# Saturn South Pty Ltd

# ESCo - ESBox Protocol V1.0 (draft 3)

**Home Automation Version**

# Overview

This document describes a RESTful HTTPS based protocol for the secure exchange of basic commands and responses as required to interact with a Saturn Energy system. The protocol uses human readable commands encoded in JSON, with communications between the client (Energy Services Box - 'ESBox') and server (Energy Services Company - 'ESCo') initiated by the client at a settable frequency.

## Security

HTTPS sessions between the ESCo (server) and ESBox (client) are required to use SSLv3. For an ESCo to be trusted, its certificate must:

- Be valid at the date of use, and
- Be signed by a root CA certificate or by an intermediate certificate (that is part of a complete chain to a root CA certificate), where each root and intermediate certificate is present on the ESBox. Because of memory limits on the ESBox LT platform, CA root certificates must be carefully selected. Contact Saturn South for more information.

## Optional Unsecured Mode

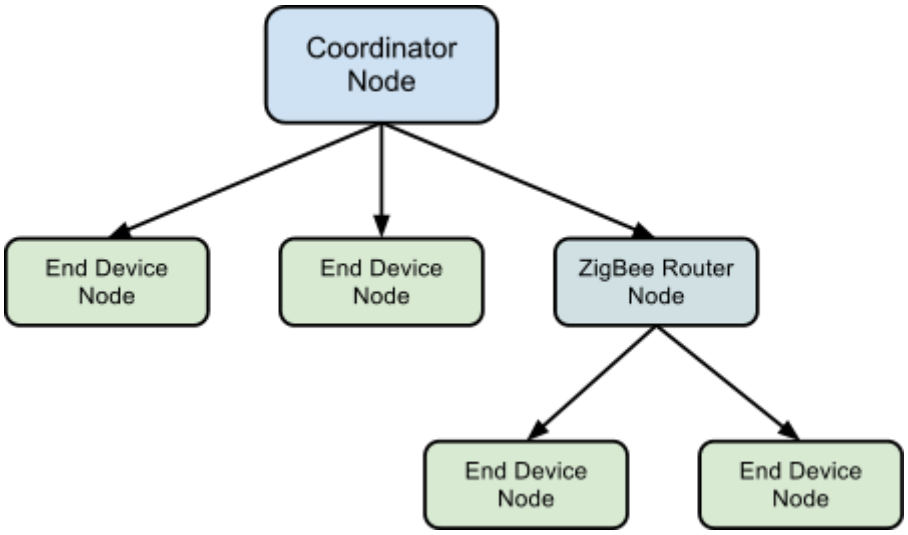
SSL can be disabled on the ESBox by changing the ESCo port in the ESBox Options to a port other than 443. ESBox devices can be provided set to unsecure mode by default as required. Unsecured mode is recommended for initial development of ESCo implementations.

## Proxy Support

This version of the ESBox-ESCo protocol does not support the configuration and use of manual proxy servers, but will work correctly on most networks that operate behind transparent proxies. Full proxy support is planned for a future release of this protocol.

## A brief introduction to ZigBee networks and devices

ZigBee networks consist of a number of devices that communicate via the ZigBee protocol. There is exactly one **coordinator** in a ZigBee network, and any number of **end devices** and **routers**. In a network built from Saturn South devices, the ESBox is the network coordinator, while end devices would be Mini CT Meters and Mini Smart Meters, and a router would be the Range Extender device.




**A typical ZigBee network**

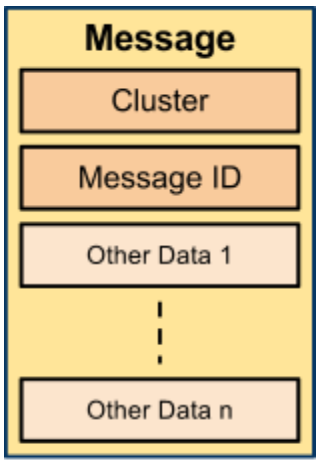
An individual physical device in the network is referred to as a **node**. Within a node, there can be one or more **endpoints**. The concept of endpoints allows a single physical unit to be made up of multiple unrelated or distinct ‘sub devices’. A good example of this would be a wall mounting light switch unit with three buttons, where each button belongs to a different endpoint. In this way, each button can be used for completely separate applications while still sharing the same ZigBee radio module and being part of the same physical product.

Each endpoint supports a set of **clusters**. Clusters are basically namespaces - a way to separate **commands** and **attributes** into sensible groups that relate to real-life applications. One example is the Simple Metering cluster, which defines a set of attributes for reporting fundamental physical quantities related to major real-life metering applications such and energy metering, water metering, and gas metering.

# Messages and Containers

## Messages

Messages are self-contained application-level JSON objects that represent either a single command/request from the ESCo (e.g. “Send me the current device list”), or a single response/alert from the ESBox (e.g. “Here is my current device list...”).




General Message Structure

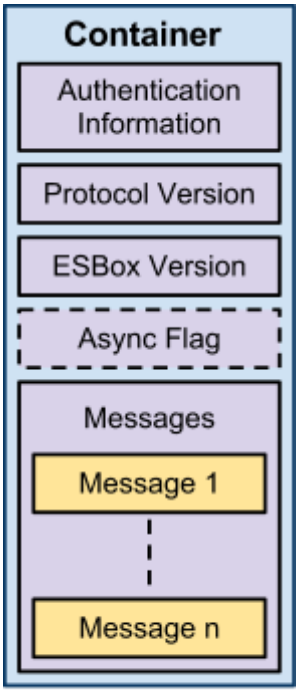
All Messages contain a Cluster identifier that defines the broad class of command (effectively a namespace, modelled off the ZigBee cluster model) - more information on clusters is provided in the API documentation.

Messages also contain a Message ID, used to identify the type of message within the indicated Cluster’s namespace.

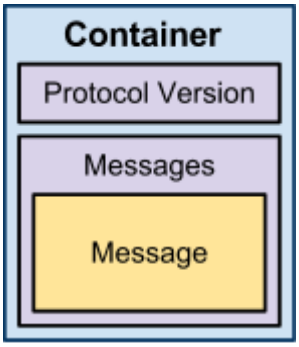
Depending on the message type, there may be any number of additional items included in each message (‘Other Data’). The exact contents of each message is defined in the API documentation.

## Containers

Containers are JSON objects that encompass all application-level data sent in an HTTP request/response, and contain one or more Messages as well as authentication and version information where appropriate.




Containers sent from ESBox to ESCo




Containers sent from ESCo to ESBox

Containers sent from the ESBox to the ESCo:

- Contain full authentication credentials for that ESBox (i.e. there is no concept of a session, unlike earlier versions of this protocol).
- Include an ESBox version string that defines the ESBox software and firmware revisions.
- Must contain **at least one** Message.
- May be marked as “Asynchronous” with the *Async* flag. See the section below on Asynchronous containers for more detail.

Containers sent from the ESCo to the ESBox:

- Are only sent in response to a Container from the ESBox. (i.e. The ESBox is responsible for initiating communications with the ESCo - communications with an ESBox can **not** be initiated by an ESCo)
- Must contain **exactly one** Message. (i.e. the ESCo can only issue one command to the ESBox at a time)

All Containers sent between the ESBox and ESCo are transmitted over HTTP with the content of Containers being packed JSON sent in the message body of each HTTP request/response.

## Sending Messages from the ESCo to the ESBox

An example Message is shown below (defined as a Python dictionary in this example):

```
example_permitJoining_message =
{
      # Select the Cluster - Saturn South ESBox
      M.F.Gen.Cluster : M.Clusters.SS_ESB,
      M.F.Gen.MsgID : M.SS_ESB.E.PermitJoining,

      # Set the permit join time in seconds
      M.F.Dat.Duration : 60,
}
```

This Message is sent by the ESCo to the ESBox to instruct the ESBox to enable Permit Joining mode in the ZigBee network so that new devices (such as smart meters) can be joined to the ESBox.

In order to be sent to the ESBox, this message must be placed into a Container:

```
example_container =
{
      M.F.Gen.ProtocolVersion : "1.0",

      M.F.Gen.Messages : [
            {
                  # Select the Cluster - Saturn South ESBox
                  M.F.Gen.Cluster : M.Clusters.SS_ESB,
                  M.F.Gen.MsgID : M.SS_ESB.E.PermitJoining,

                  # Set the permit join time in seconds
                  M.F.Dat.Duration : 600,
            }
      ]
}
```

Note that the Message is placed into the list keyed to `M.F.Gen.Messages` (that is, a list of messages of length 1).

Containers are packed using JSON and are left in a human readable form for ease of development and testing. Note that that the ESBox will not interpret the JSON identifiers `true`, `false` or `null`. In addition, hexadecimal escape sequences in strings are not supported (other JSON escape sequences are supported however).

Constants such as `M.F.Gen.Cluster` are defined in SSMessages_1_0.py, and in all cases are mapped to descriptive strings that closely match the constant name. Once packed with JSON, the Container in the above example is represented by the string:

```
'{"Msgs": [{"Duration": 600, "MsgID": "PermitJoining", "Cluster":
{"ClusterID": 0, "ClusterMfctr": 4278}}], "ProtocolVersion": "1.0"}'
```

Which is then placed into the body of the ESBox HTTP response to the ESBox. An example HTTP response that includes this Container is shown below:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Path=/dsrm-comm
Content-Length: 134
Content-Type: application/json
Date: Wed, 30 Jun 2012 05:04:19 GMT

{"Msgs": [ {"Duration": 600, "MsgID": "PermitJoining", "Cluster":
{"ClusterID": 0, "ClusterMfctr": 4278}} ], "ProtocolVersion": "1.0"}
```

Note that servers are required to support absolute URIs. The ‘application/json’ content type must be defined in the response, or the ESBox will discard the Container.

The example above defines a content length for the message, however the ESCo could also use chunked encoding when sending messages to the ESBox.

## Receiving Containers from the ESBox

An example of a typical HTTP PUT request sent by the ESBox to the ESCo is shown below:

```
PUT http://www.example.com/dsrm-comm/ HTTP/1.1
Host: www.example.com:80
Connection: Keep-Alive
Accept-Charset: US-ASCII, ASCII
Content-Type: application/json
Transfer-Encoding: Chunked

e7
{"Msgs": [{"Cluster":{"ClusterID":64784,"ClusterMfctr":4278},
"MsgID":"NoFurtherMessages"}], "Auth":["001BA501B070001A",
"11111111111111111111111111111111"], "ProtocolVersion":"1.0",
"ESBoxVersion":"SS9002.1.2_5015_4890_4200_5021"}
0
```

Note that the ESBox will only ever send requests to the ESCo using the HTTP PUT method. Chunked encoding will always be used for communication from the ESBox to the ESCo and must be supported by the ESCo.

Only charsets US-ASCII and ASCII are supported. IEC_8859-1 is also accepted, but non-ASCII characters may cause issues or be discarded without warning. In this regard, the ESBox is not HTTP/1.1 compliant.

The message body of the PUT request contains a JSON packed Container (shown below with strings substituted for constants as defined in SSMessages_1_0.py):

```
{
      M.F.Gen.ProtocolVersion : "1.0",
      M.F.Gen.ESBoxVersion : "SS9002.1.2_5015_4890_4200_5021",
      M.F.Gen.Auth : ["001BA501B070001A",
                      "11111111111111111111111111111111"],

      M.F.Gen.Messages : [
            {
                  M.F.Gen.Cluster : M.Clusters.SS_ESB,
                  M.F.Gen.MsgID : M.SS_ESB.E.NoFurtherMessages
            }
      ]
}
```

This container is comprised of a protocol version identifier, a tuple of length 2 (keyed to `M.F.Gen.Auth`) containing the IEEE address and link key of the ESBox, an ESBox version string, and a list of messages containing a single message with MsgID “NoFurtherMessages” (the default message sent by the ESBox when there are no queued responses waiting to be sent to the server. Note that IEEE address must always be 16 capitalised hex characters with no prefix or suffix.

### ESBox Version String

The version is string is constructed as follows:

“<hardware version>_<userspace version>_<ESP module version>_<coordinator version>_<bootloader-1 version>_<webapp version>”

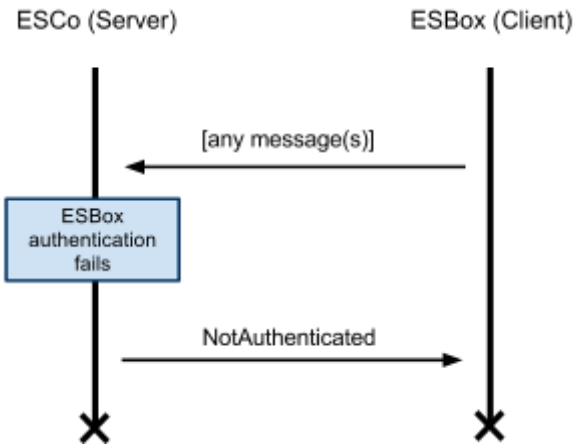
An example of an ESBox version string is: SS9002.1.2_5015_4890_4200_5021.

*Hardware version code:* The hardware model and revision of the ESBox.
*Userspace version:* The firmware revision of the main ESBox application (ESCo comms, database management, user interface, etc)
*ESP module version:* The revision of the application-level ZigBee firmware.
*Coordinator version:* The revision of the ZigBee coordinator firmware (this firmware runs on a separate processor to the applications firmware)
*Bootloader 1 version:* Firmware revision of the level 1 bootloader.
*Webapp version:* Revision of the webapp resources stored on the ESBox’s uSD card.

# ESCo - ESBox Dialogue

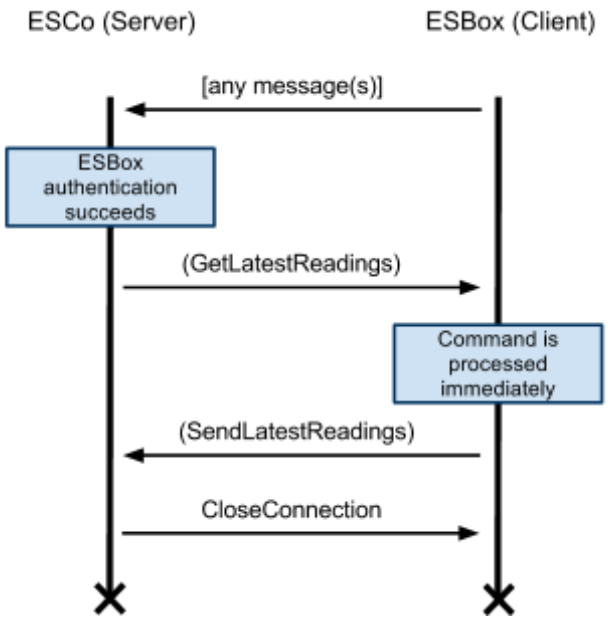
The ESBox will attempt to communicate with the ESCo every *PollESCoInterval* seconds by opening an HTTP or HTTPS connection, and sending a container to the ESCo consisting of either the accumulated messages in its outbound message queue or the default *NoFurtherMessages* message.

If the ESCo does not recognise the device that is attempting to connect, or the connecting device fails to correctly authenticate itself, the ESCo will respond with a *NotAuthenticated* message as shown below and force close the connection.




If the ESBox is successfully authenticated based on the information supplied in the Auth key of the Container, the ESCo will act upon the Messages within the Container as required, after which it is the turn of the ESCo to reply to the ESBox.

The first message sent by the ESCo to the ESBox will be the first queued application message. In the example below, the server queries the ESBox for a report of the latest readings received from all devices joined to the ESBox (*GetLatestReadings* in the example below). Upon receipt of the *GetLatestReadings* message from the ESCo, the ESBox will nominally assemble a response message immediately (*SendLatestReadings*) and send it to the ESCo as the next outbound message.



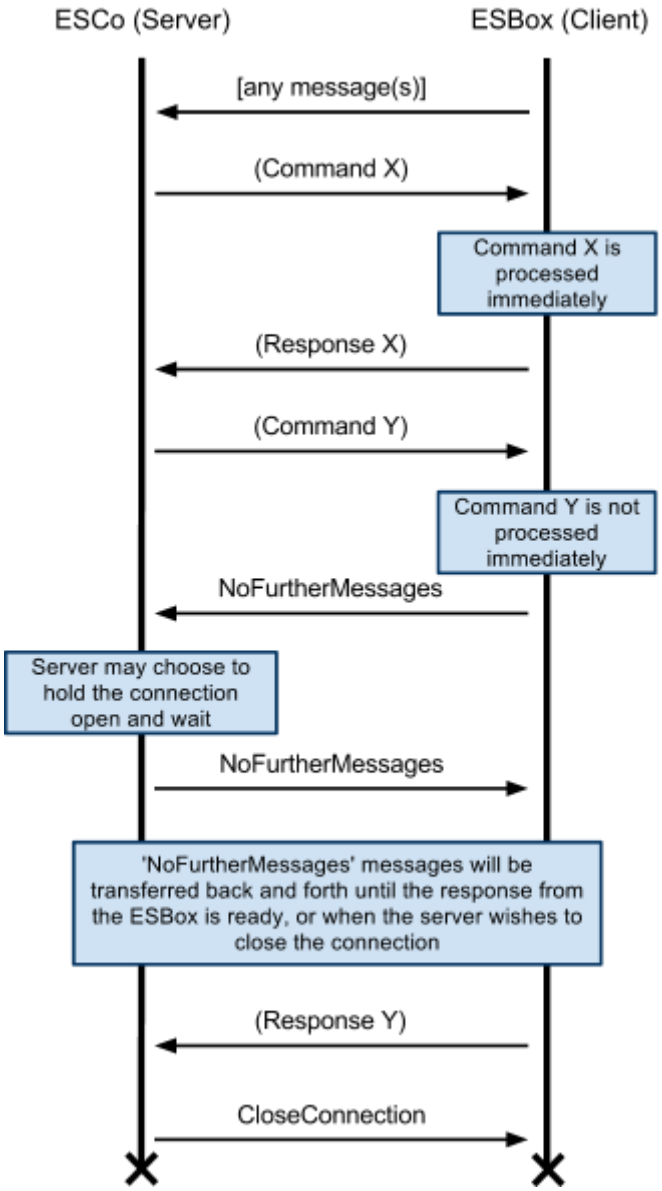

Note that the ESBox is responsible for closing the connection to the ESCo, and will do so upon receipt of a *CloseConnection* message from the ESCo (and under other circumstances, as documented below).

Although many of the commands that are issued to the ESBox by the ESCo in this specification are processed synchronously and are responded to immediately (marked as having an '**Immediate Response**' in the API documentation), some command messages that can be issued by the ESCo may have their response generated at an arbitrary time in the future. Such messages (both commands and corresponding responses) are marked in the API documentation as having a **'Deferred Response'**. Note that this concept should not be confused with **asynchronous Containers**, which are described in a later section.

In the example below, the initial message(s) from the ESBox is/are followed immediately by *Command X* from the ESCo which is then processed synchronously by the ESBox, with a response sent in the form of *Response X* in the next ESCo-bound transmission. The ESCo then sends the next message in its queue; *Command Y*, which is processed in a delayed fashion by the ESBox.

Because the ESBox has no *response Y* ready to send, it responds with the *NoFurtherMessages* message to indicate that its outbound message queue is empty (notwithstanding any other messages that have been generated for other purposes). At this point, the server can choose to hold the connection open by sending *NoFurtherMessages* messages (and receiving NoFurtherMessages Messages from the ESBox) until enough time has elapsed for the ESBox to generate and send *Response Y.* The server can also elect to wait for any arbitrary amount of time before 'polling' the ESBox with a *NoFurtherMessages* Message, provided that the wait time

is less than the currently configured *ESCoTimeout* period (which, if exceeded, will cause the ESBox to close the connection and attempt to reconnect):

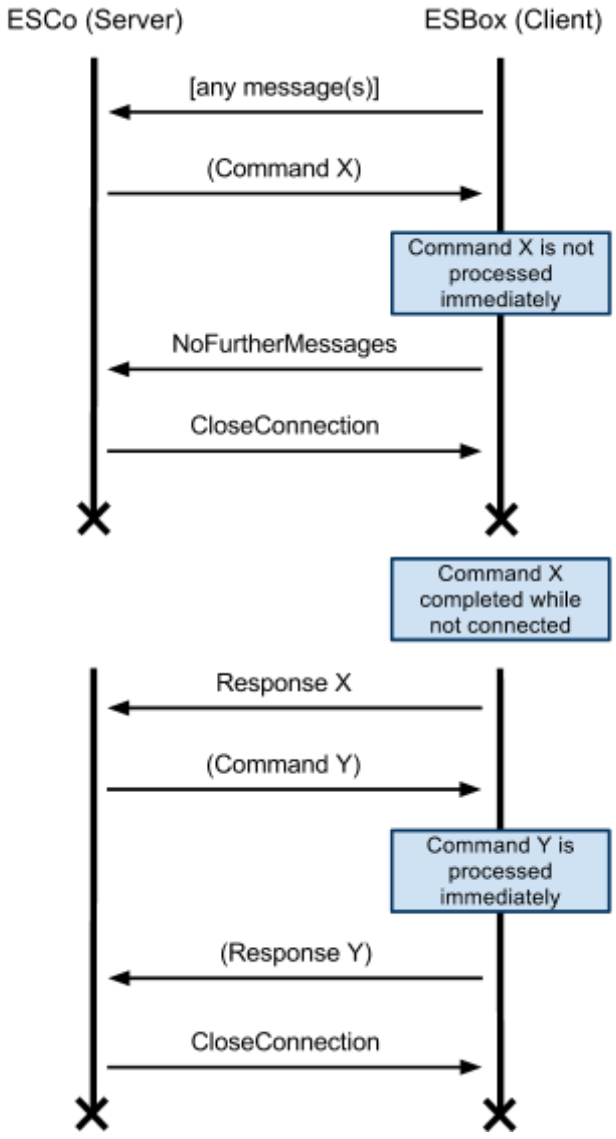



See the section on **HTTP Long Polling** below for more information on long polled operation.

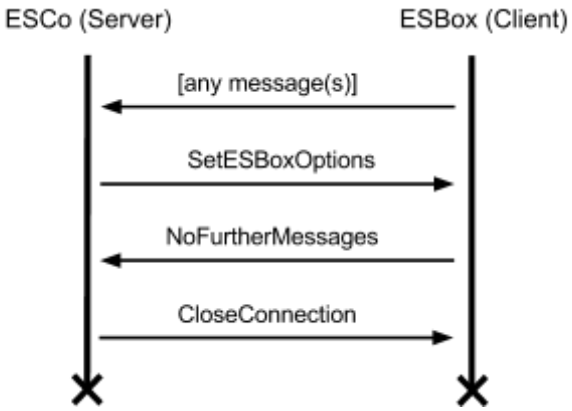
Alternatively, the ESCo can close the connection and receive the pending response during the next connection by the ESBox. In the example below, *Command X* is a 'No immediate response' type Message, and the ESCo decides to close the connection rather than wait for the response by exchanging *NoFurtherMessages* Messages. In this example we assume that

*Command X* was processed, and *Response X* generated and placed on the outbound queue in the time between the two connections.

When the next *LoginRequest* is received from the ESBox, the ESCo sends the next Message that is waiting in its outbound message queue, *Command X. Command Y* is processed by the ESBox immediately and added to the outbound queue, and is sent to the ESCo in the first transmission of the following connection.

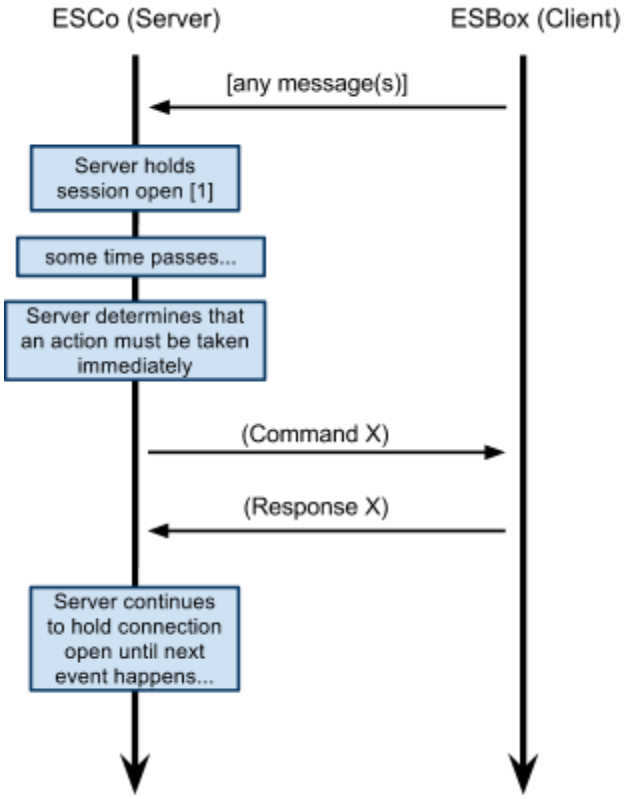


Some commands sent to the ESBox by the ESCo do not require a response. When this type of command is received, the ESBox performs the performs the necessary action if possible, and the ESCo assumes that the command was executed successfully. Messages that do not elicit a response from the ESBox are marked in the public API documentation as '**No Response**'.



# HTTP Long Polling

For applications requiring frequent reporting of collected data, HTTP long polling can be used by the ESCo to throttle the polling rate and avoid the protocol overhead associated with opening HTTP/SSL connections.




For Demand Management and time-critical applications, HTTP Long Polling is the preferred technique for driving the ESBox-ESCo protocol as it minimises the overheads associated with establishing new HTTP or HTTPS sessions and minimises the delay between a command being generated by the control server and that command being received by the ESBox.

When Long Polling used, the configuration of the ESBox Options, specifically, timeouts is different, and the following rules should be observed for proper operation.

- The *ESCoTimeout* should be set as low as possible without causing false-positive timeouts. For long polling, this is: poll interval' + 'maximum expected response time'. A typical safe value for 'maximum expected response time' is 10 seconds.
- The *ESCoPollInterval* is typically set to the same as *ESCoTimeout* in Long Polling mode. It may be set to less than *ESCoTimeout* if a more rapid retry is required after a connection failure, but under no circumstances should it be set to less than ¼ of *ESCoTimeout*.

# Asynchronous Containers

**Note: For the 1.0 release of this specification, the EnableAsynchronousContainers flag will not be settable in the ESBox LT. This section is included to allow ESCo implementors to build in support for this feature in preparation for when it becomes available, if required.**

During normal operation, the sequence of the HTTP dialogue between the ESBox and the ESCo will follow the traditional Request->Response model, i.e.:

1 ESBox connects and sends queued messages (HTTP Request)
2 ESCo processes messages and issues commands to ESBox (HTTP Response)
3 ESBox sends next outbound message(s) (HTTP Request)
4 Repeat 2 and 3 until either party closes the connection

In certain circumstances it may be preferable for the ESBox to deliver specific high priority messages to the ESCo immediately, irrespective of the current state of the ESBox-ESCo Request/Response cycle. In the current version of this specification, there are only two circumstances in which this is likely to be the case:

- For sending **DispatchReport** Messages to the ESCo announcing the success or failure of a **BroadcastDispatch** command for high-speed Demand Management purposes, and
- For sending **UFLSReport** Messages to the ESCo announcing that a device has autonomously shed a load based on detection of an under frequency event.

When this protocol is being used with HTTP Long Polling, in which the server will hold the connection idle for some time between sending commands to the ESBox, the ESBox would typically not be able to communicate these messages to the ESCo until the ESCo has once again polled the ESBox. In some systems this may be acceptable if the polling interval is quite short, however this specification includes reservation for a special **Asynchronous Container** that, if enabled, may be sent at any time from the ESBox to the ESCo.

The *Async* flag marks a Container as being asynchronous. This flag would typically be used by the ESCo to determine that the Container has not been sent in response to the last command issued to the ESBox (in the event that an asynchronous Container is delivered to the ESCo while the ESBox is still processing a blocking command), and that it should not proceed to send the next command in its outbound queue to the ESBox. In this way, asynchronous messages can be sent to the ESCo without disrupting the overall Request->Response model.

It is important to reiterate that the ESCo may only send one command to the ESBox at a time in normal operation. This is enforced partially by the requirement for Containers sent by the ESCo to the ESBox to contain exactly one Message, and an ESCo would typically wait for the next Container to arrive from the ESBox before sending another API message. The *Async* flag, if included in a Container that arrives at the ESCo, would inform the ESCo that it should not consider receipt of the Container to indicate that the ESBox has finished processing the last

Message that was issued, and that the ESCo should therefore not yet send the next outbound Message.

Asynchronous Containers will only be sent by the ESBox if the *EnableAsynchronousContainers* flag is set in ESBoxOptions. This flag is clear by default. When this flag is set, **dispatch report** and **UFLS report** Messages are automatically transmitted asynchronously.

# Public API Message Reference

This section defines the ESCo->ESBox and ESBox->ESCo messages for each of the clusters covered by this specification.
This section should be read with reference to the SSMessages_1_0 constants definition file and the set of example Python message structures in ExampleMessages_1_0.py.

# SS_ESB (Saturn South ESBox Cluster)

**Cluster ID:** 0
**Manufacturer ID:** 4278
The SS_ESB cluster consists of messages to manage site-wide policies and ESBox specific settings, as well as low-level management of the ZigBee network and interaction with the ESBox database.

## ESCo to ESBox

### Protocol Messages

#### NoFurtherMessages

*No Response*
Sent when the outbound message queue of the ESCo is empty, but the ESCo does not wish to close the connection.

#### NotAuthenticated

*No Response*
Sent to inform an ESBox that it is not authenticated with the ESCo. The HTTPS connection will be closed by the ESCo immediately after this message is sent.

#### CloseConnection

*No Response*
Sent to instruct the ESBox to close the HTTP connection.

### ESBox Management Messages

#### GetESBoxOptions

*Immediate Response: SendESBoxOptions*
Sent to request that the ESBox send the current state of its ‘ESBox Options’. For a description of the contents of ‘ESBox Options’, see the *ESBox Options* section below.

**SetESBoxOptions**

*No Response*

Sent to modify the value of one of the writable ESBox Options. This command contains one or more Setting/Value pairs. For each valid Setting/Value pair present in this command, the ESBox will update the corresponding Setting in its local 'ESBox Options' dictionary. This command forces the ESBox to write its current 'ESBox Options' dictionary to non-volatile memory. If a Setting/Value pair is defined in this command with an invalid Value or with a Setting that does not exist, that Setting/Value pair will be ignored.

**RestartESBox**

*No Response*

Force the ESBox to perform a restart. This command forces the ESBox to commit any unsaved settings to non-volatile memory before restarting. Note: This command does not cause the ESBox to return to factory state.

**SendUpdateToken**

*No Response*

Used to send an update token to the ESBox that it will use to retrieve and install an update directly from the update server. See update server specifications for more information on how this token is used. Once this message is received by the ESBox, it will immediately close the connection to the ESCo and connect to the update server defined in the update token.

**GetUpdateStatus**

*Immediate Response: SendUpdateStatus*
*Not currently implemented*

This message requests the ESBox to provide the current status of any updates it is performing.

**ExecuteTerminalCommand**

*No Response*
*Not currently implemented*

Executes a command in the ESBox's terminal.

**GetTerminalOutput**

*Immediate Response: SendTerminalOutput*
*Not currently implemented*

Requests the ESBox to send its current terminal output buffer.

**GetFileList**

*Immediate Response: SendFileList*
*Not currently implemented*

Requests the ESBox to send a list of files in the nominated directory.

**UploadFile**

*No Response*
*Not currently implemented*
Requests the ESBox to disconnect the current session, download a file at the specified URL to the specified path, and then reconnect to the ESCo.

**GetFile**

*No Response*
*Not currently implemented*
Requests the ESBox to send a specified file from its filesystem to the ESCo.

### ZigBee Network Management Messages

**GetDeviceList**

*Immediate Response: SendDeviceList*
Sent to request that the ESBox send the current device list to the ESCo. If this Message is issued with the *Detailed* key, the response will include a representation of the network that includes information on existing binds and report configurations. See the ExampleMessages_1_0 file for a full example of each type of a standard and detailed device list report.

**RequestDeviceToLeave**

*No Response*
*Not currently implemented*
This message is used to remotely request a device to leave the ZigBee network. In the case of Saturn South devices (e.g. Mini Smart Meter, Mini CT Meter) this will cause the device to return to ‘factory reset’ state.

**LocateDevice**

*No Response*
*Not currently implemented*
This message will instruct the ESBox to place a device into Identify Mode. This usually makes the chosen device blink with a known pattern for the purposes of identifying the device that has already been installed and joined to the Zigbee network.

**PermitJoining**

*No Response*
Used to remotely enable Permit Joining mode in the ZigBee network for a specified length of time. The time specified may be between 1 and 254 seconds. Specify 255 seconds to enable permit joining until the ESBox restarts.

**AddToGroup**

*No Response*
*Not currently implemented*
This message is used to add one or more ZigBee endpoints to a specific group.

**BindUnbind**

*No Response*
*Not currently implemented*
This message is used to set up or remove bindings between individual clusters or groups in the ZigBee network.

**ConfigureReporting**

*No Response*
*Not currently implemented*
This message is used to configure attribute reporting for a set of attributes on a specific cluster of a specific endpoint of a specific node in the network. Reported attributes will automatically be transmitted to any configured bind destination(s) for the corresponding cluster. If an empty attribute list is sent in this message, reporting will be unconfigured for this device.

**GetConfigurationTemplates**

*Immediate Response: SendConfigurationTemplates*
*Not currently implemented*
Request a list of current configuration templates from the ESBox.

**SetConfigurationTemplate**

*No Response*
*Not currently implemented*
Define a new configuration template for one device model number. Configuration templates are used to automatically set up bindings and attribute reporting for specific ZigBee products based on their manufacturer and model number definitions. See the section on Configuration Templates for more information.

**ClearConfigurationTemplates**

*No Response*
*Not currently implemented*
Clears all existing configuration templates.

## Database Management Messages

### GetAvailableData

*Immediate Response: SendAvailableData*
*Not currently implemented*
Request ESBox to send the time of the oldest and most recent datapoints in the database within the specified range. This is a good way of quickly assessing the extent of logged data on the ESBox.

### GetData

*Immediate Response: SendData*
*Not currently implemented*
Request data for the specified device from a specified start time to a specified end time. Guidelines for using this Message are included in the *Database Interaction* section of the Implementation Guidelines below.

### GetLatestReadings

*Immediate Response: SendLatestReadings*
This message is used to retrieve the most recent data reported to the ESBox by any devices

### ClearDatabase

*No Response*
*Not currently implemented*
This message is used to clear some or all data from the database on the ESBox.

### ConfigureDataLogging

*No Response*
*Not currently implemented*
This message is used to define a set of attributes for a specific cluster on a specific endpoint of a given node that should be logged to the ESBox database. Data that is logged to the database can be retrieved with the *GetData* Message. See the Implementation Guideline section on *Database Interaction* usage for more information.

## ESBox to ESCo

These messages to do not generate a related response from the ESCo.

**NoFurtherMessages**

*Sent when the outbound message queue of the ESBox is empty.*

**SendESBoxOptions**

*Sent in response to a GetESBoxOptions message.*
This message contains a copy of 'ESBox Options' as it was at the time the response was generated.

**SendDeviceList**

*Sent in response to a GetDeviceList Message.*
The fields returned in this message are described in the **Device List Entries** section below.

**SendAvailableData**

*Sent in response to a GetAvailableData Message.*
*Not currently implemented*
Sends a list of data available in the database.

**SendData**

*Sent in response to a GetData Message.*
*Not currently implemented*
Sends a set of recorded data for the single selected device. Only one *GetData* request will be responded to at any one time. Subsequent *GetData* requests made while the initial is pending will be silently discarded.

**SendLatestReadings**

*Sent in response to a GetLatestReadings Message*.
Sends the current state of the latest readings buffer at the time the message is generated.

**SendUpdateStatus**

*Sent in response to a GetUpdateStatus Message.*
*Not currently implemented*
Contains a summary of any updates that are currently in progress.

**SendConfigurationTemplates**

*Sent in response to GetConfigurationTemplates Message.*
*Not currently implemented*
Sends a list of current configuration templates from the ESBox.

**SendTerminalOutput**

*Sent in response to GetTerminalOutput Message.*
*Not currently implemented*
Sends the current terminal output lines with UID than that provided (or all lines if the provided UID was 0).

**SendFileList**

*Sent in response to GetFileList Message.*
*Not currently implemented*
Sends the requested directory listing.

**SendFile**

*Sent in response to GetFile Message.*
*Not currently implemented*
Sends the requested file to the ESCo.

# SS_LC (Saturn South Load Control Cluster)

**Cluster ID:** 64784
**Manufacturer ID:** 4278
The SS_LC cluster is used to interact with load control devices, schedule load dispatches, and to retrieve and manage stored data in the ESBox database.

## ESCo to ESBox

### WriteAttributes

*No Response*
This message is used to set one or more attributes of a particular cluster on a particular endpoint of a particular node.

### ReadAttributes

*No Response (result placed in LatestReadings buffer)*
This message is used to read one or more attributes of a particular cluster on a particular endpoint of a particular node. There is no response generated for this command; instead, when the requested attribute is sent to the ESBox, the ESBox will add the attribute to the LatestReadings buffer, from where it can be retrieved by the ESCo using the *GetLatestReadings* Message in the SS_ESB cluster.

### GetWaveform

*No Response (result placed in LatestReadings buffer)*
*Not currently implemented*
Request a waveform sample from the chosen device.

### BroadcastDispatch

*No Response*
This command is used to instruct one or more devices to immediately ‘dispatch’ - that is, to change the state of their primary switching element (e.g. relay) to the opposite of the currently configures safe state. This command can be sent with the optional ‘Revert’ flag, which will instruct the nominated devices to begin their delayed ‘return to safe state’ procedure.

## ESBox to ESCo

### DispatchReport

*Sent automatically.* **(multiple messages may accrue in the output queue)**
This Message is used to report that a device has successfully responded to a *BroadcastDispatch* command. This Message contains the IEEE address of the device that has successfully delivered a dispatch report to the ESBox.

### UFLSReport

*Sent automatically.* **(multiple messages may accrue in the output queue)**
This Message is used to report that a devices has automatically disconnected based on its Under-Frequency Load Shedding threshold settings. This Message contains the IEEE address of the device that has successfully delivered a UFLS disconnection report to the ESBox. The detected frequency at disconnect time is also reported.

# OnOff (ZigBee On/Off Cluster)

**Cluster ID:** 6
**Manufacturer ID:** 0
The ZigBee On/Off cluster is a ZigBee Cluster Library cluster for controlling the state of the primary switchable element of a device.

## ESCo to ESBox

### WriteAttributes

*No Response*
Note: This Message is not meaningful in this cluster, as the only supported attribute in this cluster (OnOff) should be set as result of the use of the SwitchState Message, not by directly setting the attribute value.

### ReadAttributes

*No Response (result placed in LatestReadings buffer)*
This message is used to read one or more attributes of a particular cluster on a particular endpoint of a particular node. There is no response generated for this command; instead, when the requested attribute is sent to the ESBox, the ESBox will add the attribute to the LatestReadings accumulator, from where it can be retrieved by the ESCo using the *GetLatestReadings* Message in the SS_ESB cluster.

### SwitchState

*No Response*
Used to change the state of the primary switchable element of a device. Can set the state to ON, OFF, or toggle the state.

## ESBox to ESCo

None.

# All other Clusters

**Cluster ID:** [any]
**Manufacturer ID:** [any]

Attributes can be written to and read from any attribute of any cluster using the generic *WriteAttributes* and *ReadAttributes* Messages. As cluster specific commands are implemented in this specification for other clusters, corresponding sections will be added to this API documentation.

# ESBox Options

'ESBox Options' refers to the set of read/writable and read-only settings stored locally on the ESBox that govern the operating conditions of the ESBox. Items that are included in the ESBox Options list, but which have no effect or are currently disabled, are marked as **[no effect]**.

The read/writable settings are listed below:

| Setting Name | Description |
|---|---|
| PollESCoInterval | The period, in seconds, between connection attempts made by the ESBox to the ESCo. (Including time spent on the previous connection)<br>The minimum safe value for this setting is ¼ of *ESCoTimeout*.<br>In long polling mode, the typical value for this setting is equal to *ESCoTimeout*. |
| ESCoTimeout | Number of seconds to wait to receive a valid API message from the ESCo before dropping an active connection. If the ESBox is sending a large amount of data (using HTTP chunked encoding), this timeout applies to outbound chunks.<br>**A typical safe value for this setting in normal mode is 10 seconds.**<br>A typical safe value for this setting in Long Polling mode is 10 seconds + the expected maximum server long poll interval.<br>**The maximum value for this setting is 107 seconds.** |
| Primary ESCoAddress | Primary address to use to connect to the ESCo. |
| Secondary ESCoAddress | Secondary ESCo address to try if the ESBox is unable to successfully communicate with the Primary ESCo three times in a row. |
| (Primary/Secondary)ESCoPath | Path to use to access the ESCo. |
| (Primary/Secondary)ESCoPort | Port to use to access the ESCo. If this is set to 443, HTTPS will be used, else HTTP. |
| ProxyAddr | Address of proxy server. If no proxy server is being used, set this value to an empty string.<br>**[no effect]** |

| | |
|---|---|
| ProxyPort | Proxy server port. **[no effect]** |
| ProxyUsername | Proxy server username, if proxy server requires authentication via the Proxy-Authenticate header. If this value is set to an empty string, the Proxy-Authenticate header will not be sent in outgoing messages to the proxy server. **[no effect]** |
| ProxyPassword | Proxy server password, if proxy server requires authentication via the Proxy-Authenticate header **[no effect]** |
| EnableAsynchronousContainers | Flag that determines if the ESBox will be permitted to send certain high-priority containers asynchronously. Defaults to False. **[no effect]** |
| AutomaticallySendLatestReadings | If this flag is set to 1, a SendLatestReadings message will be automatically generated by the ESBox and sent in the first transmission to the ESCo when it connects (i.e. every *PollESCoInterval* seconds). If set to 0, SendLatestReadings will only be sent in response to a GetLatestReadings Message from the ESCo. |
| NoESCoCommsSafeStateTimeout | If no valid API messages are received from the ESCo after this many seconds, the ESBox will instruct all connected Saturn South Demand Management devices to return to their configured Safe States. |
| CurrentTime | Defines the current system time. This should be maintained within 15 seconds of the ESCo time for most applications. |
| DbMaxReportedEsboxTimeDiff | The maximum difference between the current ESBox time and the time reported by a datapoint from an End Device. If a received datapoint exceeds the threshold, then it is marked as discarded.<br>Note that this settings currently only affects datapoints received from some SSLC End Devices. |
| DbCreateNewFileThreshold | If a datapoint's timestamp is more than *DbCreateNewFileThreshold* seconds after the previous datapoint, then a new datapoint file |

| | |
|---|---|
| | will be created. This setting can be used to tune database performance for extreme high and low loads. |
| DbWriteDiscardedDatapoints | Integer; if non-zero, datapoints received out of order will be discarded. If zero, these datapoints will be recorded in the order received (but marked as discarded datapoints). |

The read-only entries in the *ESBox Options* are listed below:

| Name | Description |
|---|---|
| TotalUptimeMs | The total number of milliseconds this ESBox has been running since production. |
| ThisUptimeMs | The number of milliseconds this ESBox has been running since the last restart. |
| LastGoodESCoAddress | Stores the last known good ESCo address (i.e. the address of the last ESCo from which this ESBox received a valid API message). The ESBox will attempt to use this ESCo address if it fails to contact the Primary and Secondary ESCo three times each. |
| LastGoodESCoPath | Stores the last known good ESCo path. |
| LastGoodESCoPort | Stores the last known good ESCo port. |
| NumReboots | Stores a tally of all startups of this ESBox. |
| NumSoftReboots | Stores a tally of startups of this ESBox that were purposefully triggered by the ESBox software. |
| NumWatchdogReboots | Stores a tally of the startups of this ESBox that were triggered by the system watchdog. |
| NumCmdsProcessed | Stores a tally of the total number of ESCo commands that have been received and successfully processed by the ESBox. |
| NumCmdsFailed | Stores a tally of commands received from the ESCo that have failed (e.g. due to bad arguments, impossible requests, etc). Not yet fully implemented. |
| NumCmdsUnrecognised | Stores a tally of commands received from the ESCo that were unrecognised. |
| NumHTTPConnectionsFailed | Stores a tally of HTTP connections that have been closed unexpectedly. |
| NumSSLConnectionsFailed | Stores a tally of SSL connections that have been closed unexpectedly. |

# Device List Entries

The following is a list of attributes that are reported for each node that in the *SendDeviceList* Message from the ESBox. The ESCo can request either a standard device list or 'detailed' ESBox from the device list. To request the 'detailed' device list, the ESCo should add the `M.F.Nwk.Detailed` key to the *GetDeviceList* request Message. Items that are only included in the 'detailed' device list report are marked with a *. **Detailed device list elements are not implemented in this release, and will be added in the near future.**

| Attribute Name | Description |
|---|---|
| DeviceIEEE | The IEEE address of the device. |
| ModelIdentifier | The reported model identifier of the device, if present. Defaults to "Unknown" |
| ManufacturerName | The reported manufacturer name of the device, if present. Defaults to "Unknown" |
| LocationDescription | The custom location description for the device, if set. |
| LastContact | The time (in seconds since beginning of epoch) since the device last communicated to the ESBox |
| JoinTime | The time (in seconds since beginning of epoch) that the device first joined this ESBox's network |
| TimeSinceRejoin | The time (in seconds since beginning of epoch) since the device last joined the network (e.g. following a reboot / update) |
| OnlineStatus | Indicates whether the ESBox consider this device to be online or offline. Valid values are "Online" or "Offline". |
| NodeType | Indicates what type of node this is. Valid values are "Zc" (ZigBee Coordinator), "Zr" (ZigBee Router), or "Zed" (ZigBee End Device) |
| *OTAStatus** | A dictionary containing information about any ongoing OTA operations for this device. This dictionary is only present if there is an OTA update in progress for this device. |
| *Children** | A list of children of this device, and the receive RF link quality of each child. |
| *Endpoints** | List of Endpoints on this device (node) that have configured database, binding, or report settings. |

The *OTAStatus* dictionary contains the following:

| Attribute Name | Description |
|---|---|
| PercentCompleted* | The percentage of the OTA update process that is complete. |
| TimeRemaining* | The expected time remaining for the current OTA process in seconds. |

Each item in the list of *Children* contains the following:

| Attribute Name | Description |
|---|---|
| DeviceIEEE* | The IEEE address of this child. |
| LQIToChild* | The most recent receive RF link quality of this child from its parent. |
| LQIFromChild* | The most recent receive RF link quality from this child to its parent. |

Each item in the list of *Endpoints* contains the following:

| Attribute Name | Description |
|---|---|
| EndpointID* | The ID of this endpoint. |
| LoggedAttributes* | A list containing structures that define which attributes from which clusters should be logged to the ESBox database. |
| BindConfiguration* | A list containing structures that define which bindings are currently configured for this endpoint. |
| ReportConfiguration* | A list containing structures that define which attributes are set up to be reported for a specific cluster, and under what conditions. |

Each item in the list of *LoggedAttributes* contains the following:

| Attribute Name | Description |
|---|---|
| ClusterID* | The ID of the cluster to which the attributes described in the next field belong. |
| LogAttributeIDs* | A list of attribute IDs denoting the attributes that will be stored in the ESBox database. |

Each item in the *BindConfiguration* list contains the following:

| Attribute Name | Description |
|---|---|
| TargetIEEE* | The IEEE address of the target of this bind. |
| BindType* | Defines the type of bind. Valid values are “Cluster” or “Group”. |
| TargetEndpoint* | The endpoint on the target node. (Note: This is omitted if the *Type* field is set to ‘Group’) |
| ClusterOrGroupID* | Contains the cluster ID if *Type* is set to ‘Cluster’, else contains the group ID. |

Each item in the *ReportConfiguration* list contains the following:

| Attribute Name | Description |
|---|---|
| ClusterID* | The ID of the cluster from which attributes are set up to report. |
| MinInterval* | The minimum reporting interval for the specified attributes, in seconds. |
| MaxInterval* | The maximum reporting interval for the specified attributes, in seconds. |
| *AttrList** | A list of attributes to report, with accompanying ‘minimum change’ values. |

Each item in the *AttrList* list contains the following:

| Attribute Name | Description |
|---|---|
| AttrID* | The ID of the attribute. |
| MinChange* | The minimum change to the value of this attribute that is required to trigger a report. |

# Implementation Guidelines

This section provides a set of recommendations for ESCo implementations, and discusses a number of different approaches for managing the server depending on the requirements of the application.

## What is required to implement an ESCo server?

### A Basic ESCo

Depending on the nature of the application, developing an ESCo can potentially be very straightforward. An example of the simplest possible application would be:

- Only Saturn South devices are being used in the network.
- Saturn South metering devices are being used to monitor power consumption, e.g. total load and solar generation at a residential premises.
- ESBox is configured to connect to the ESCo every 30 seconds and report the most recent power readings to the ESCo.

ESBoxes deployed as part of this system would have their *PollESCoInterval* set to 30 seconds, and would have the *AutomaticallySendLatestReadings* flag set in ESBox options. All Saturn South ESBox devices come with Configuration Templates pre-installed that automatically configure joined Saturn South meters to report their basic power metrics to the ESBox every 10 seconds.

Every 30 seconds the ESBox would contact the ESCo at the configured *PrimaryESCoAddress*, *PrimaryESCoPath* and *PrimaryESCoPort*, and would send a *SendLatestReadings* Message to the ESCo in the initial HTTP Request containing a snapshot of the most recent data that has been received for each connected device. The ESCo would receive this message and process the transmitted data, responding to the ESBox with a *CloseConnection* message. The ESBox would then close the connection, and the cycle would repeat after 30 seconds.

All that would be required to implement this ESCo would be a basic web server and framework capable of examining the contents of an HTTP PUT Request and responding with the appropriate JSON packed *CloseConnection* Message in the HTTP Response. The server may elect to verify the authenticity of the connecting ESBox by examining the *Auth* component of the incoming ESBox Container, and should also verify that the *ProtocolVersion* of the inbound Container is correct before processing the contents.

## A More Flexible ESCo

A somewhat more advanced ESCo might be required for a system that:

- Contains energy metering devices from Saturn South, as well as other metering devices from third party vendors (e.g. another vendor's water meter).
- Must be configured to have meter readings stored in the ESBox database, to be extracted as required.
- Must have detailed network status information available for debugging purposes.
- Requires that the ESBox connects to the server at most once every 5 minutes.
- Must have the ESBox Options exposed to and changeable by the ESCo
- Is composed of many sites (e.g. hundreds) !thousands

ESBoxes that are deployed to the field as part of this system are pre-configured with the correct ESCo address/port/path settings.

Because the ESCo needs to be able to configure binding and reporting manually for devices that aren't Saturn South devices, the ESCo must implement the *GetDeviceList* Message to regularly determine which nodes are on the network. Information provided in the device list is also useful for network troubleshooting purposes, and might be made available through the admin interface of the ESCo's web service.

By including the *Detailed* flag in the *GetDeviceList* Message, the server can learn about any existing bindings or report configuration settings for the devices that already exist in the network, and compare this against a set of policies that are manually entered into the ESCo based on information provided by device manufacturers. For example, Saturn South can provide documentation for the Mini Smart Meter that defines all endpoints, clusters, and available attributes on the device, and this information can be used to determine when and how the ESCo will use the *BindUnbind* and *ConfigureReporting* messages to configure each device to report the necessary meter readings to the ESBox.

Alternatively, the ESCo implementation might push some or all of these policies down to the ESBox itself, which can be done using the *SetConfigurationTemplate* Message. *SetConfigurationTemplate* allows the ESCo to define a set of standard bindings and attribute report settings that will be applied to any new devices of a particular model number and manufacturer name that join the network. Two such configuration templates are defined in the ESBox by default; one for the SS9000 Mini Smart Meter, and one for the SS9007 Mini CT Meter. Existing templates present on the ESBox can be discovered using the *GetConfigurationTemplates* Message. Existing configuration templates can be removed with *ClearConfigurationTemplates* and replaced with successive use of the *SetConfigurationTemplate* Message. Once these configuration templates are installed on the ESBox, the ESBox will automatically manage the configuration of the necessary binds and reporting configurations for matching devices. Note that when new templates are pushed to the ESBox, they are immediately applied to any matching devices that are already joined to the ESBox.

Now that the ESCo is taking the correct actions to configure metering devices to report data to the ESBox, the ESBox must be configured to log this data on its local database. For each device that must have logging enabled, the ESCo will issue *ConfigureLogging* Messages to the ESBox to define which set of attributes from a specific cluster of a specific endpoint of a specific device IEEE should be logged. The current logged attribute settings for a device are reported in the *SendDeviceList* Message from the ESBox, sent in response to a *GetLatestReadings* Message from the ESCo.

When the database logging settings are in place, attribute reports that arrive at the ESBox will begin to be stored in the ESBox's database. Every 5 minutes (*ESCoPollInterval* is set to 600 seconds) the ESBox will contact the ESCo on the configured address/port/path and deliver any pending messages to the ESCo (in normal operation, there should be no pending messages, and the first transmission will just contain a *NoFurtherMessages* Message that can be discarded by the ESCo), after which the server can request data from the ESBox database using the *GetData* Message. The ESCo would choose the parameters of the *GetData* Message to reflect the range of data that the ESCo needs to fill gaps in its local records, as well as within the best practice limits set out in the Database Interaction section of this Implementation Guide.

For debugging purposes, as well as to allow the ESCo to change various settings including the ESBox's *ESCoPollInterval* and Primary/Secondary ESCo Address/Port/Path, the server is also capable of sending the *GetESBoxOptions* and *SetESBoxOptions* Messages to the ESBox.

# Device List Polling

In the Home Automation profile, ZigBee devices are able to join a network whenever ‘Permit Joining’ mode is enabled in that network. Unlike in other profiles (e.g. Smart Energy), the network coordinator is not required to have any prior knowledge of joining devices, and joining devices are implicitly authenticated by the fact that ‘Permit Joining’ mode must have been enabled in order for the device to join.

Permit joining mode can be enabled by the ESCo using the *PermitJoining* Message, or by a user/installer at the site using the appropriate button press sequence on the ESBox. There is no explicit notification to the ESCo that a new device has joined the network; instead, the ESCo should regularly poll the ESBox for its device list using the *GetDeviceList* Message.

The *SendDeviceList* Message that is sent by the ESBox in reply to a *GetDeviceList* Message contains other useful information on the state of the network, including the last time each node contacted the ESBox, and what the last reported RF signal strength was for each node. Exposing this information through the ESCo’s admin interface is often an invaluable tool for network debugging and deployment troubleshooting.

# Retrieving Attributes

All data that is delivered to the ESBox by devices in the ZigBee network will be stored temporarily in the Latest Readings buffer on the ESBox. This is the case irrespective of why the attributes were sent to the ESCo - i.e. as the result of a regular report that has been set up for meter readings, or in response to an explicit *ReadAttributes* message from the ESCo.

In order to retrieve the latest attributes from the ESBox, the ESCo must issue a *GetLatestReadings* Message to the ESBox. The ESBox will respond immediately with a *SendLatestReadings* Message that contains the most recent new data for each attribute that has been reported to the ESBox in the interval since the last *SendLatestReadings* message was sent to the ESCo.

Data can also be saved to the ESBox’s local database by defining a set of a logged attributes with the *ConfigureLogging* Message. More detail on this message and other database interaction is provided in the next section.

# Database Interaction

*Note: Database features are currently not enabled in this release. Please contact Saturn South if you are planning an implementation that requires database functionality.*

The ESCo can configure the ESBox to log a set of attributes from a specific cluster of a specific endpoint of a specific device using the *ConfigureLogging* Message. Logging configuration settings are shown in the contents of *SendDeviceList* messages sent by the ESBox in response to *GetDeviceList* request Messages from the ESCo.

When attribute reports are received containing attributes that the ESBox is configured to log to its database, each attribute is sent to the Latest Readings buffer as well as to the database. Data is accumulated in the database for all time once logging is enabled. If the database reaches its maximum size, old datapoints are trimmed from the database and new data will continue to be added.

Data in the database can be retrieved by the ESCo using the *GetData* Message.

To get a quick summary of what is contained in the database, the *GetAvailableData* Message can be used. This will retrieve the time of the oldest and most recent datapoint in the database for the specified cluster, device and endpoint.

The entire database can be cleared using the *ClearDatabase* Message. This will clear all datapoints and device information for a specified cluster.