices and software applications.

It should be stressed that a xAP connector capable of controlling a device using BSC may also be capable of controlling the same device using a more abstract Schema. If a BSC device is so controlled, it should be capable of reporting in BSC no matter how it is controlled. An example might be an X10 lighting controller which is capable of controlling X10 devices using either BSC or the X10 schema.

Basic Devices are categorised in two ways. Indeed the definition of a Basic device is, in round terms, whether it can be so categorised.

- Input or Output
- Binary, Level or Stream

An **Input** device is defined as a device that only provides input into a xAP domain. Specifically, it has no mechanism for control of its state from xAP. An example might be a temperature sensor, which can only report temperature. Another example might be a switch input, which will tell whether a manual switch is in "On" or "Off" position. An input device reports its state within an input.state body section

An **Output** device is defined as a device that can additionally be controlled from xAP. An example might be a relay. It should be noted that Output devices can also have their state changed from outside the xAP domain – for example a relay might have a manual switch. It is, of course, vastly preferred if an Output device is physically constructed so as to allow such external changes to be notified back using xAP. Output devices are contained within output.state body sections

All three types of devices must support a **State=** key value pair that reflects the devices state as **On** , **Off** or **?** Additionally a State=toggle may be **sent** to a device to ask it to change state from Off to On or On to Off.

A **Binary** device has only one of three possible states. **On, Off, or ?**. The ? state is used to designate a state either unknown, or in error. An example binary device might be a relay, or an input switch. Careful consideration should be given to whether the "open" position on a switch should represent "on" or "off" - for momentary switches, the "on" state should indicate that the switch is being pressed, irrespective of whether the press forces open or closed. Binary devices report using one mandatory key value of State=

example **State= Off**

A **Level** device can be set to a level between **0% and 100%** or to an exact level. It uses the additional key name **Level** and always repots its level as a value in the format Level=current/max where max reflects the resolution of the device based on a 0 to maxvalue index. For example Level=128/255 for an 8 bit device at 50%. Again, the ? state is used to designate a state either unknown, or in error (Level=?). A dimmer would be the classic example of a level device. For consistency across BSC devices the 0% to 100% are a way of being resolution independent, and if a device's internal representation uses some other upper and lower bounds, these should be scaled so that "on full" is always represented as 100%. A level device can be set to a specific level at its native resolution using Level=45 for example (no % sign) or Level= 64/1023. In the latter example if the level device does not support 1024 discreet levels it should scale proportionally accordingly. Note that a level device can be set to a % but should never report as a %. Level devices report two mandatory key values of **State=** and **Level=**

example **State=On**
Level= 256/511

A **Stream** device is a data based device where **some other value, typically text** might be used. An example might be the display text on a simple LCD driver. Stream devices include two mandatory key values of **State= and Text=**

example **State=ON**
Text=23.5°

All 3 types of devices can include one more optional defined key value of **DisplayText=** this is used as an annotation for display purposes to provide more appropriate feedback. For example a door sensor may report as a binary device using State=On and State=Off but would wish to display as Open or Closed

example **State=Off**
DisplayText=Closed

example **State=On**
Level=250/255
DisplayText=Full

Additional key/value pairs may be included in a BSC body part and applications should either interpret these appropriately or gracefully ignore unexpected key/value pairs

BSC Device Type	**State=**	**Level=**	**Text=**	**DisplayText=**
BINARY	Yes	No	No	optional
LEVEL	Yes	Yes	No	optional
STREAM	Yes	No	Yes	optional

BSC v1.3 defines four schemas that are used with Basic Devices.

xAPBSC.cmd
xAPBSC.query
xAPBSC.event
xAPBSC.info

The Schema – overview

xAPBSC.cmd

Messages in xAPBSC.cmd are used to instruct a device to change the state of its outputs.

xAPBSC.query

Messages in xAPBSC.query are used to query the state of either an Output or an Input device.

xAPBSC.event

Messages in xAPBSC.event are sent to indicate that the state of an Input or Output device has just changed.

xAPBSC.info

Messages in xAPBSC.info are sent to report the current state of an Input or Output, but to indicate that this message has not been triggered by a state change. (It might be sent periodically, or on device startup, or in response to a xAPBSC.query)

Subaddressing, Targeting and Wildcarding

The xAP specification allows for a sophisticated level of wildcarding of addresses, using multiple operators.

It should not be assumed that wildcarding is supported by all BSC devices, particularly wildcarding of hardware subaddresses. It is, of course, strongly recommended that devices support it where possible. (This will help with discovery and configuration later.)

However, all devices must respond to the wildcard address *.*.>

One of the key principles of BSC is that each "endpoint" within a xAP device has a unique UID subaddress (subUID). This is a two-hex-digit address, and forms the last two digits of the UID header field, it is a unique numerical identification equivalent to the human readable sub address name which is everything after the : in the source address

> UID=FF1234**03**
> Source=ACME.Controller.apartment:**BedsideLamp**

It is the responsibility of the programmer to determine the mapping between actual endpoints and subUID's. It is generally to be recommended that the subUID's used should be contiguous wherever possible. In any event, the mapping should be deterministic and documented.

xAPBSC.cmd

Messages in xAPBSC.cmd are used to set the state of an Outputs within one device. They are typically sent TO a device supporting BSC.

A message in xAPBSC.cmd must set the header field Class to “xAPBSC.cmd”.

A message in xAPBSC.cmd must include a TARGET field in the xAP header. This target field must match the device being controlled. It may include wildcarding (but see the note above.)

A message in xAPBSC.cmd will include a body part(s) titled output.state.n. One body part per endpoint to be controlled will be included and these will be indexed sequentially output.state.1 output.state.2 etc. Thus several endpoints can have their state changed instantaneously.

This body part contains a series of key/value pairs appropriate to the device being controlled. Included in every body part is an ID= key which is a two-hex-digit number corresponding to the UID subaddress of the endpoint to be controlled. Alternatively a single body part can be used with ID=* to address all endpoints that match the target address.

- If the two-hex-digit number given does not match a UID subaddress that exists, then no response should be given and no action taken.

- If the two-hex-digit number does not align with the TARGET address (taking into account wildcarding of the subaddress and the internal mapping of targets to UID subaddresses), then no response should be given and no action taken.

The other key values included in the body should be appropriate to the device being controlled ie Binary, Level or Stream.

If the control causes a value to change, a xAPBSC.event message must be raised in response. If the control does not cause the value to change then you should issue a xAPBSC.info event.

Example

```
{
   v=12
   hop=1
   uid=FF123400
   class=xAPBSC.cmd
   source=ACME.Controller.Central
   target=ACME.Lighting.apartment:>
}
output.state.1
{
  ID=03
  State=ON
  Level=50%
}
output.state.2
{
  ID=1B
  State=OFF
}
```

In this case, the xAP device is a lighting controller, configured with apartment as the instance

ID. The message is generally targeted at all UID subaddresses on this device by using the '>' wildcard. Within the body two sections change outputs corresponding to subUID 03 (BedsideLamp) and 1B. device 03 we already know to be the BedsideLamp

Example

```
{
   v=12
   hop=1
   uid=FF123400
   class=xAPBSC.cmd
   source=ACME.Controller.Central
   target=ACME.Lighting.apartment:outside.>
}
output.state.1
{
  ID=*
  State=OFF
}
```

This message would cause all binary outputs named outside.<*anything*> to turn off

Eg these would all turn off

ACME.Lighting.apartment:outside.Floodlights
ACME.Lighting.apartment:outside.sprinklers
ACME.Lighting.apartment:porchlight

xAPBSC.query

Messages in xAPBSC.query are used to query the state of either an Output or an Input device.

A message in xAPBSC.query must set the header field Class to "xAPBSC.query".

A message in xAPBSC.query must include a TARGET field in the xAP header. This target field must match the device being controlled. It may include wildcarding (but see the note above.)

A message in the xAPBSC.query must include at least one body part (to conform with the xAP specification.) This body part may be called anything (though "request" is recommended), and its contents will be ignored and may be left blank.

The BSC device responds with a separate xAPBSC.info message for each and every endpoint matched by the TARGET (including wildcarding.) As such, a xAPBSC.query message targeted to subaddress > can be used as a rudimentary form of discovery for endpoints as every endpoint will respond individually.

Example

```
xap-header
{
   v=12
   hop=1
   uid=FF123400
   class=xAPBSC.query
   source= ACME.Controller.Central
   target= ACME.Lighting.apartment:BedsideLamp
}
request
{
}
```

In this case, the status of the BedsideLamp is being requested from by the ACME controller. The Acme lighting controller will respond with a xAPBSC.info message.

xAPBSC.event

Messages in xAPBSC.event are sent to indicate that the state of an Input or Output device has just *changed*.

A xAPBSC.event must be returned by a device supporting BSC whenever an endpoint changes status, irrespective of how that status change was triggered. Some devices with rapidly changing inputs may choose to modify this behaviour (see below) but must always respond to any xAPBSC.cmd message requesting a state change. For the avoidance of doubt, this includes the following:

- A xAPBSC.cmd message
- A message in another xAP schema affecting that device
- An externally triggered event (such as a button press)
- A change caused by an event within the xAPplication (such as a Homevision Macro)

NB. If a xAPBSC.cmd is received but the device is already in the state requested a xAPBSC.info response should be sent and not a xAPBSC.event

A message in xAPBSC.event must set the header field Class to "xAPBSC.event".

A message in xAPBSC.event must contain a fully qualified source specifying the hardware endpoint generating the event. This source is in human-readable format, and need not contain the two-hex-digit representation of the endpoint. The UID must be set to be that of the endpoint affected.

A message using xAPBSC.event must contain one of the following body parts: input.state or output.state. Which part is contained depends on whether an input or an output device has changed state.

The body part must contain key/value pairs appropriate to the device type being reported (BINARY|LEVEL|TEXT/STREAM). The value should correspond to the new value for the device.

Example

```
xap-header
{
   v=12
   hop=1
   uid=FF776107
   class=xAPBSC.event
   source= ACME.Lighting.apartment:BedsideLamp
}
input.state
{
   State=ON
   Level= 64/255
}
```

In this case, the hardware endpoint with subUID is reporting that it is on at 25% brightness. The device supports 256 discreet levels. All level based devices report in their native resolution and not as % values.

NB At the hardware developers discretion a rapidly changing input may choose to not report each and every change instead electing to report periodically or after a certain degree of change or when a certain counter value has been reached for example. For example it's probably pointless to report a fluctuating voltage level at 12 bit precision and instead perhaps elect to only report when the most significant 5 bits changed. Such devices however must always send xAPBSC.event messages rather than xAPBSC.info messages if their new value is different to their last reported valur, including responses to xAPBSC.query messages.

xAPBSC.info

Messages in xAPBSC.info are sent to indicate the current state of an Input or Output, but to indicate that this message has not been triggered by a state change.

A xAPBSC.info **must** be returned by a device supporting BSC whenever a xAPBSC.cmd message is received querying the status of that endpoint.

A xAPBSC.info **should** be returned for each hardware endpoint when the xAP connector / device start up.

A xAPBSC.info **may** be returned periodically.

A message in xAPBSC.info must set the header field Class to "xAPBSC.info".

A message in xAPBSC.info must contain a fully qualified source specifying the hardware endpoint generating the event. This source is in human-readable format, and need not contain the two-hex-digit representation of the endpoint. The UID must be set to be that of the endpoint reported.

A message in xAPBSC.info must contain one of the following body parts: input.state, output.state. Which part is contained depends on whether an input or an output device has reported its state.

The body part must contain key/value pairs appropriate to the device type being reported (BINARY|LEVEL|TEXT/STREAM).

The value should correspond to the value of the device.

Example:

```
xap-header
{
   v=12
   hop=1
   uid=FF776147
   class=xAPBSC.info
   source= ACME.Lighting.apartment:Outside.Floodlights
}
output.state
{
   State=OFF
}
```

In this case, the hardware endpoint known Outside.Floodlights corresponding to subUID endpoint 47 (Hex) has provided information. It is a binary device and is currently off. However, this information represents a request to a query, or a startup, or a periodic notification. It is explicitly NOT a state change (because that would have been a xAPBSC.event message instead.) However, it may be a "new" state as a result of device startup.

Revision History

BSC v1.3 16th August 2004
Public Release

Level values clarified to be based on a 0 index ie Level = 128/255 rather than 128/256 previously. To aid consistency any fractional values should be adjusted to nearest integer with .5 values rounding upwards as above for 50%.

BSC v1.3draft 16 July 2004

Existing single subUID= parameter segmented into STATE LEVEL and TEXT
"subUID=" key values are now deprecated
Multiple individual indexed body sections are used to address individual endpoints
All endpoint addressing using ID=*
Schema names changed as follows :

NEW	OLD
xAPBSC.cmd	xAPcmd.state
xAPBSC.query	xAPcmd.query
xAPBSC.info	xApstatus.info
xAPBSC.event	xAPstatus.event

DisplayText optional key/value added for labelling
Level devices may be set by % or specific value in any resolution
Level devices report in native resolution only (never %)
Added toggle STATE change
Unexpected extra key/value pairs within body sections must be gracefully ignored (true in headers too)
Developer may 'opt out' of continual xAPBSC.event reporting for frequently changing input values

BSC v1.2 First Public Release 18 May 2004