Bluetooth Mesh Models

Technical Overview

Bluetooth® technology is a wireless standard with agreed, formal specifications that support global interoperability between devices from different manufacturers. The same thinking went into Bluetooth mesh. Luminaires, sensors, switches, and other types of devices *just work* when installed in a state-of-the-art smart building, with interoperability assured.

Interoperability is a benefit of standardization across every layer of the entire communications stack — from the physical layer, dealing with the analogue world of radio at the bottom, to user level behaviors that products may exhibit at the top. The Bluetooth mesh specifications define those product behaviors in terms of granular, standard building blocks called *models*. This paper provides a guided tour of Bluetooth mesh models.

Author: Martin Woolley

Version: 1.0

Revision Date: 27 March 2019

Revision History

Version	Date	Author	Changes
1.0	27 March 2019	Martin Woolley	Initial Version

table of contents

table of contents (cont.)

back to contents

1.0 What is a Mesh Model?

As noted in the Bluetooth mesh glossary of terms, a model:

定义一组状态，状态转换，状态绑定，消息和其他相关行为。 节点中的元素必须支持一个或多个模型，并且它是定义元素具有的功能的模型。 有许多模型由蓝牙信号定义，其中许多模型被故意定位为“ 通用 ”模型，在各种设备类型中具有潜在的效用。

> **“...defines a set of States, State Transitions, State Bindings, Messages, and other associated behaviors. An Element within a Node must support one or more models, and it is the model or models that define the functionality that an Element has. There are a number of models that are defined by the Bluetooth SIG, and many of them are deliberately positioned as “generic” models, having potential utility within a wide range of device types.”**

The glossary and the Bluetooth Mesh Technology Overview are recommended reading if any of these terms are new to you.

Essentially, models are specifications for *standard software components* that, when included in a product, determines what it can do as a mesh device. Models are self-contained components and products will incorporate several of them. Collectively, from a network's point of view, models make the device what it is.

从本质上讲，模型是标准软件组件的规范，当包含在产品中时，它决定了作为网状设备它可以做什么。模型是独立的组件，产品将包含其中的几个组件。总的来说，从网络的角度来看，模型使设备成为现实。

1.1 State

模型包含状态。 状态是指示设备状况的数据项，例如开/关或高/低。 状态可能很简单，只包含单个值或复合，包含多个字段，类似于c等编程语言中的结构。

Models contain states. States are data items that indicate the condition of the device, such as on/off or high/low. States may be simple, containing only a single value, or composite, containing multiple fields, similar to a struct in programming languages like C.

In some cases, there are relationships defined between state items. These relationships are called *state bindings*. A state binding indicates that if one of the states in the relationship changes, then the other one needs to have its value recalculated. Sometimes state bindings are conditional and may be enabled or disabled by some other state. Developers must implement the required logic for any state bindings that are defined for the models they are using and ensure that logic is executed whenever required.

在某些情况下，状态项之间定义了关系。 这些关系称为状态绑定。 状态绑定表示如果关系中的某个状态发生更改，则另一个状态需要重新计算其值。有时状态绑定是有条件的，可能由某些其他状态启用或禁用。 开发人员必须为为其使用的模型定义的任何状态绑定实现所需的逻辑，并确保在需要时执行逻辑。

Conversely, where state bindings are not explicitly defined in the Bluetooth Mesh Model Specification, states must act independently. For example, if the generic on/off state indicates that a device is currently off, increasing the generic level state should have no user-discernible effect. Switching the device on by setting the generic on/off state to 1 should not only switch the device on, but it should begin functioning at the level that has been set. This can be readily understood if you consider a rotary dimmer switch that is rotated to change the level of the lights in the room but can also be pressed to switch them on or off. You can rotate the control when the lights are off and nothing will appear to happen, but if you then press the switch, with it in the same rotated position, the lights will come on at the selected level of brightness.

相反，在蓝牙网格模型规范中没有明确定义状态绑定的情况下，状态必须独立行动。 例如，如果通用开/关状态指示设备当前处于关闭状态，则增加通用级别状态应该没有用户可识别的效果。 通过将通用开/关状态设置为1来打开设备不仅应该打开设备，而且应该在已设置的级别开始运行。 如果你考虑旋转调光开关来改变房间里的灯光水平，但也可以按下来打开或关闭它们，这很容易理解。 您可以在灯熄灭时旋转控件并且不会发生任何事情，但是如果您按下开关，并且它处于相同的旋转位置，灯光将以所选的亮度级别亮起

back to contents

模型被分类为既不包含状态的客户端，也包括服务器。 state是用于数据项的术语，表示设备的某些方面所处的条件，例如它是打开还是关闭或者转到哪个级别。

1.2 Categories of Model

某些服务器模型与另一个服务器模型相关联，其名称相似但包含“ setup ”。 例如，传感器服务器模型具有关联的传感器设置服务器模型。 设置服务器模型在技术上与其他服务器模型没有什么不同，因为它们包含状态并生成和使用特定类型的消息。 它们的目的是允许将模型的配置设置与主模型状态项分离，以便可以应用不同的访问控制策略。 通常允许网络管理员通过其设置服务器模型配置模型的关联设置，但不允许标准用户执行此操作。

Models are classified as being either *clients*, which do not contain state, or *servers*, which do. State is the term used for a data item which represents the condition that some aspect of a device is in, such as whether it is on or off or what level it is turned up to.

Some server models are associated with another server model with a name that is similar but includes “SetUp” in it. For example, the *Sensor Server* model has an associated *Sensor Setup Server* model. SetUp server models are technically no different to other server models in that they contain a state and produce and consume particular types of messages. Their purpose is to allow the separation of a model’s configuration settings from the main model state items so that distinct access control policies can be applied. It is common to allow a network administrator to configure a model’s associated settings via its SetUp Server model but not allow standard users to do this.

1.3 Model Communication and Behaviors

模型通过发送和接收消息相互通信。 有许多类型的消息，这些消息被定义为每个模型的规范的一部分，因此很清楚模型可以产生什么类型的消息以及它可以接收和理解的消息类型

Models talk to each other by sending and receiving messages. There are numerous types of message, and these are defined as part of the specification for each model so that it is clear what types of message a model can produce and what types of message it can receive and understand.

Messages either communicate a state value to other devices or they change a state value, eliciting a response, often visible, from a device. 消息或者将状态值传递给其他设备，或者它们更改状态值，从设备中引出通常可见的响应。

Models defined by the Bluetooth Special Interest Group (SIG) in the Bluetooth Mesh Model Specification are known as Bluetooth SIG models. Vendors may define their own models too, and these are known as vendor models. Vendor models should be used with caution and only when there is no possible way to use Bluetooth SIG models to meet the requirements.

由蓝牙网格模型规范中的蓝牙特殊兴趣小组（sig）定义的模型被称为蓝牙sig模型。 供应商也可以定义自己的模型，这些模型称为供应商模型。 只有在没有可能的方法使用蓝牙sig模型来满足要求时，才应谨慎使用供应商模型

模型可以在其他模型上具有指定的依赖关系。 模型可以扩展另一个模型，即第一个模型将状态添加到第二个模型的过程。 模型还可能需要存在扩展它的模型。 不扩展其他模型的模型称为根模型。

Models can have specified dependencies on other models. A model may extend another model, a process whereby the first model adds states to the second model. A model may also require that a model which extends it be present. Models that do not extend other models are known as *root models*.

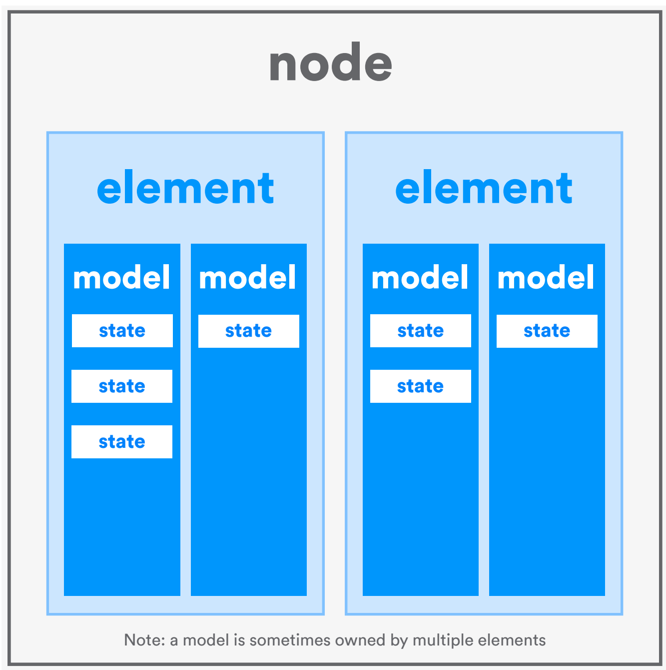

Figure 1 — Node Composition

1.4 Software Developers and Bluetooth Mesh Models

软件开发人员应该很容易将模型规范想象成类似于面向对象（oo）软件工程范例中的类以及作为模型或对象实例的设备内部代码中的模型实现。

Object Orientation

Software developers should find it easy to imagine model specifications as being akin to classes in the object-oriented (OO) software engineering paradigm and model implementations in code inside a device as an instance of the model or *object*.

The Bluetooth mesh specifications do not stipulate any particular approach to implementing models in code; that’s left to the developer and

蓝牙网格规范没有规定在代码中实现模型的任何特定方法；这是由开发人员和编程语言和apis使用。 但是模型确实适用于面向对象的方法，规范甚至讨论了一个扩展另一个模型的模型，这个概
也让人联想到oo。

the programming language and APIs in use. But models do lend themselves to an object-oriented approach, and the specification even talks about one model *extending* another, a concept that is also reminiscent of OO.

SDK Variations

有许多用于开发网格固件的sdks（软件开发工具包）。 一些来自蓝牙模块供应商，专门为其模块创建代码。 其他的，例如zephyr rtos sdk，是硬件无
的，允许为许多不同的目标板创建固件。 此时，zephyr支持100种不同的目标板。

There are a number of SDKs (software developer kit) for developing mesh firmware. Some are from Bluetooth module vendors that are specific to creating code for their modules. Others, such as the Zephyr RTOS SDK, are hardware agnostic and allow for the creation of firmware for numerous different target boards. At this time, Zephyr supports 100 different target boards.

Whatever SDK you use, the principals involved in implementing mesh firmware will be the same. In this paper, code created with the Zephyr SDK will be presented as a way of illustrating points from a developer's point of view.

无论你使用什么sdk，实现mesh固件所涉及的原则都是一样的。 在本文中，使用zephyr sdk创建的代码将作为一种从开发人员的角度说明点
方式。

```
// models - an array of specific model definitions
static struct bt_mesh_model sig_models[] = {
        BT_MESH_MODEL_CFG_SRV(&cfg_srv),
        BT_MESH_MODEL_CFG_CLI(&cfg_cli),
        BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
        BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV,
generic_onoff_op,
                                              &generic_onoff_pub, NULL),
        BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_LEVEL_SRV,
generic_level_op,
                                              &generic_level_pub, NULL)};

// elements - contains arrays of SIG models and vendor models (none in
this case)
static struct bt_mesh_elem elements[] = {
        BT_MESH_ELEM(0, sig_models, BT_MESH_MODEL_NONE),
};

// node composition - contains an array of elements
static const struct bt_mesh_comp comp = {
        .elem = elements,
        .elem_count = ARRAY_SIZE(elements),
};
```

网格固件开发人员必须承担的第一个关键任务之一是定义其产品的网格节点组成。 这意味着在代码中定义节点具有多少元素以及每个元素包含哪些模型。 第6页上的图1显示了节点，其元素，元素中包含的模型以及每个模型包含的状态项之间的关系。

Node Composition

One of the first key tasks a mesh firmware developer must undertake is to define their product's mesh node composition. This means defining in code how many elements the node has and what models each of the elements contains. Figure 1 on page six shows the relationships between the node, its elements, the models contained within elements, and the items of state that each model contains.

Details will vary across SDKs, but using the Zephyr SDK node composition involves creating a series of arrays, each of which contains structs defined by macros that the SDK provides. It might look something like the example shown in the code block shown above that shows four models belonging to an element, which is the sole element of the node.

详细信息将因sdks而异，但使用zephyr sdk节点组合涉及创建一系列数组，每个数组都包含由sdk提供的宏定义的结构。 它可能看起来像上面显示的代码块中显示的示例，它显示属于元素的四个模型，它是节点的唯一一元素

Properties

数据项可以在蓝牙网格模型中采用两种形式

There are two forms that data items can take in a Bluetooth mesh model.

State values are members of particular models and have a value with a meaning that the specification defines. They are not self-describing, and the state a message relates to is inferred from the opcode of the message.

状态值是特定模型的成员，并且具有规范定义的含义值。 它们不是自描述的，并且消息所涉及的状态是从消息的操作码推断出来的

另一方面，属性是在给定上下文中要解释的特征的实例。

Properties, on the other hand, are instances of *characteristics* to be interpreted in a given context.

Characteristics are also used with *GATT*. A characteristic defines the fields its value consists of, such as permissible values and their meaning and, in the case of GATT, includes an explicit type identifier in the form of a UUID (universally unique identifier). When used in GATT, characteristics are members of *services*, and the service that owns a characteristic provides a context within which to interpret and work with the characteristic. For example, the Alert Level characteristic can be a member of either the Link Loss service or the Immediate Alert service. The meaning of the characteristic varies depending on which service it is a member of, and this is defined in the GATT service specification.

Bluetooth mesh does not use GATT services. Instead, properties provide context for interpreting a related characteristic:

gatt也使用了特征。 特征定义其值包含的字段，例如允许值及其含义，并且在gatt的情况下，包括uuid形式的显式类型标识符（通用唯一标识符）。 当在gatt中使用时，特征是服务的成员，并且拥有特征的服务提供了解释和使用特征的上下文。例如，警报级别特征可以是链路丢失服务的成员或者 即时提醒服务。 特征的含义取决于它所属的服务，这在gatt服务规范中定义。 蓝牙网格不使用gatt服务。 相反，属性提供了解释相关特征的上下文：

> **"The Temperature 8 Characteristic is a type which represents a temperature measurement, has a format of uint8, and uses units of 0.5 degrees Celsius. Several properties are defined for this characteristic, thus allowing it to be interpreted in various contexts. The Present Indoor Ambient Temperature property indicates that the Temperature 8 characteristic should be interpreted as being a measurement which was taken indoors, whereas the Present Outdoor Ambient Temperature property relates to measurements taken outdoors, and the Present Ambient Temperature property is not specific about the type of location, and this is left to be derived from other location properties."**

“ 温度8特性是表示温度测量的类型，具有uint8的格式，并且使用0.5摄氏度的单位。 为此特征定义了几个属性，从而允许在各种上下文中对其进行解释。 目前的室内环境温度特性表明温度8特性应该被解释为在室内进行的测量，而当前的室外环境温度特性与室外进行的测量有关，而当前的环境温度特性并不是特定于室内的位置类型，这是从其他位置属性派生的。 “

back to contents

属性由属性id明确标识。 在使用属性的模型中，属性id和属性值包含状态的值。 例如，传感器数据状态包含一对或多对属性id和相应的传感器值。

Properties are explicitly identified by a *Property ID*. In a model where a property is in use, the property ID and property value comprise the value of a *state*. For example, the *sensor data* state contains one or more pairs of property ID and a corresponding sensor value.

Properties allow the same model to be used with a wide range of data types, which, in the case of models like the *sensor server model*, is hugely advantageous since any type of sensor data can be handled and interpreted with respect to any context, provided a suitable property has been defined. Without this approach to describing and encapsulating data, many different types of sensor models would be required, or the sensor server model would need to have a large number of states for each of the different types of sensor data it might need to support.

属性允许相同的模型与各种数据类型一起使用，对于像传感器服务器模型这样的模型，这是非常有利的，因为任何类型的传感器数据都可以根据任何上下文进行处理和解释，前提是 已经定义了合适的属性。 如果没有这种描述和封装数据的方法，则需要许多不同类型的传感器模型，或者传感器服务器模型需要为其可能需要支持的每种不同类型的传感器数据具有大量状态。

Client and Server Decoupling

When implementing models, it is important to respect the fact that client models and server models must know nothing about each other's implementation details. For example, a server should not need to know or choose to exploit knowledge of the specific values that a client might be able to send. Each is a black box to the other.

Coding Models

除了指定哪些模型属于节点组成中的每个元素之外，开发人员还需要做些什么才能合并为其产品选择的模型？ 在某些情况下，什么都没有。 像健康服务器模型这样的某些模型是必需的（在节点的主要元素中，所有节点都有），而sdk可以提供完整的实现，这很容易合并到节点的组合中。

Apart from specifying which models belong to each element in node composition, what else do developers need to do to incorporate the models that have been selected for their product? In some cases, nothing at all. Some models like the *health server model* are mandatory (in the primary element of a node, which all nodes have), and the SDK may provide a complete implementation, which is easily incorporated in the node's composition.

In most other cases, a number of other steps will be necessary: 在大多数其他情况下，还需要许多其他步骤

```
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_GET BT_MESH_MODEL_
OP_2(0x82, 0x01)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_SET BT_MESH_MODEL_
OP_2(0x82, 0x02)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_SET_UNACK BT_MESH_MODEL_
OP_2(0x82, 0x03)
#define BT_MESH_MODEL_OP_GENERIC_ONOFF_STATUS BT_MESH_MODEL_
OP_2(0x82, 0x04)

// each array member contains opcode, min msg len, handler function
static const struct bt_mesh_model_op generic_onoff_op[] = {
        {BT_MESH_MODEL_OP_GENERIC_ONOFF_GET, 0, generic_onoff_get},
        {BT_MESH_MODEL_OP_GENERIC_ONOFF_SET, 2, generic_onoff_set},
        {BT_MESH_MODEL_OP_GENERIC_ONOFF_SET_UNACK, 2,
generic_onoff_set_unack},
        BT_MESH_MODEL_OP_END,
};
```

back to contents

1. RX Message Handler Functions

必须注册与每个模型关联的消息的操作码以及节点可能接收的消息（rx），并且实现
用于处理这些消息类型的一个或多个函数。 这是上面的zephyr代码中的样子。

The opcodes of messages associated with each model and which the node might receive (RX) must be registered and, one or more functions for handling those message types, implemented. Here's what that looks like in Zephyr code above.

模型接收的消息要么更改状态值（set），要么请求在状态消息（get）中报告特定状态的
当前值。 设置消息有两种形式：不需要响应（未确认）的那些以及需要在状态消息中发
回新状态值的消息。 术语集有时用于表示这两种变体中的任何一种。

Messages received by a model either change a state value (*set*) or request that the current value of a particular state be reported in a status message (*get*). Set messages come in two forms: those that do not require a response (unacknowledged) and those that require the new state value to be sent back in a status message. The term set is sometimes used to mean either of these two variations.

When handling state changes produced by set messages, developers must ensure that any defined and active state bindings are processed, recalculating other dependent state values as required.

处理set消息产生的状态更改时，开发人员必须确保处理任何已定义和活动的状态绑定
，并根据需要重新计算其他依赖状态值。

2. TX Message Producer Functions

Models almost certainly need to transmit (TX) messages as well as receive them. Functions that formulate mesh messages and use the appropriate API to send messages need to be written and their execution triggered by suitable events or device interactions, such as the user pressing buttons or turning knobs. Developers will be largely concerned with the *access layer* part of messages rather than those fields that are related to lower layers of the stack, though there can be exceptions. It may be necessary to explicitly increment the *SEQ* field to avoid having devices reject messages as forming part of a suspected replay attack, or the software framework may do this automatically.

模型几乎肯定需要传输（tx）消息以及接收消息。 制定网格消息并使用适当的api发送消息的函数需要写入并由适当的事
件或设备交互触发它们的执行，例如用户按下按钮或转动旋钮。 开发人员将主要关注消息的访问层部分，而不是那些与
堆栈的较低层相关的字段，尽管可能有例外。 可能有必要显式增加seq字段以避免设备拒绝消息作为可疑重放攻击的一部
分，或者软件框架可以自动执行此操作。

3. Bind Application Keys to Models

All mesh messages are encrypted and authenticated using AES-CCM. Header fields are also obfuscated to make network-pattern-analysis attacks difficult. Fields from upper layers of the stack are encrypted using an *application key*, and fields from lower in the stack are encrypted using a *network key*. This separates network and application security and allows nodes to perform network functions, such as the relaying of messages without needing or having the ability to decrypt the application payload of the message.

所有网状消息都使用aes-ccm进行加密和验证。 标头字段也被混淆，以使网络模式分析攻击变得困难。 使用应用
程序密钥对来自堆栈上层的字段进行加密，并使用网络密钥对堆栈中较低层的字段进行加密。 这分离了网络和应
用程序的安全性，并允许节点执行网络功能，例如消息的中继，而无需或无法解密消息的应用程序有效负载。

A good mesh software framework automatically secures messages through encryption and obfuscation, using the network and application keys established when the device was *provisioned*. But a node may have several application keys, and each must be associated with specific models through a process known as *key binding*. This ensures that the stack knows which application key to use with which types of message. Developers will almost certainly need to perform explicit application key binding in their code. On Zephyr, application key binding looks like this:

良好的网状软件框架使用在设置设备时建立的网络和应用程序密
钥，通过加密和混淆自动保护消息。 但是节点可能有多个应用
程序密钥，每个应用程序密钥必须通过称为密钥绑定的过程与特
定模型相关联。 这可以确保堆栈知道要使用哪种应用程序密钥
以及哪种类型的消息。 开发人员几乎肯定需要在他们的代码中
执行显式应用程序密钥绑定。 在zephyr上，应用程序键绑定看起
来像这样：

```
/* Bind to generic level server model */
err = bt _ mesh _ cfg _ mod _ app _ bind(net _ idx,
                                         addr,
                                         addr,
                                         app _ idx,
                                         BT _ MESH _ MODEL _ ID _ GEN _ LEVEL _ SRV,
                                         NULL);
```

back to contents

net_idx和app_idx是索引值，它引用节点在初始配置和配置时可能配备的一个或多个网络和应用程序密钥列表中的特定密

net_idx and *app_idx* are index values that reference specific keys from the list of one or more network and application keys that a node might have been equipped with when initially provisioned and configured.

Application key binding is the basis for access control in a Bluetooth mesh network. Issuing the network administrator with the application key bound to the *sensor setup* server model gives that user the ability to update that model's state and configure the associated *sensor server* model. Other users, not in possession of this application key, cannot configure the sensor setup server.

应用程序密钥绑定是蓝牙网状网络中访问控制的基础。 向网络管理员发布绑定到传感器设置服务器模型的应用程序密钥，使该用户能够更新该模型的状态并配置相关的传感器服务器模型。 没有此应用程序密钥的其他用户无法配置传感器设置服务器

back to contents

2.0 Overview of Mesh Models

The standard Bluetooth SIG models are defined in a dedicated specification called the Bluetooth Mesh Model Specification. In this specification, you will find extensive and rigorous information on each of the 52 standard mesh models.标准蓝牙sig模型在称为蓝牙网格模型规范的专用规范中定义。 在本规范中，您将在52种标准网格模型中找到广泛而严格的信息。

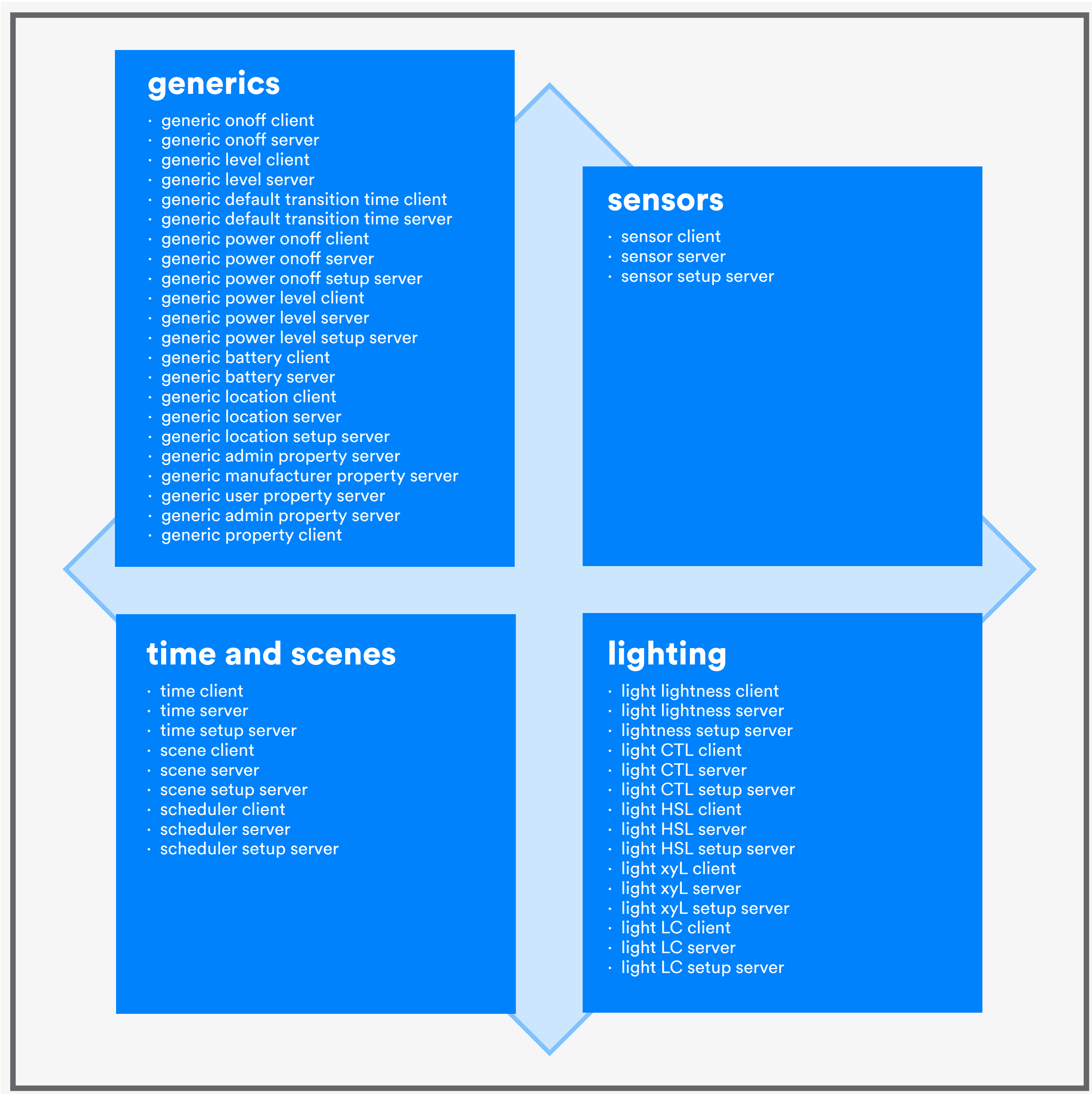

Figure 2 - The Bluetooth Mesh Models

back to contents

从图2中我们可以了解到关于网格模型的什么？首先，有四组模型：通用模型、传感器模型、照明模型以及与时间相关的模型以及称为场景的网格自动化功能。如果您查看图2中的列表，您还会发现每个客户机模型都有一个对应的服务器模型，反之亦然，并且一些服务器模型也有一个对应的安装服务器模型。

What can we learn about mesh models from Figure 2? First, there are four groups of models: the generics, models for sensors, models for lighting, and models concerned with time and a mesh automation feature called the *scene*. If you review the lists in Figure 2, you will also find that every client model has a corresponding server model and vice versa and that some server models have a corresponding setup server model too.

Generally, models are optional. Developers implement those models that equip their products with the mesh capabilities they need. But there are two models whose inclusion is mandatory and, collectively, these models are the heading of the *foundation models*.

一般来说，模型是可选的。开发人员实现这些模型，使他们的产品具备所需的网格功能。但是有两个模型是强制性的，这些模型是基础模型的标题。

3.0 A Guided Tour of Foundation Models

基础模型涉及启用蓝牙网状网络及其包含的设备的配置和管理。 有两套基础模型，这些在蓝牙网格剖面规范中有描述。

The foundation models are concerned with enabling the configuration and management of the Bluetooth mesh network and the devices it contains. There are two sets of foundation models and these are described in the Bluetooth Mesh Profile Specification.

3.1 The Configuration Server and Client Models

所有设备都需要配置。 因此，实施配置服务器模型是必需的，并为设备提供配置能力，通常使用将实现配置客户端模型的智能手机应用程序

All devices need to be configurable. Implementing the configuration server model is therefore mandatory and provides the device with the ability to be configured, typically using a smartphone application that will implement the configuration client model.

The configuration server model contains a significant number of states that allow various aspects of a device to be configured. The device's overall composition is held within a state called the *Composition Data* state. The destination address to use when publishing messages and other parameters relating to periodic message publication; the addresses subscribed to; and which, if any, of the special relay, friend, low power node, and proxy roles a device may play are all part of the configuration model's data.

配置服务器模型包含大量允许配置设备各个方面的状态。 设备的整体组成保持在称为组合数据状态的状态。 发布与定期消息发布有关的消息和其他参数时使用的目标地址；订阅的地址；设备可能播放的特殊中继，朋友，低功率节点和代理角色（如果有的话）都是配置模型数据的一部分。

Typically, developers only need to ensure the configuration server model is part of their device's firmware. The configuration data comes from a configuration client application, usually at the same time the device is provisioned to equip it with security cases. However, sometimes developers explicitly perform part of the device's configuration from within their code.

通常，开发人员只需要确保配置服务器模型是其设备固件的一部分。 配置数据来自配置客户端应用程序，通常在配置设备以配备安全案例的同时。 但是，有时开发人员会在代码中明确执行设备的部分配置。

3.2 The Health Server and Client Models

The health models are concerned with fault reporting and diagnostics. The primary element of all nodes in a Bluetooth mesh network must include the health server model. Other elements may inform the health server model of faults. A series of fault-related states, such as *current fault*, are defined for the health server model. Faults are represented by single octet codes. Some values in the available range are reserved for Bluetooth SIG use and others are for vendor specific codes. Table 4.2.1 in the Bluetooth Mesh Profile Specification identifies the standard fault codes defined by the Bluetooth SIG.

健康模型涉及故障报告和诊断。 蓝牙网状网络中所有节点的主要元素必须包括运行状况服务器模型。 其他元素可以通知健康服务器模型的故障。 为健康服务器模型定义了一系列与故障相关的状态，例如当前故障。 故障由单个八位字节代码表示。 可用范围内的某些值保留用于蓝牙sig，其他值用于供应商特定代码。 蓝牙网格剖面规范中的表4.2.1标识了蓝牙sig定义的标准故障代码。

back to contents

4.0 A Guided Tour of Generic Models

蓝牙网格模型的泛型集合被设计用于任何类型的设备，提供一组通用的，通用的功能。 如图3所示，有22个与8个状态相关的通用模型。

generics

- generic onoff client
- generic onoff server
- generic level client
- generic level server
- generic default transition time client
- generic default transition time server
- generic power onoff client
- generic power onoff server
- generic power onoff setup server
- generic power level client
- generic power level server
- generic power level setup server
- generic battery client
- generic battery server
- generic location client
- generic location server
- generic location setup server
- generic admin property server
- generic manufacturer property server
- generic user property server
- generic admin property server
- generic property client

Figure 3 - The Bluetooth Mesh Models

The generics collection of Bluetooth mesh models are designed to be used by any kind of device, offering a set of common, generally applicable capabilities. As Figure 3 shows, there are 22 generic models relating to 8 states.

4.1 The Generic OnOff Client and Server Models

At a Glance

The generic onoff models make it possible for one device to switch other devices on or off.

通用onoff模型使一个设备可以打开或关闭其他设备

About These Models

The server model contains one state only: the generic onoff state. This is a simple boolean state that indicates whether an element is currently switched on or off. A value of 0x00 means it is off, and a value of 0x01 means it is on. The generic onoff client may send generic onoff get, generic onoff set, or generic onoff set unacknowledged messages. It must be able to receive and handle generic status messages if it is able to send get or set (acknowledged) messages.

服务器模型仅包含一个状态：通用onoff状态。 这是一个简单的布尔状态，指示元素当前是打开还是关闭。 值0x00表示它已关闭，值0x01表示它已打开。 通用onoff客户端可以发送通用onoff get，通用onoff集或通用onoff set unacknowledged消息。 如果它能送获取或设置（确认）消息，它必须能够接收和处理通用状态消息

4.2 The Generic Level Client and Server Models

At a Glance

Some devices can be turned up or down; lights can be dimmed and the temperate of a room can be increased by turning up the thermostat. The generic level models allow control to be exercised over the level of other devices.有些设备可以打开或关闭；通过调高温控器可以调暗灯光，增加房间的温度。 通用级别模型允许在其他设备的级别上执行控制。

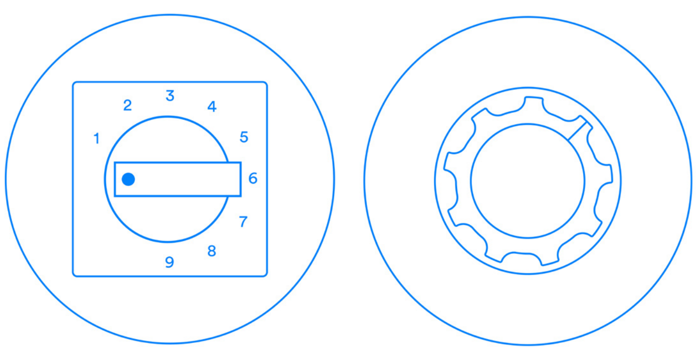

Figure 4a (left) - A 9 x Position Rotary Switch
Figure 4b (right) - Analogue Rotary Control

About These Models

The generic level server model contains a state called *generic level* that can be positive or negative and has a range of -32,767 to +32,767. 通用级别服务器模型包含一个称为通用级别的状态，可以是正数或负数，范围为-32,767到+32,767。

Different products may need to approach level control in different ways, such as from a user interface point of view. Imagine a 9-position rotary switch like the one in Figure 4a. 不同的产品可能需要以不同的方式接近水平控制，例如从用户界面的角度来看。 想象一下像图4a中那样的9位旋转开关。

考虑这种类型的液位控制器之间的相似点和不同点，它有9个固定的位置选择和一个模拟旋转控制器，可以在任何地方设置连续范围。

Consider the similarities and differences between this type of level controller, with its 9 fixed choices of position and an analogue rotary control that allows the position to be set anywhere through a continuous range.

在任何一种情况下，控件都需要实现通用级客户端模型。

In either case, the control needs to implement the generic level client model.

图4a中的固定位置控制必须将正的通用电平值范围划分为9个等间隔的频带，将其9个可选电平映射到由蓝牙网格模型规范定义的通用电平状态值。 描绘频带的值（0,3641,7282,10922,14563
04,21845,25485,29126）是在来自设备的通用级别集消息中发送的级别值。

The fixed position control in Figure 4a must divide the positive generic level value range into 9 equally spaced bands, mapping its 9 selectable levels to the generic level state values defined by the Bluetooth Mesh Model Specification. The values delineating the bands (0, 3641, 7282, 10922, 14563, 18204, 21845, 25485, 29126) are the level values sent in generic level set messages from the device.

It might be tempting to think that the rotary control in Figure 4b will not need to perform this kind of value mapping, but it too will deliver values at a certain level of granularity and magnitude to the firmware of the device it is a part of, and they will need mapping to the generic level state's value range in an appropriate way.

可能很容易认为图4b中的旋转控制不需要执行这种值映射，但它也会将粒度和幅度的某个值提供给它所属设备的固件， 并且他们需要以适当的方式映射到通用级别状态的值范围。

Changing Levels

Several ways of changing generic level are supported by the generic level models' set and set unacknowledged messages. 通用级别模型的set和set unacknowledged消息支持几种更改泛型级别的方法

Generic level set - changes the generic level state to an absolute value. 将通用级别状态更改为绝对值

通过相对，正数或负数更改通用级别状态

Generic delta set - changes the generic level state by a relative, positive, or negative amount.

Generic move set - initiates changing the generic level state in either a positive or negative direction and at a given speed. The speed with which the transition takes place is calculated from a *delta level* field in the message and a value known as the *transition time*. Transition time must either appear in the *generic move set* message itself, where it is an optional parameter, or be available in a state called the *generic default transition time,* which belongs to the *generic default* transition time model which may or may not be present. If transition time is not available from either of these two sources, the operation will not be executed and *generic level* will not be changed.

启动以正方向或负方向以给定速度更改通用级别状态。 根据消息中的增量级别字段和称为转换时间的值计算转换发生的速度。 转换时间必须出现在通用移动集消息本身中，它是可选参数，或者在称为通用默认转换时间的状态下可用，该状态属于可能存在或不存在的通用默认转换时间模型。 如果这两个源中的任何一个都没有转换时间，则不会执行操作，也不会更改通用级别。

The move transition may continue indefinitely. It will stop if a *move set* message with the *delta level* field set to zero is received. When *generic level* reaches its upper or lower limit, during a move transition, the implementation may decide to either terminate the transition at that point or take some other action, such as wrapping around and continuing.

移动过渡可能会无限期地持续下去。 如果收到delta级别字段设置为零的移动设置消息，它将停止。 当通用级别达到其上限或下限时，在移动转换期间，实现可以决定在该点终止转换或采取一些其他操作，例如环绕并继续。

Each of *generic level set*, *generic delta set*, and *generic move set* support the optional fields *delay* and *transition* time.

通用级别集，通用增量集和通用移动集中的每一个都支持可选字段延迟和转换时间。

The *delay* field allows the client to inform the server to defer execution of operation for a period of time after receiving the message. This can be helpful in synchronizing operations that affect multiple receiving devices.

延迟字段允许客户端在接收到消息之后通知服务器将操作的执行推迟一段时间。 这有助于同步影响多个接收设备的操作。

Transition time is used to calculate the speed with which a transition should take place. It encodes two data items, from which an elapsed time for the transition must be calculated. It is one octet in size and its 8 bits are used as follows:

转换时间用于计算转换应该发生的速度。 它编码两个数据项，必须从中计算过渡的经过时间。 它的大小是一个八位字节，其8位使用如下：

Field	Size (bits)	Definition
Default Transition Number of Steps	6	The Number of Steps
Default Transition Step Resolution	2	The resolution of the Default Transition Number of Steps field

Figure 5 - Transition Time (from the Bluetooth Mesh Model Specification)

The four values which *transition step resolution* may take represent 100 milliseconds (0b00), 1 second (0b01), 10 seconds (0b10), and 10 minutes (0b11), respectively. The transition time represented by this state is calculated by multiplying the *number of steps* and the time value represented by the step resolution. Durations from 0 seconds (immediate) to 10.5 hours can be encoded with the *transition time* state.

转换步骤分辨率可以采用的四个值分别代表100毫秒（0b00），1秒（0b01），10秒（0b10）和10分钟（0b11）。 通过将步数和步长分辨率表示的时间值相乘来计
该状态表示的转变时间。 可以使用转换时间状态对从0秒（立即）到10.5小时的持续时间进行编码。

The word “steps” might suggest that transitions should take place in a series of discreet increments/decrements. This is not the case. The steps and step resolution fields are solely there to allow the calculation of the elapsed time of the transition. How the change manifests itself in user-visible ways is a product issue, and how the transition takes place in code is an implementation detail.

“ 步骤 ” 这个词可能暗示过渡应该以一系列谨慎的增量/减量进行。 不是这种情况。 步骤和步骤分辨率字段仅用于计算转换的经过时间。 变更如何以用户可见的方式表现出来是一个产品问题，如何在代码中进行转换是一个实现细节。

Note that some level control requirements cannot be completely met by the simple, generic level models. Lighting is a case in point. Human perception of brightness in lights is not linear, and so more specialized models for controlling the brightness or *level* of lights are provided in a Bluetooth mesh network. We will review the lighting models later in this paper.

请注意，简单的通用级别模型无法完全满足某些级别控制要求。 照明就是一个很好的例子。 人类对灯光亮度的感知不是线性的，因此在蓝牙网状网络中提供了用于控制灯的亮度或水平的更专业的模型。 我们将在本文后面回顾一下照明模型。

4.3 The Generic Power OnOff Client, Server, and Setup Server Models

At a Glance

These models enable the initial state that a device is in immediately after powering up to be configured. For example, it may be preferable that the initial state of a device when powered up is that it is off, as indicated by a value of 0x00 in the generic onoff state. Alternatively, for another product, it may make more sense for the initial state to be on, with generic onoff set to 0x01.

这些模型可以在启动配置后立即启用设备所处的初始状态。 例如，可能优选的是，当通电时设备的初始状态是关闭，如通用开关状态中的值0x00所示。 或者，对于另一个产品，初始状态开启可能更有意义，通用开关设置为0x01。

About These Models

The generic power onoff server model has a single state, generic on powerup which has three values defined with meanings shown below in Figure 6 from the Bluetooth Mesh Models Specification.

通用电源开关服务器模型具有单个状态，在通电时是通用的，其具有由蓝牙网格模型规范中的图6中所示的含义定义的三个值。

Value	Description
0x00	Off. After being powered up, the element is in an off state.
0x01	Default. After being powered up. the element is in an on state and uses default state values.
0x02	Restore. If a transition was in progress when powered on, the element restores the target state when powered up. Otherwise, the element restores the state it was in when powered down. 恢复。 如果在通电时正在进行转换，则该元件在通电时恢复目标状态。 否则，该元素恢复掉电时的状态。
0x03-oxFF	Prohibited.

Figure 6 - Generic OnPowerUp states

This model has several relationships with other models. It extends the generic onoff server model, and it requires the generic power onoff setup model be present. The latter model extends both the generic power onoff server model and the generic default transition time server model. This is depicted in Figure 7. 这个模型与其他模型有几种关系。 它扩展了通用的onoff服务器模型，它需要存在通用的电源开启设置模型。 后一种模型扩展了通用电源开关服务器模型和通用默认转换时间服务器模型。 这在图7中描述。

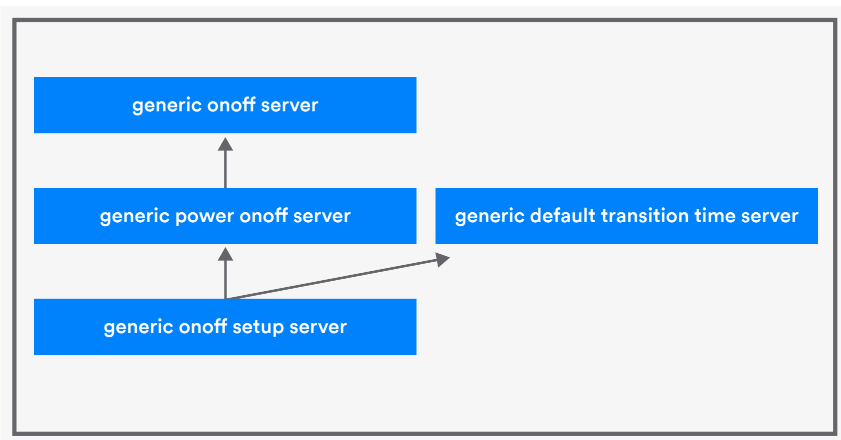

Figure 7 - Generic Power OnOff Server and Associated Models

It may not be obvious why the generic default transition time server is part of this picture. The *generic on powerup* state can be used to define what action to take when powering up, if a transition had been in progress when powering down. Therefore, since this behavior is not triggered by the receipt of a message, which could contain the transition time field, the generic *default transition time* state must be available for use in re-establishing transitions on power up.

为什么通用默认转换时间服务器是此图片的一部分可能并不明显。 如果在断电时正在进行转换，则可以使用通电状态下的通用状态来定义启动时要采取的操作。 因此，由于此行为不是由接收可能包含转换时间字段的消息触发的，因此通用默认转换时间状态必须可用于在加电时重新建立转换。

4.4 The Generic Power Level Client, Server, and Setup Server Models

At a Glance

These models allow control over a device element's power to be exercised. Through relationships with other models, such as the generic onoff server, generic level server, and generic power on server, various state bindings allow specific power levels to be established or re-established when the device is switched on or off or has its generic level state changed.

这些模型允许控制设备元件的功率。通过与其他模型的关系，例如通用OnOff服务器、通用级别服务器和通用电源开启服务器，各种状态绑定允许在设备打开或关闭或其通用级别状态更改时建立或重新建立特定的电源级别。

About These Models

Figure 8 depicts the relationships the *generic power level* server has with other models. It extends any model depicted with an arrow directed from this model to another model, directly or indirectly. It is extended by a model that has an arrow going to the *generic power level server.*

back to contents

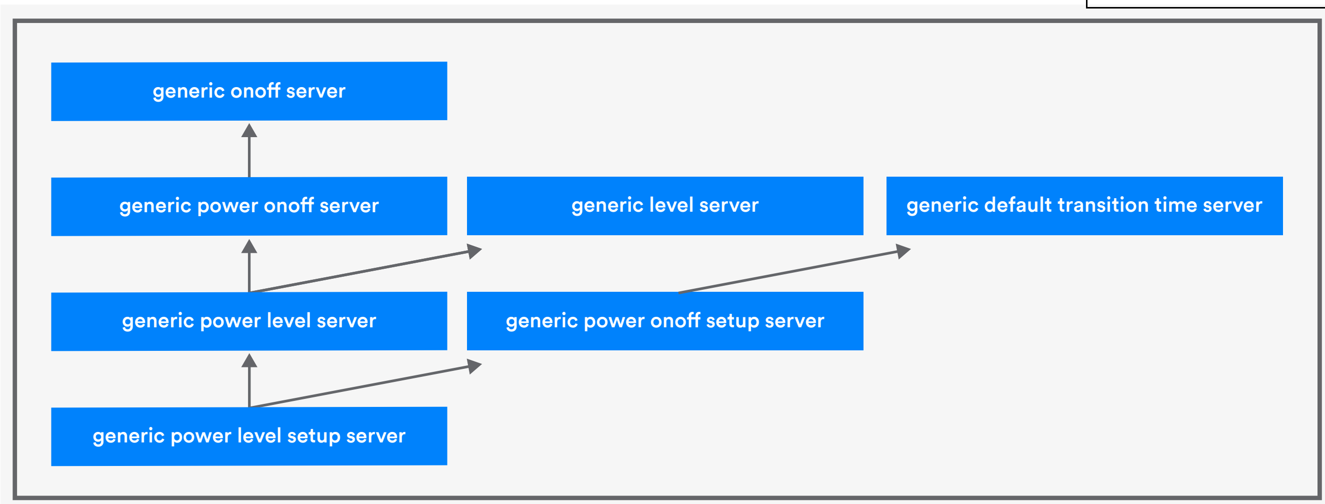

Figure 8 - Generic Power Level Server and Associated Models

理解这些模型的目的和行为的最好方法，尤其是通用的电源级别服务器模型，是了解服务器包含的状态。

The best way to understand the purpose and behavior of these models, especially the *generic power level server* model, is to understand the states the server contains.

通用电源级别服务器模型包含一个状态，即通用电源级别状态。它还从它扩展的模型继承了通用ONOFF和通用级别。

The *generic power level server* model contains one state, the *generic power level* state. It also inherits *generic onoff* and *generic level* from the models it extends.

Generic power level is a composite state, meaning it consists of a number fields, each of which is a state in its own right. These are shown and described in Figure 9 .

`通用功率级是一个复合状态，这意味着它由一个数字字段组成，每个字段本身就是一个状态。这些在图9中显示和描述。

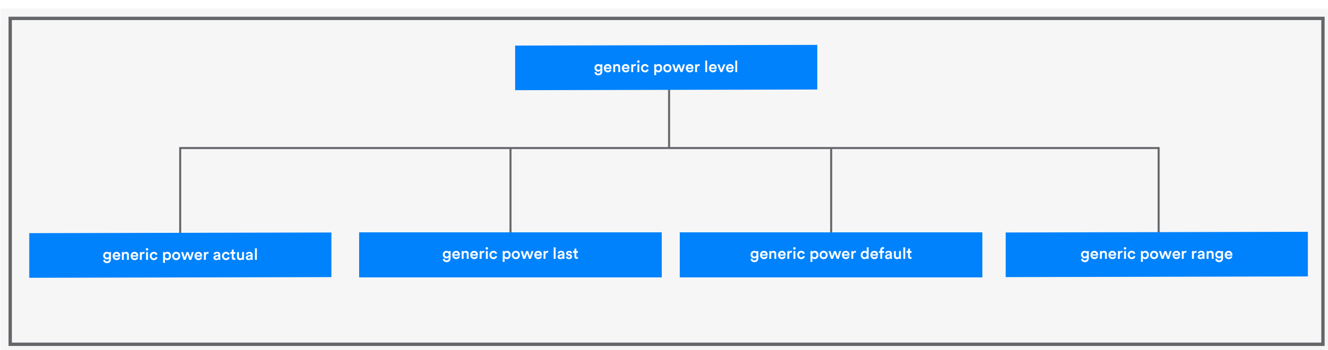

Figure 9 - The Composite Generic Power Level State

State	**Description** 将元素的功率级别设置为最大可用功率的线性百分比。请注意，当此状态设置为零时，允许设备继续保持足够的电源以使无线通信保持可用。这就像将设备置于待机模式。
Generic power actual	Sets element's power level as a linear percentage of the maximum available. Note that with this state set to zero, the device is permitted to continue to be sufficiently powered for wireless communication to remain available. It is like putting the device into standby mode.
Generic power last	Records the last known value of *generic power actual,* so the last power level can be restored when the device is switched on. This behavior is governed by a state binding with the *generic onoff* state and whether or not the *generic power default* state is zero.

记录最后一个已知的通用电源实际值，以便在设备打开时恢复最后一个电源级别。此行为由一个具有通用ONOFF状态的状态绑定控制，并且通用电源默认状态是否为零。

如果设备打开时此状态有一个非零值，并且通用的打开关闭状态更改为0x01，则功率级别将恢复到此状态的值。

State	Description
Generic power default	If this state has a non-zero value when the device is switched on and the generic onoff state changes to 0×01, the power level is restored to the value of this state.
Generic power range	Contains the minimum and maximum power levels the device can be set to as a percentage of the maximum level the device is capable of outputting. 包含设备可以设置为的最小和最大功率级别，以设备能够输出的最大级别的百分比表示。

State Bindings

A number of state bindings are defined, and these have a variety of behaviors to be achieved. Readers should review the [Bluetooth Mesh Model Specification] for full details and definitions of these state bindings. 定义了许多状态绑定，这些绑定有各种各样的行为需要实现。读者应该查看蓝牙网格模型规范，了解这些状态绑定的完整细节和定义。

They can be summarized as follows:

State	Bound to State	Description
Generic power actual	Generic level	Generic power actual = generic level + 32768
Generic level	Generic power actual	Generic level = generic power actual – 32768
Generic power actual	Generic onoff 根据通用onoff，通用功率last和通用功率默认状态的值的组合确定通用功率实际状态的值。 请参阅规格了解详情。	Determines the value of the generic power actual state depending on combinations of the values of the generic onoff, generic power last, and generic power default states. See specification for details.
Generic power actual	Generic onpowerup 在元件的物理加电期间确定通用功率实际状态的值。 取决于通用onpowerup，通用电源默认值，通用电源last和通用电源默认状态的值的组合。 请参阅规格了解详情。	Determines the value of the generic power actual state during the physical powering up of an element. Depends on combinations of the values of the *generic onpowerup*, generic power default, *generic power last*, and *generic power default* states. See specification for details.
Generic power actual	Generic power range	Establishes minimum and maximum values for generic power actual when it is not zero. 当非零时，确定通用功率实际的最小值和最大值。

4.5 The Generic Battery Client and Server Models

At a Glance 通用电池服务器模型表示由电池供电的元素。客户机模型可用于监控电池供电元件的状态。

The generic battery server model represents an element that is battery powered. The client model can be used to monitor the state of battery-powered elements.

About These Models

The *generic battery* server model is a root model that contains a single state that messages may act upon: the *generic battery state*. This state contains four values that provide information about a battery's current level, time to charge and discharge, and various other aspects of the battery, such as whether or not it is removable.

通用电池服务器模型是根模型，其包含消息可以作用于的单个状态：通用电池状态。 此状态包含四个值，提供有关电池电流水平，充电和放电时间以及电池的各种其他方面的信息，例如是否可拆卸。

The client and server models produce or consume *generic battery get* and *generic battery status* messages. The server must implement support for both message types. For the client, support for the *get* message is optional, and support for the status message is mandatory if *get* is supported, otherwise it is optional.

客户端和服务器模型生成或使用通用电池获取和通用电池状态消息。 服务器必须实现对这两种消息类型的支持。 对于客户端，对get消息的支持是可选的，如果支持get，则必须支持状态消息，否则它是可选的。

4.6 The Generic Location Client, Server, and Setup Server Models

At a Glance

Sometimes it is useful to know where a device is in your network. The generic location models allow you to do that. The generic location server model allows a node's location to be encoded in various ways and queried or reported to clients using associated messages. Want to know where a particular device is? Ask it.

有时，了解设备在网络中的位置很有用。 通用位置模型允许您这样做。 通用位置服务器模型允许以各种方式对节点的位置进行编码，并使用关联的消息查询或报告给客户端。 想知道特定设备在哪里？ 问它。

About These Models

The generic location models center around the *generic location* state, so that is a good place to start when looking at what these models make possible. The *generic location* state consists of a series of fields, which between them allow the following information about a node's location to be encoded.

通用位置模型围绕通用位置状态，因此在查看这些模型可以实现的内容时，这是一个很好的起点。 通用位置状态由一系列字段组成，它们之间允许有关节点位置的以下信息进行编码。

Global location - This is expressed as a longitude and latitude, using the WGS84 World Geodetic System and an altitude in meters above the WGS84 coordinates.

全球位置 - 使用wgs84世界大地测量系统和高于wgs84坐标的高度表示为经度和纬度。

Local location - This is expressed as a number of decimeters North and East, relative to some externally defined local coordinate system. A local altitude is also available, and this is a measure of altitude relative to the global altitude, also measured in decimeters.

本地位置 - 相对于某些外部定义的局部坐标系，表示为北和东的分数。 也可以使用当地高度，这是相对于全球高度的高度的度量，也以分米为单位测量。

Floor number - This field contains the floor number that the node is found on in a building. It is encoded in a special way, usually with a +20 delta. So, the encoded floor number value 22 represents the second floor in the building. Some special values are defined too. 0x00 represents floor *-20 or below*. 0xFC represents floor *232 or above*. The ground floor might either be floor 0 or floor 1 according to local conventions and the special values 0xFD and 0xFE represent these two possibilities.

楼层编号 - 此字段包含在建筑物中找到节点的楼层编号。 它以特殊方式编码，通常具有+20 delta。 因此，编码的楼层数值22表示建筑物中的第
层。 还定义了一些特殊值。 0x00表示-20或以下的楼层。 0xfc表示232或更高的楼层。 根据当地惯例，底层可能是0楼或1楼，特殊值0xfd和0xfe
这两种可能性。

Uncertainty - This field contains 16 bits of information. It can indicate whether the node is stationary or moving. If it is a mobile device, the time since its position was last updated is available. The precision of location measurements is also encoded in this field and ranges from 0.125 meters to 4096 meters.

不确定性 - 该字段包含16位信息。 它可以指示节点是静止还是移动。 如果是移动设备，则可以获得自上次更新位置以来的时间。 位置测量的精度也在该字段中编码，范围从0.125米到4096米。

The *generic location server* model requires the *generic location setup server* model to be present. The generic location setup *server model* allows the generic location state to be updated using generic location global set [unack] and generic location local set [unack] messages. The *generic location server* model supports get and status messages only, and so is effectively a read-only model. Devices that implement the generic location server model can either report their location on demand, when they receive a *generic location local get* message or a *generic location global get* message, or they can report it in a proactive way by publishing *generic location local status* and *generic location global status* messages.

通用位置服务器模型要求存在通用位置安装服务器模型。通用位置设置服务器模型允许使用通用位置全局集[未确认]和通用位置本地集[未确认]消息更新通用位置状态。通用位置服务器模型只支持获取和状态消息，因此实际上是只读模型。实现通用位置服务器模型的设备可以在收到通用位置本地获取消息或通用位置全局获取消息时按需报告其位置，也可以通过发布通用位置本地状态和通用位置全局状态消息以主动方式报告其位置。

back to contents

4.7 The Generic Default Transition Time Client and Server Models

状态更改可以是即时的，也可以在指定的时间段内发生。有两种方法可以启动非瞬时状态更改。许多消息类型支持一个称为转换时间的可选字段，如果包含在消息中，这将决定执行状态更改所需的时间。此外，可选的通用默认过渡时间服务器模型中可能提供的通用默认过渡状态也可以是状态更改的过渡时间信息源。

At a Glance

State changes can either be instantaneous or they can take place over a specified period of time. There are two ways in which a non-instantaneous state change can be initiated. Many message types support an optional field called *transition time* and, if included in a message, this will determine the time it takes for a state change to be executed. In addition, the *generic default transition state*, which might be available in the optional *generic default transition time server* model, can also be a source of transition time information for state changes.

About These Models

这些是简单的模型，定义了常见的get、set、set unacknowledged和status消息。这些消息所作用的通用默转换时间状态已经在通用级别模型的第（2）节中介绍和解释。

These are simple models with the usual get, set, set unacknowledged, and status messages defined. The *generic default transition time* state that these messages act upon has already been introduced and explained in the section (2) on the generic level models.

4.8 The Generic Property Client and Server Models

At a Glance

As explained earlier, the property models allow lists of arbitrary numbers of properties to be associated with a device. Properties are grouped in different models so that different user groups — the manufacturer, administrator, and standard user — only have access to permitted properties. It is also possible for a property server model to find a client that is capable of consuming and using a particular property type. Collectively, the property models provide a generalized data storage and communication mechanism that can accommodate a wide range of data values and types without models themselves needing to be changed.

如前所述，属性模型允许将任意数量的属性列表与设备相关联。 属性按不同的模型分组，以便不同的用户组（制造商，管理员和标准用户）只能访问允许的属性。 属性服务器模型也可以找到能够使用和使用特定属性类型的客户端。 总的来说，属性模型提供了一种通用的数据存储和通信机制，可以容纳各种数据值和类型，而无需更改模型本身。

About These Models

The specific models that deal with Bluetooth mesh properties are as follows:

处理蓝牙网格属性的具体模型如下

- Generic manufacturer property server
- Generic admin property server
- Generic user property server
- Generic property client
- Generic client property server

制造商，管理员和用户属性服务器拥有制造商，管理员和标准用户应具有某种级别访问权限的属性。 对这三种服务器模型中的每一种的访问由用户的客户端设备控制，该客户端设备需要拥有绑定到用户希望访问的模型的应用程序密钥。 对模型中特定属性的访问权限由属性状态中的字段控制，该字段确定由状态表示的属性是授予只读，只写还是读写访问权限。

The manufacturer, admin, and user property servers hold those properties to which manufacturer, administrator, and standard users should have some level of access. Access to each of these three server models is controlled by the user's client device needing to possess the application key bound to the model that the user wishes to access. Access to specific properties in a model is controlled by a field in the property state that determines whether read only, write only, or read-write access is granted by the property represented by the state.

The manufacturer, admin, and user property servers contain similar states called *generic manufacturer property*, *generic admin property*, and *generic user property*. Each has three fields containing the property ID, access flags (read, write, read-write), and the property value.

制造商，管理员和用户属性服务器包含类似的状态，称为通用制造商属性，通用管理属性和通用用户属性。 每个都有三个字段，包含属性id，访问标志（读，写，读写）和属性值。

The fourth server model, the *generic client property server* model, allows applications, such as the provisioning and configuration application, to find clients that are capable of consuming and processing particular properties. For example, it might be desirable to find a device with a user interface that can display a particular temperature property. The *generic client property server* contains a list of one or more *generic client property* states, each of which contains the ID of a property supported by the client.

第四服务器模型（通用客户端属性服务器模型）允许应用程序（例如供应和配置应用程序）查找能够使用和处理特定属性的客户端。 例如，可能希望找到具有可以显示特定温度特性的用户界面的设备。 通用客户端属性服务器包含一个或多个通用客户端属性状态的列表，每个状态包含客户端支持的属性的id。

back to contents

5.0 A Guided Tour of Lighting Models

Lighting can be surprisingly sophisticated and therefore needs specialized Bluetooth mesh models to meet its sometimes complex requirements. The Bluetooth mesh lighting models allow control over the on/off state of lights, their lightness, color temperature, and their color (using various color spaces). Importantly, they also provide a highly sophisticated software-based lighting controller that can enable smart lighting automation scenarios. As Figure 10 shows, there are 16 lighting models related to 5 distinct aspects of lighting.

Before beginning the guided tour of the models, consider the nature of lights and the various ways they can be controlled.

lighting

- light lightness client
- light lightness server
- lightness setup server
- light CTL client
- light CTL server
- light CTL setup server
- light HSL client
- light HSL server
- light HSL setup server
- light xyL client
- light xyL server
- light xyL setup server
- light LC client
- light LC server
- light LC setup server

Figure 10 -The Lighting Models

5.1 Lighting Overview

Controlling Lights

Lights are often controlled manually by pressing buttons, turning knobs, or pushing sliders. But they can also be controlled by sensors, indicating to the lights that there is someone in the room or that the ambient light level has become low because it is getting later in the day or because a cloud has obscured the sun. Lights can be controlled by timers too.

The generic onoff and generic level models detailed in section 4 could be used to control some of the basic attributes of a light, but people perceive lighting conditions in more complex ways, with brightness or *lightness* perceived according to a non-linear scale.

Lights have more attributes than their on/off state or their lightness that we might wish to control. Some lights can have their color controlled, and there are a number of ways of modelling color in lights.

Smart Lighting

Smart buildings require smart lighting. Smart lighting can be controlled by manual actions taken by building occupants, but, more importantly, a smart lighting system is informed by sensors and uses control algorithms to achieve self-optimizing behaviors that make the system efficient, cost effective, and pleasing to the people that use the building. The Bluetooth mesh lighting models include a particularly special set of models, such as the Light LC models that provide sophisticated, automated control of lights.

5.2 Lighting Concepts

To appreciate the lighting models, it helps to understand certain concepts from the world of lighting. The key ones are as follows:

Color Temperature

The color temperature of a light source is what leads people to describe colors as either *cool* or *warm*. It has a more scientific definition that relates to the temperature of the light radiated by the object, measured in Kelvin. Surprisingly, lower color temperatures are those we describe as *warm* and higher temperatures as *cool*. In commercial lighting applications, warmer color temperatures are often used to promote relaxation and cooler temperatures to enhance the concentration of occupants working in the room.

Color-Tunable Light

Color-tunable light (CTL) is a capability of some lights that allows color temperature to be controlled via two dimensions: lightness and color temperature.

Hue

Colored light has a number of properties, of which hue is one of the main ones. Typically, hue measures the angular position of a color in a color wheel.

Lightness

Lightness is the term used to refer to the perception of brightness.

Saturation

Saturation is another property of colored light and measures the ratio of an object's color to its lightness. A given color with a high lightness is said to be less saturated than the same color with low lightness.

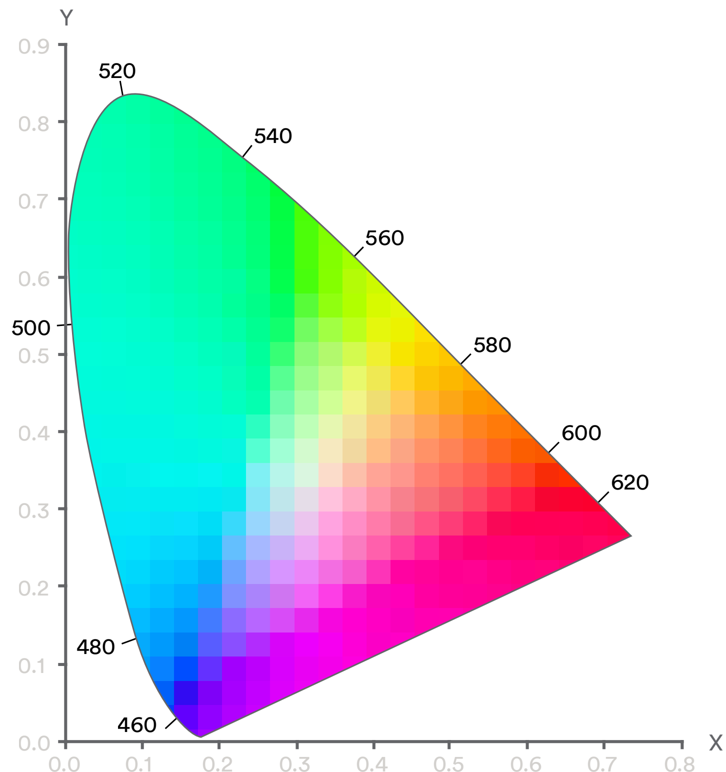

Figure 11 - CIE1931 is especially popular in professional lighting applications

Color Models

A color model, not to be confused with a Bluetooth mesh model, is a mathematical way of representing colors. There are several color models in popular use, each with its own strengths and weaknesses.

HSL (hue, saturation, lightness) represents colors using a cylindrical representation. The angular position in the circular cross section of the cylinder represents the hue, the distance from the center of this circle represents the saturation, and the distance from one end of the cylinder represents the lightness, with one end representing black and the other white.

The **RGB** (red, green, blue) color model is an additive model where given levels of red, green, and blue light are mixed to produce a color that people can perceive.

The **CIE1931** color space defines the mathematical relationships between wavelengths of light and perceived colors in vision. Just like RGB and HSL, colors in this model are defined by three values: x, y, and Y. x and y are coordinates of the color on a color chart, and Y measures the luminous intensity. CIE1931 is especially popular in professional lighting applications.

Each color model has an associated color space that is a set of colors that the model allows to be reproduced.

5.3 The Light Lightness Client, Server, and Setup Models

At a Glance

These models allow the lightness of a lamp to be controlled by mesh messages and events, such as powering up the device.

About These Models

Figure 12 depicts the relationships the light lightness server model has with other models. It extends any model depicted with an arrow pointing to it from this model directly or indirectly. It is extended by a model which has an arrow going to the light lightness server.

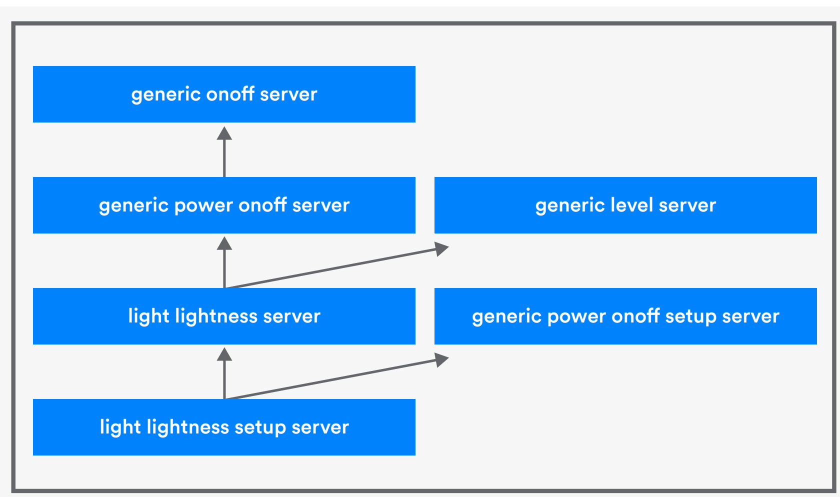

Figure 12 - Light Lightness Server and Associated Models

A number of states are involved in the control of lightness and contained within these models. Understanding these states will provide a good start to understanding the models.

The *light lightness* state is a composite state, consisting of the *light lightness linear*, *light lightness actual*, *light lightness last*, and *light lightness default* states.

There are two distinct ways that lightness may be changed using these models. The *light lightness linear* state provides control along a linear scale, but which people will perceive as non-linear lightness changes. Conversely, the *light lightness actual* state provides control along a non-linear scale that produces lightness changes perceived by people as being linear.

A range of supported lightness levels, from a minimum level to a maximum level, may be set for the server using its setup model, which contains the *light lightness range* state, a composite state that includes the light *lightness range min* state, and the li*ght lightness range max* state. The configured range is used in lightness state transitions to ensure only valid, supported values are used by the model.

In addition to states concerned with controlling lightness on a given scale, there are states concerned with restoring the lightness level when the device is switched back on or powered up. These are the *light lightness last* state and light *lightness default state*, both of which are involved in the functioning of the *generic power onoff server* model.

State Bindings

Light lightness actual and *light lightness linear* are related by two-way bindings. If one changes then the other must be recalculated.

Light lightness actual is also bound to the *generic level*, *generic onoff*, *generic onpowerup*, and *light lightness range* states. The precise details of these bindings are defined in the specification, but the general nature of these bindings should be intuitive enough. For example, changing the generic level in a light that has the *light lightness server* model will change its lightness states as well.

5.4 The LC Client, Server, and Setup Models

At a Glance

Collectively, the lighting control (LC) models form a lighting controller: a software component that allows sophisticated, sensor and user-driven lighting control to be set up. Occupancy and ambient light sensors are catered for so that techniques like daylight harvesting can be employed. As the state of the lighting controller changes, the *light lightness* state of the light under control progresses through a series of levels, with the transition from one to another governed by configurable timing parameters so that changes are not abrupt and feel natural to building users.

Decentralized Control

Legacy lighting control requires the installation of dedicated, physical devices, called controllers, sitting in between sensors and lights. This is called a centralized lighting control architecture. See Figure 13.

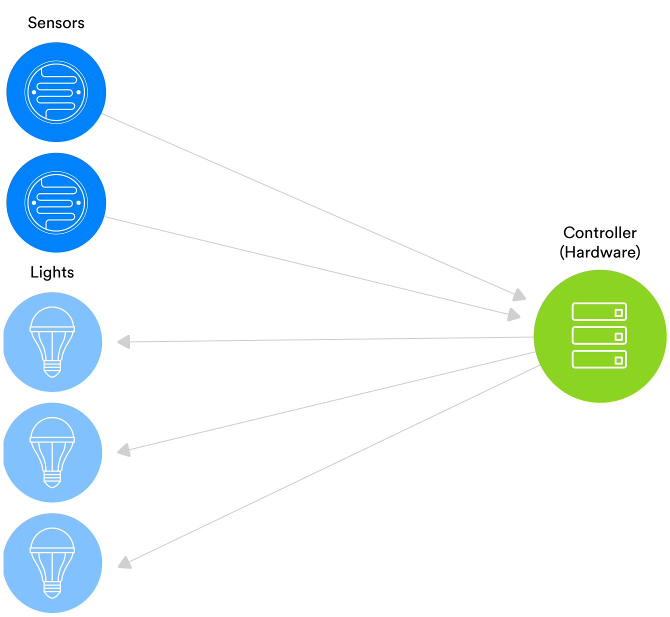

Figure 13 -The Lighting Models Legacy, Centralized Lighting Control

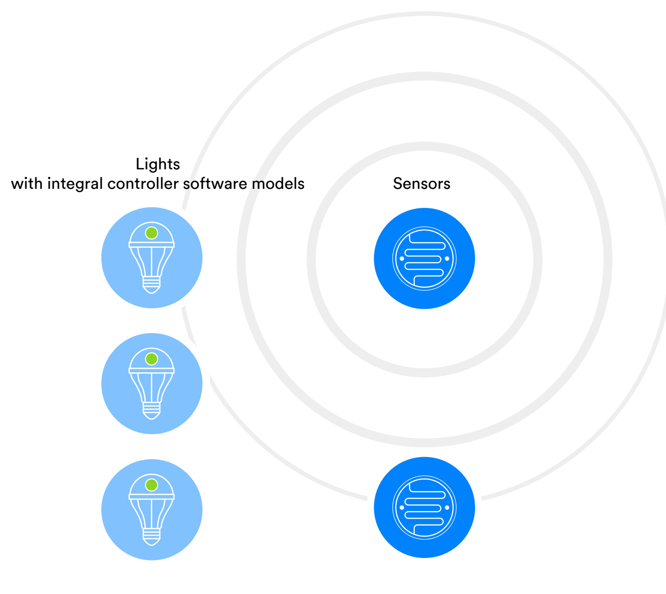

Figure 14 -The Bluetooth Mesh Decentralized Controller Architecture

Bluetooth mesh lighting control is entirely software based and supports a superior, decentralized controller architecture with the controller embedded in the lights rather than in physically separate hardware devices. There are cost advantages and, as described in an article on Bluetooth mesh and scalability, significant performance advantages to this approach. Figure 14 illustrates the Bluetooth mesh decentralized controller architecture.

Of States and States Machines

The terms *controller* and *lighting controller* are used in this paper as an informal shorthand for an element that has the *light LC server* and *light LC setup* server models. The aggregate capability given by these models is known as a lighting controller. The *light LC client* model is used by elements that are able to configure a *light LC setup server* model on a remote device.

The *light LC* server is unusual in that it consumes messages from a model that is not part of the same family, namely the *sensor server* model. This is so that sensors, such as ambient light and occupancy sensors, can provide input to the controller's operation.

The concept of a *finite state machine* is important to understanding the way the Light LC models work to form a lighting controller. Indeed, the Bluetooth Mesh Model Specification approaches the definition of lighting control in a different way to that of the other collections of models. A finite state machine for lighting control is presented and much of the specification refers to that state machine. The state machine defined in the specification is an abstraction that defines how the overall lighting controller works. Sitting underneath this are the mesh models and their mesh states, and it is these mesh states that the finite state machine acts upon and is informed by. The use of the word *state* in these two contexts, that of the overall lighting controller and that of a mesh state data item inside a mesh model, can be a little confusing at first, but it makes perfect sense if you keep the context in mind when reading this section. In this paper, the term *mesh state* refers to a state that is part of a mesh model, and the term *controller state* refers to a state that is part of the lighting controller finite state machine.

The following controller states are defined as part of the lighting controller's finite state machine. Note that this information is a summary of section 6.2.5.1 of the Bluetooth Mesh Model Specification:

State Machine State	Meaning
Off	The lighting controller is disabled and light lightness is not controlled.
Standby	The lighting controller is enabled, but occupancy state changes reported by sensors are ignored.
Fade on	Occupancy has been detected, and the lightness level of lights are in the process of transitioning to the level defined in the light *LC lightness* on mesh state.

State Machine State	Meaning
Run	Lights are now at the lightness level defined by the light *LC lightness on* mesh state, and lightness stays at this level until a timer expires and causes a transition to the Fade controller state to take place. Occupancy events reset the timer. The controller transitions to and stays in the Run controller state when a room is occupied, and it will stay in that state as long as the room continues to be occupied.
Fade	The room is regarded as no longer occupied, so the lightness level starts to transition to the level defined by the *light lightness prolong* mesh state.
Prolong	The Prolong controller state can be thought of as an intermediate state, with a corresponding, intermediate lightness level to which lights fade after occupancy has ceased to be detected. On entering the Prolong controller state, a timer is started. When the timer expires, the controller will start to transition into the next controller state. One example that illustrates the purpose of the Prolong controller state is to avoid abruptly plunging an area of an open-plan office into complete darkness when there are still people working at the other end of the office, which is monitored by different occupancy sensors.
Fade standby auto	After the Prolong controller state's timer expires, the controller switches into the *fade standby auto* state and transitions the lightness level to that defined by the *light lightness standby* mesh state over some transition period.
Fade standby manual	In this state, the controller also transitions the lightness level to the level defined by *light lightness standby*, but switches into this state in response to a manual event, such as receipt of a mesh message like *light LC light onoff set*, which switches the lights off.

Figure 6.7 of the Bluetooth Mesh Model Specification provides a diagrammatic reference to the controller's finite state machine, showing the set of controller states, the valid transitions between states, and the events that trigger them.

Figure 6.4 from the specification shows an example of the controller states being transitioned through and the effect this has on lightness levels at each stage. It is repeated here in Figure 15 for convenience.

Transition Times

Each of the four fade states are transitional in that the system is in the process of transitioning to another state. For example, the Fade On controller state is a state the controller will be in whilst transitioning from the Standby controller state to the Run controller state and corresponding lightness level. How long it takes to transition from the current lightness level to the target lightness level, defined for the next state, can be specified in the *transition time* optional field in relevant mesh

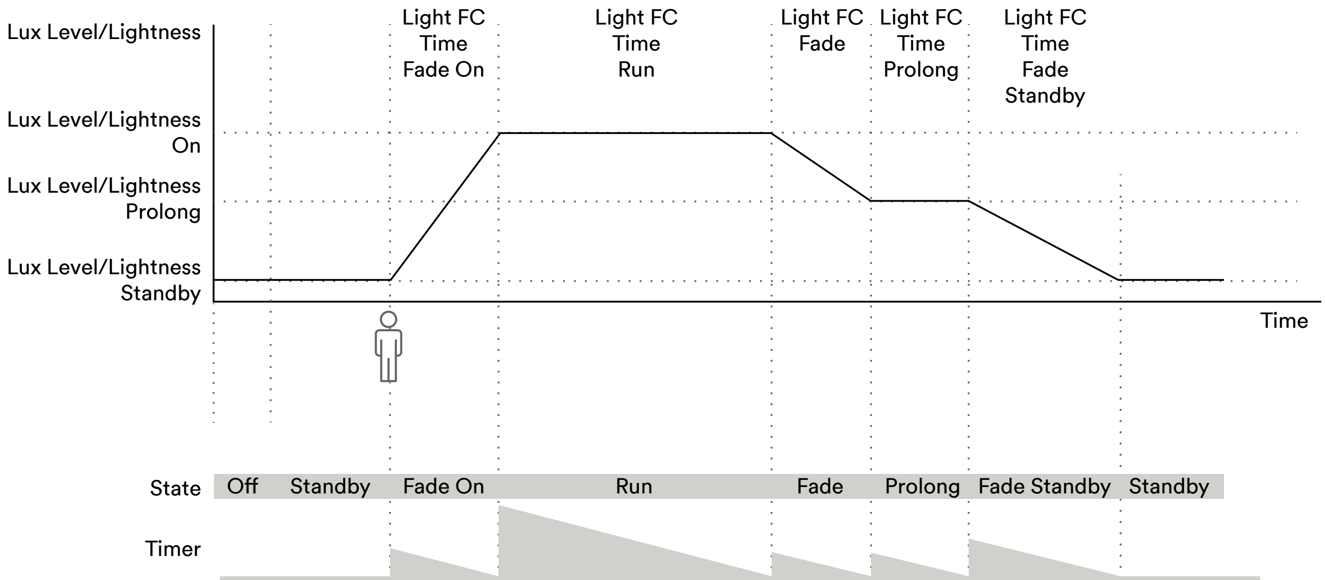

Figure 15 -Lighting controller state transitions

messages, or it can be taken from mesh states, such as *light LC time fade on*.

See section 6.2.5.13.1 of the Bluetooth Mesh Models Specification for details of mesh states that define lighting controller state transition times.

The Details

The LC client, LC server, and LC setup server models form the most sophisticated and, in some ways, complex family of models defined for Bluetooth mesh. They sit at the heart of the support Bluetooth mesh has for advanced commercial lighting systems. This paper has reviewed the concepts governing the operation of these models and how they form a lighting controller, but not looked closely at the underlying mesh states or even the models themselves. There are a significant number of mesh states and properties, some of which allow a lighting controller to be configured to behave in a number of ways. If you are happy with the introduction to the Bluetooth mesh lighting controllers that this section has provided, your next step should be to drill down to the detail provided in the Bluetooth Mesh Model Specification.

5.5 Light CTL Client, Light CTL Server, Light CTL Temperature Server, and Light CTL Setup

At a Glance

These models allow the control of a tunable, white light source. Tunable white lights offer control over the color temperature of a white light and leverage the most recent research into human biological and cognitive responses to light.

About These Models

Figure 16 depicts the relationships the *light CTL server model* has with other models.

Figure 17 shows the simpler *light CTL temperature server* model that extends the *generic level server* model only.

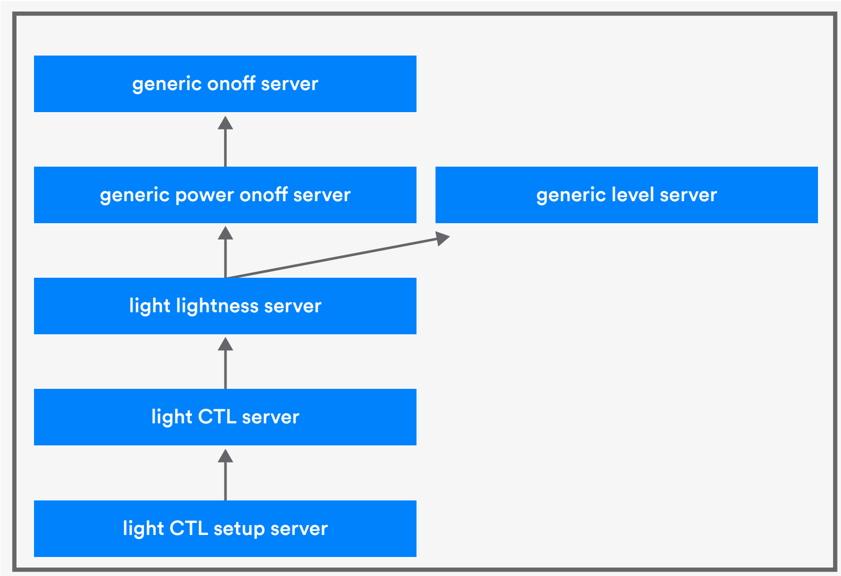

Figure 16 - Light CTL Server and Associated Models

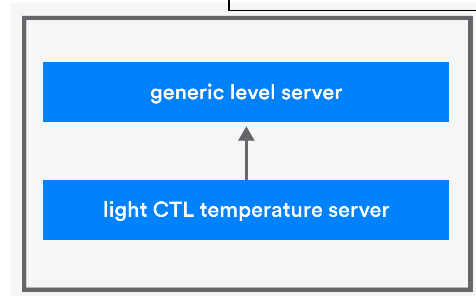

Figure 17 - Light CTL Temperature Server

Central to the *light CTL* models is a composite state called the Light CTL state that belongs to the light CTL server model. It consists of the 6 sub-states shown in the table below.

State	Description
Light CTL temperature	Sets the color temperature level.
Standby Light CTL temperature range	Sets the maximum and minimum color temperature that an element is capable of supporting.
Light CTL temperature default	A default color temperature level for use when powering up in a way determined by the *generic onpowerup* state.
Light CTL delta UV	Some lights allow the color temperature to be varied by some delta away from the usual curve that color temperatures are measured from (known as the *black body locus*). This technique allows certain colors, such as pinks, to accentuated. This state allows a delta value to be set for this purpose.
Light CTL delta UV default	A default *delta UV* value for use when powering up in a way determined by the *generic onpowerup* state.
Light CTL lightness	Controls the lightness of a tunable white light source. Comparable to the *light lightness* state, but for tunable white lights whose color temperature can by definition be varied.

The *light CTL temperature* server model contains only the *light CTL temperature* state plus the *generic level* state due to its extension of the *generic level server* model. It is simpler than the *light CTL server* model, but it may not be obvious why the *light CTL temperature* state appears in both these models.

The answer is that color-tunable light can be changed by manipulating two dimensions: lightness and temperature. It was a requirement that each of these be controllable by making changes to the *generic level* state or, in other words, through a state binding with that state. This implies there must be two distinct instances of the generic level server model to support the two distinct bindings with the *generic level* state, and the only way to accomplish this is to have two elements in the node's composition; the first allows *light CTL lightness* to be modified via the *generic level* state, and the second allows the *light CTL temperature* state to be controlled via *generic level* state changes. The specification designates one element as the *main* element and the other as the *temperature* element. Developers must ensure their node composition reflects this dual-element approach when implementing if independent control via level changes is needed for the two dimensions of CTL.

The *Light CTL client* model provides access to the states in both the *light CTL server* and *light CTL temperature server* models and includes support for the usual set, get, and status message types for each state. Check the section 6.6.2 of the specification for details.

State Bindings

Various state bindings are defined for the states involved in these models, and some of the more interesting ones have been mentioned already. Generic level can be used to control the two dimensions of CTL; power-up events can be used to restore CTL states via bindings with the *generic onpowerup* state. CTL temperature values are restricted by a binding with the *CTL temperature range state*, which is involved in various state binding calculations to ensure values do not fall outside the permitted range.

5.6 The Light HSL Client, Server, and Setup Models

At a Glance

These models provide control over color-changing *lights*, using the hue/saturation/lightness (HSL) model of color representation.

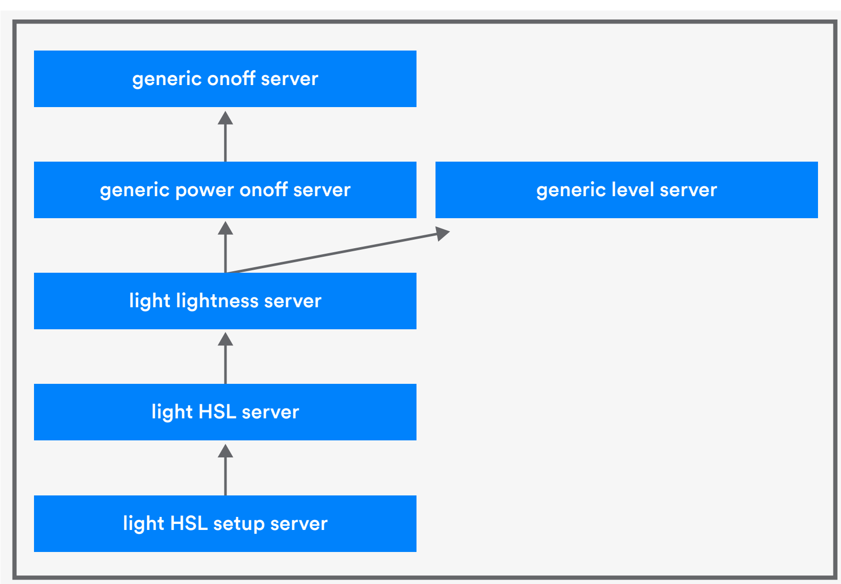

Figure 18 - Light HSL Server and Associated Models

About These Models

Figure 18 depicts the relationships the light HSL server model has with other models.

The most important Bluetooth mesh state involved in these models is the light *HSL state*. It is a composite state consisting of sub-states *light HSL hue*, *light HSL hue default*, *light HSL saturation*, *light HSL saturation default*, and *light HSL lightness*.

Light HSL hue represents the hue as a 0-360-degree angle around a color wheel.

The Perceived lightness of a light (*L*) is approximately the square root of the measured light intensity (*Y*):

$$L = 65535\sqrt{\frac{Y}{65535}}$$

Where *L* is the perceived lightness and *Y* is the measured light intensity (from 0 to 65535).

Figure 19 - Relationship Between Perceived Lightness and Measured Light Intensity

Light HSL saturation represents saturation as a 16-bit value with 0x0000 representing the lowest perceived saturation level and 0xFFFF the highest perceived level.

Light HSL lightness measures lightness on a perceptually uniform scale (see Figure 19).

These states within the *light HSL server* model can be controlled by messages from the corresponding client model in the usual way. Additional messages, *light HSL target get*, and *light HSL target status* allow all three of *light HSL lightness*, *light HSL hue*, and *light HSL saturation* to be queried and reported on by a single message type. If a transition of any of these states is in progress at the time the status message is to be produced, a *remaining time* field is included in the message to indicate how long it will be before the transition to the target state has been completed.

State Bindings

Bindings are defined such that HSL color can be controlled via the generic level of an element and so that the color can be restored to some state when the element is powered up. In addition, there is a relationship between *light lightness actual* and *light HSL lightness,* which makes sense given HSL has lightness as one of its dimensions. In brief:

> *Light HSL hue is bidirectionally bound to generic level, to generic onpowerup, and to light HSL hue range.*
>
> *Light HSL saturation is bidirectionally bound to generic level, generic onpowerup, and light HSL saturation range.*
>
> *Light HSL lightness is bidirectionally bound to light lightness actual.*

5.7 The Light xyL Client, Server, and Setup Models

At a Glance

These models provide control over color changing lights, using the CIE1931 model of color representation.

About These Models

Figure 20 depicts the relationships the *light xyL* server model has with other models, and it is similar to the relationship that the *light HSL server* model has with other models, as shown in Figure 18.

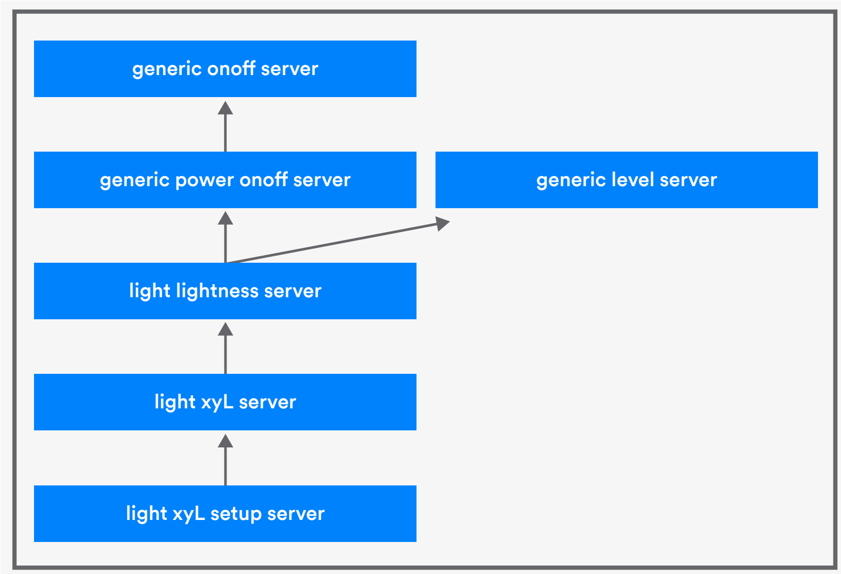

Figure 20 -Light XYL Sserver and Associated Models

Reviewing the *light xyL* state reveals much of what you need to know about these models. It is a composite state that consists of states *light xyL x*, *light xyL x default, light xyL y*, *light xyL y default*, and *light xyL lightness.* These states and the messages provided by the client and server model allow the coordinates of color, according to the CIE1931 color space chart, to be manipulated and defaults to be used when powering up the device to be set.

The *light xyL x* and *light xyL y* states represent coordinates in the range 0 to 1 and are transformed to a 16-bit state value by the formulae:

CIE1931_x = (Light xyL x) / 65535

CIE1931_y = (Light xyL y) / 65535

The special state values 0×0000 and 0xFFFF represent the CIE1931 coordinate values of 0 and 1, respectively.

State Bindings

Light xyL x is bound to *generic onpowerup* and *light xyL x range*. This means the x coordinate can be restored when powering up the device and state binding calculations will keep coordinate values within the valid range for this device. *Light xyL y* has similar bindings.

Lights may implement the server models for both HSL color control and CIE1931 (i.e. *light HSL server* and *light xyL server*). When this is the case, indirect state bindings will exist between the *light xyL* state and the *light HSL* state. This means that lights can be controlled by clients of either type of model.

back to contents

6.0 A Guided Tour of Sensor, Scene, and Time Models

Bluetooth mesh scenes define entire collections of settings for an environment, optimizing it for a particular purpose. For example, you could choose to define a scene that puts a room into the perfect state for a presentation. Switching to a particular scene can be triggered by sensors or a time schedule.

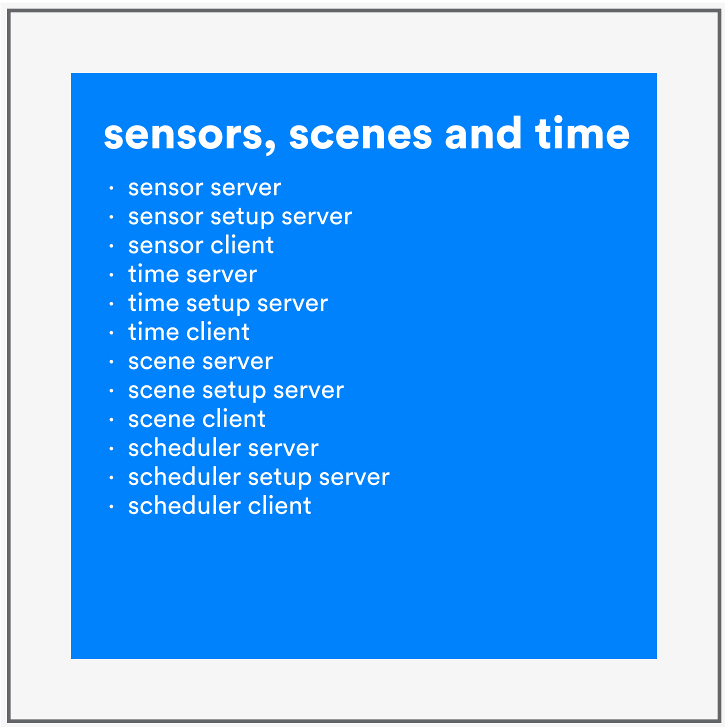

Figure 21 - The Sensor, Scene, and Time-Related Models

Sensors play a critical role in many mesh networking applications, including, but not limited to, that of the smart building. They detect and report events like the changing occupancy status of rooms, and they measure attributes of the environment, sharing this data with other devices.

Sensor data can be used to influence or control the operation of one particular type of device, or it can be used to change the state of many devices of many different types, all in one go.

As an example of the first of these cases, the lightness of lights in a room can be dynamically adjusted in response to changing ambient light levels, as reported by ambient light sensors.

As an example of the second case, consider what we might want to happen when a person walks into a previously empty room. We might want the lights to switch on, the heat to be turned up slightly, and the blinds to open. A Bluetooth mesh network makes this scenario possible through the use of *scenes*. Scenes are collections of memorized model states that are identified by a scene number. Devices can be instructed by a Bluetooth mesh message to switch to the states that belong to a specific scene. This is how mass changes, affecting many different types of devices, can be orchestrated in response to an event like an occupancy change.

Some state changes, including scene switches, can be executed according to a time schedule. A Bluetooth mesh network includes a *scheduler* that is responsible for this behavior. To work though, nodes must have access to a common, accurate system time. Consequently, there are time models, states, and messages, as well as some special roles nodes may play regarding the propagation of time across the network.

6.1 The Sensor Client, Server, and Setup Models

At a Glance

These models provide a generalized approach to sensor operation in a Bluetooth mesh network and allow any type of sensor to communicate sensor readings to other nodes in the network. The sensor setup server allows the sensor and format of its data to be configured.

About These Models

The sensor models make extensive use of properties within a relatively small number of *states*.

Properties differ from states in that they contain both an identifier and a value. The identifier tells us what type of data the property contains so that it is self-describing. States, on the other hand, have no explicit type identifier, and it is the model or message the state is contained within that tells us the data's state.

Leveraging properties has allowed the three sensor models to accommodate any type of sensor and sensor data, rather than requiring different models, messages, and states for each conceivable type of sensor that might be part of a network.

An element that implements the sensor server model must also have the sensor setup server model, which extends it. The sensor client model is not related to other models and can be used standalone. Figure 22 illustrates the relationship between the three sensor models:

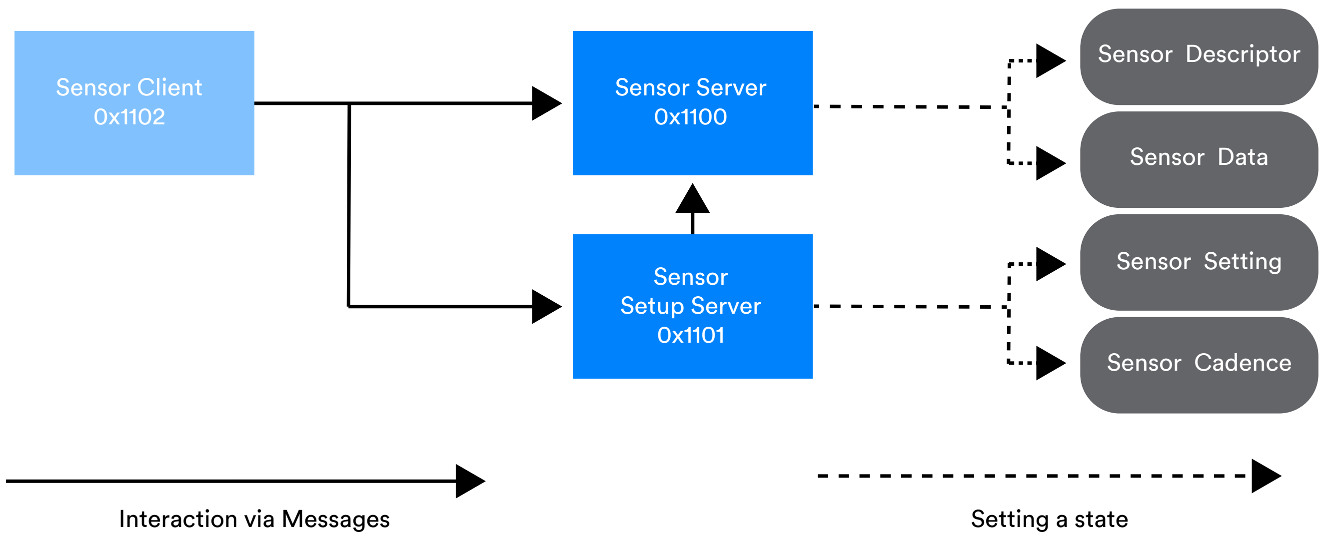

Figure 22 - The Sensor, Scene, and Time-Related Models

Sensor States

The sensor models are defined around a single composite state called the sensor state. This is a fairly complex state whose primary parts are distributed across two models, the *sensor server* model and the *sensor setup server* model, as shown in Figure 22.

The complete breakdown of the sensor state is shown in Figure 23 on the next page.

back to contents

Figure 23 - The Sensor State

The *sensor data* state contains an array of property ID / raw value pairs. The Bluetooth mesh device properties specification defines properties and the characteristics to which they relate. In some cases, a referenced property has a simple value that may be acted upon using sensor model messages, such as *sensor get*, which return the sensor data state value in a *sensor status* message. Some properties define arrays of data, suited to creating histograms, and individual columns from within this tabular data can be accessed with messages like *sensor column get*, which returns a *sensor column status message*.

The *sensor descriptor* state contains information that describes the sensor data available from this sensor. It is not expected to change over the entire lifetime of the sensor.

The tolerance fields provide an indication of the magnitude of possible errors in measurements reported by the sensor. The *sensor sampling* function field indicates the type of function applied to measured sensor values. For example, some sensor data values are instantaneous snap shots of the measured phenomena. Or, perhaps, an averaging function, such as the arithmetic mean, is being applied to measured values and it is this that is contained within the sensor data state. Where a function, such as an averaging function, is being applied, the *sensor measurement period* field indicates the time period over which measurements are being averaged, and the sensor *update interval* indicates the frequency with which each measurement is made by the sensor.

Sensors often have configurable settings, such as sensitivity thresholds. The *sensor setting* state contains a list of such settings and their values. Each member of the list consists of the ID of the property to which the setting applies, the ID of a property that identifies the setting itself, an indication of whether the setting is read only or may also be written to, and the raw setting value itself. For example, occupancy sensors often have motion-sensitivity settings that allow the sensor to be configured so that false alarms, triggered perhaps by small furry animals, are not created. Property 0x0043 Motion Threshold allows the configuration of the required sensitivity level in this case.

The *sensor cadence* state allows the frequency with which a sensor publishes status reports relating to each sensor data type (identified by property ID) to be configured. The rate of publication can be configured to vary according to various conditions. When the value falls within a configured range, the publication rate can be increased. If particularly large increases or decreases in the sensor data value are measured, the reporting rate can also be increased. In each case, the *fast cadence period* divisor indicates by how much the rate of publication should be increased when any of these circumstances arise.

Sensors and Other Models

The *light LC server* model can consume *sensor status* messages. This allows sensors, such as ambient light sensors and occupancy sensors, to be used with important lighting control scenarios. The lighting models are explored in a previous section of this paper.

6.2 Time, Scenes, and Scheduling

At a Glance

Bluetooth mesh *scenes* allow collections of devices of various types to be instructed to load specific settings simultaneously. This allows changes that affect many types of devices to be orchestrated all in one action. Scene selection can be triggered by a Bluetooth mesh message or via a time schedule. In support of scheduled operations, Bluetooth mesh makes it possible for an accurate system time to be propagated to nodes across the network.

Scenes and Scene Registers

A scene is a uniquely numbered list of states with associated state values that is split up and distributed across a number of elements within nodes in the network. Each element that uses scenes has a *scene register*, which is a state contained within the scene server model. The scene register is a table with each row identified by a scene number. Each row in the table also contains an object that acts as a container for all of the states and values that need to be memorized as part of that scene. The specific structural details for this container object are not specified and are left to the implementor.

The aggregate of all rows with the same scene number from all nodes in the network is a unique scene.

Example scene registers for elements within two types of node are shown in Figure 24 and Figure 25.

scene number	state container
10	generic onoff=0,light lightness actual=0,light HSL hue=0x42f4f4
Standby13	generic onoff=1,light lightness actual =65535,light HSL hue=0x42f4f4

Figure 24 - Example Scene Register for an Element of a Light

scene number	state container
10	generic level=100
Standby13	generic level=0

Figure 25 - Example Scene Register for the Blinds

The scene models define messages that the scene client model can publish to store, recall, or delete scenes from within receiving elements' scene registers.

In an example building and network, the requirement might be that when a room is occupied, lights are switched on and set to a given lightness level and hue, and the blinds are opened. An occupancy sensor could publish a scene recall message which specifies that scene 13 be activated when the room becomes occupied, and similarly, publish a scene recall message, activating scene 10 when the room becomes unoccupied. The example *scene register* in Figures 24 and 25 should illustrate how, in the first case, scene 13 would switch the lights on, set their lightness to full brightness, their color to a subtle blue color, and cause the blinds to open (level 0). In the second case, switching everything to scene 10 results in the lights being switched off and the blinds closing (level 100). All of these changes happen simultaneously in response to the scene client model in the sensor publishing a single scene recall message.

A Bluetooth mesh network can have up to 65,535 distinct scenes defined for it. Individual elements can store state values for up to 16 distinct scenes in their scene register, which should be more than enough for any type of device.

Time and Time Propagation

Times in a Bluetooth mesh network are based on the International Atomic Time (TAI) standard. Three models, the *time client*, *time server*, and *time setup server* are defined. When the time server model is present in an element, the time setup server model must also be present, per the usual usage pattern.

The time server model contains a single state called *time*. It contains a TAI time, information about uncertainty, and the degree to which the time can be trusted, plus information about time zone offsets. The *time setup server* adds a further state called *time role*. Time roles define whether or not an element participates in the propagation of time state values across the network and, if so, how. Four distinct roles are defined and represented by the time role state. They are listed in Table 5.2 of the Bluetooth Mesh Model Specification, which is repeated here for convenience:

Value	Role	Description
0x00	None	The element does not participate in propagation of time information.
0x01	Mesh Time Authority	The element publishes Time Status messages but does not process received Time Status messages.
0x02	Mesh Time Relay	The element processes received and published Time Status messages.
0x03	Mesh Time Client	The element does not publish but processes received Time Status messages.
0x04-0xFF	Prohibited	

The *time setup server* model defines messages that allow time role to be maintained in an element.

Time server models respond to messages relating to their time state in the usual way. But to propagate messages across the network, *time servers* with the *mesh time authority* role publish the time periodically in accordance with the publish period state, which is part of the *configuration server* model. Elements with the role mesh *time relay* also publish unsolicited time status messages, but they do so only when one is received from a *mesh time authority*. *Mesh time clients* are end points in the time propagation process, receiving and storing time data from *time status* messages, but not publishing or relaying them. It is in this way that time is distributed across the network.

Scheduling

Certain types of state changes can be scheduled to take place at a specific time every day, on a specific day of the week. This is made possible by a set of models, the *scheduler server*, *scheduler setup server*, and the *scheduler client*. The *scheduler server* extends the *scene server* model, and the *scheduler client* model extends the *scene server* model so the ability to schedule actions that change state is dependent on the scene models being implemented.

The *scheduler server* model contains a tabular state called the *scheduler register*. This state allows up to 16 sets of scheduling data to be stored, each consisting of scheduling time and frequency information, an action to take, and a scene number (optional). It therefore allows up to 16 state changing actions to be scheduled. Actions allowed switch the element on, switch it off, or recall the specified scene.

The scheduler offers a great deal of flexibility in how actions are scheduled, and every action can have an associated transition time specified for it as well.

back to contents

7.0 Summary

This ends the technical overview of the Bluetooth mesh models. You should have a good understanding of how the *generics* support the use of fundamental capabilities that many device types possess, how commercial lighting requirements are met by the lighting models, and how sensors can be used to inform other devices of environmental data, perhaps initiating scene changes as a result. Finally, you should know how time plays a role in a Bluetooth mesh network, with a scheduler available to trigger state changes on a scheduled basis.

The Bluetooth mesh models are the building blocks for interoperable mesh products and the means by which diverse requirements can be met in smart buildings and elsewhere.

7.1 Additional Resources

The following resources are recommended to help you learn more about Bluetooth mesh:

Mesh Technology Overview	A short technical introduction to Bluetooth mesh technology.
Mesh Glossary of Terms	A glossary of Bluetooth mesh terminology.
Mesh Developer Study Guide	Self-study material with hands-on programming exercises for developers.