**Zigbee应用框架开发向导**

Zigbee应用程序框架是**一组嵌入的C代码**，通过AppBuilder可以**配置这些代码**来实现任何Zigbee cluster(ZCL)应用程序。从EmberZNet SDK的2.6.0版本开始，Zigbee应用程序框架取代了应用程序框架V2。本向导介绍了Zigbee应用程序框架的**结构和用法**。

关键点：

* 提供应用框架的所有组件的参考，包括回调、插件、API和CLI
* 讨论了扩展ZCL（zigbee ckuster library）
* 提供基于appbuilder设计应用程序指南

**一 介绍**

* 1. 目的

Zigbee应用框架（或者叫做ZCL应用框架）是一种嵌入式C代码，可以被AppBuilder配置用来实现任何ZCL应用；应用框架的路径在app/framework。

该向导覆盖应用框架的结构和使用方法。

* 1. 创建一个应用

使用应用框架创建一个应用，需要以下几步：

1. 使用SSV4的AppBuilder创建一个应用框架的配置文件，平台（芯片）配置文件和工程文件都是由AppBuilder创建的；AppBuilder的使用和如何关联到应用框架，详见UG103.2 zigbee原理；关于如何使用AppBuilder详见其帮助信息SSV4/Help/Help Concents/SS AppBuilder.
2. 在callback回调函数中写你自己的代码；使用应用框架的API操作属性、发送、接收和响应zigbee标准命令。关于应用框架API详见第五章应用框架API。
3. 在IDE中打开生成的工程，编译并且下载到开发套件中。
4. 运行应用程序，并在IDE Console窗口中使用CLI命令行与硬件交互；详细信息请参考在线帮助Help/Help concents.

1.3移植应用程序

从AFV2移植一个应用程序到当前的应用框架，参考知识库：使用AppBuilder迁移工程，从AFV2到当前zigbee应用框架。

应用框架描述：参考UG103.2-1.3硬件和软件元素

一个zigbee方案需要RF射频和处理器（SOC或者单独两部分），并且需要基于协议栈实现应用层操作；EmberZnet Pro是silicon labs 实现的zigbee pro协议栈。

通常开发者可以购买一个RF射频组件和一组软件包进行开发，尽管有第三方提供的协议栈；通常软件和硬件供应商会提供硬件参考设计和软件的参考代码；基于这些组件，硬件开发者可以定制自己的硬件，另外，一般供应商会提供模块作为低成本的方案选择。

由于zigbee的开发的特性，软件开发通常是和硬件相关的，以提供最佳的性能；Silicon Labs提供了一个基于Zigbee规范的标准网络API和一个应用程序框架，该框架为客户提供了一种基于Zigbee profile和Zigbee clusters(ZCL)快速开发其应用程序的方法，如下所述。

另外，一些第三方软件开发公司专门开发Zigbee应用程序，并可以帮助开发新产品。

1.3.1关于应用框架

任何软件都可以从头开始创建，但是这是一个漫长和复杂的过程；另一种方法是在已经可以工作的代码基础上修改，以满足自己应用程序的需求；修改已经可以工作的代码是一种快速和有效的方法，特别是刚接触这项技术的时候；为了实现这个目的，silicon labs不仅提供了示例代码，而且提供了几个应用框架；**应用框架包含大量silicon labs提供的代码，以库文件和插件的形式提供；**silicon labs同时提供SSV4开发环境，这个环境包含IDE和AppBuilder

;AppBuilder是一款交互式的图形界面工具，**允许配置应用框架中包含的库和插件**，以实现由zigbee应用层（如zigbee 3.0）定义的设备，如智能电表等。

Zigbee应用框架主要基于ZCL，用于开发zigbee设备；它提供了一组嵌入式代码来实现**ZCL处理、网络任务**（创建、入网等）、典型的zigbee应用处理（commisioning/steering状态机等、zigbee3.0应用层的支持-加密和APS）,同时包含兼容老版本的profile 如HA,SE,ZLL等。

这些代码根据silicon labs的最佳推荐和不同的zigbee应用profile指定，可以作为“软件参考设计”运行在SOC模式或者通过EZSP协议运行在NCP模式；

应用框架提供基本的服务，当与特定的代码组合使用，可以提供大部分无线应用；你的工作将变为开发特定的硬件，并将之与silicon labs的软件相结合的任务了；每个silicon labs的协议栈，都提供了基于其应用框架的示例代码。

使用应用框架可以带来以下益处：

1. 有助于维护一个通用的方法，并且遵循典型协议
2. 有助于确保遵从zigbee协议规范，因为silicon labs应用框架是遵从与认证的规范
3. 方便移植，因为silicon labs维护内核组件
4. 提供在应用程序中包含**复杂功能**的能力，而不需要做**代码级**管理
5. 提供可配置、遵从性、和易升级协议栈的能力
6. 提供一组**回调/命令**接口和设置点，这个设置点原理协议栈，它会更靠近应用层行为，而不是协议层行为。
7. 提供一组**插件**更容易实现高层应用逻辑，例如debug功能和CLI功能，ZCL luster的实现、应用状态机、或者复杂功能。

以上可以帮助我们快速实现产品上市和提高产品的可靠性。

1.3.2 其他工具

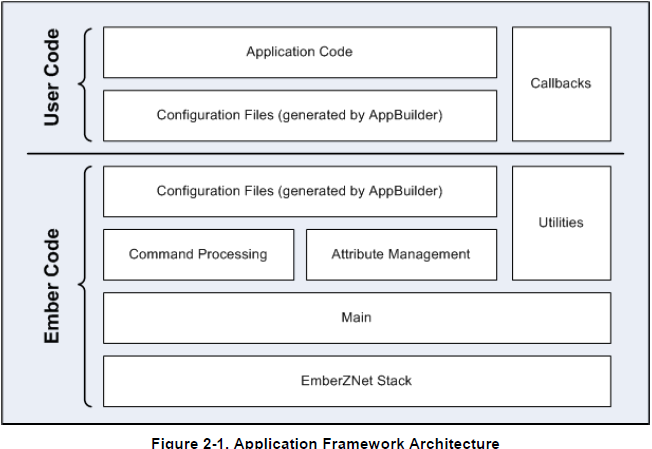
除此以外silicon labs 提供以下工具帮助开发者设计产品：

* OTA BootLoader允许产品上市后更新系统软件
* 网关接口允许和其他系统对接
* 处理器编程工具

**二 应用框架架构**

Zigbee应用框架位于协议栈之上，使用协议栈**”Handler”**接口，并向开发者展示高度抽象的、面向应用的接口。

Zigbee应用框架最主要的一个特性是区分用户和silicon labs创建的代码；silicon labs提供所有的应用框架代码，用户的代码应该位于框架之外，并且应该通过应用框架工具的API和回调函数与框架交互；以下框图显示了应用框架的预览和两种代码是如何分开的。



**Simple main**插件在app/framework/plugin(C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.7\protocol\zigbee\app\framework\plugin)使用zigbee协议栈”handler”接口，**并将zigbee应用框架绑定过到zigbee协议栈；**另外两个main文件位于app/framework/util

(C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.7\protocol\zigbee\app\framework\util **–utilities实用工具**),一个为soc应用，另一个为NCP应用。

**Af-main-soc和af-main-host文件实现了emberIncomingMessageHandler()，**将收到的所有消息传给应用框架用于命令处理；一旦消息被处理，它们要么传给对应的zcl handler处理，要么直接传给cluster特定的回调函数（appbuilder生成的）；大部分的命令处理代码通过app/zcl

（C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.7\app\zcl）的ZCL XML（定义了profile）文件生成的。

所有从ZCL XML文档产生的代码和头文件，被放在<user workspacedirectory>/<applicatiOn

name>/ 中，同时应用程序相关头文件和回调函数一起放在这里。

**三 应用框架路径结构**

当生成一个工程，这个工程默认放在workspace路径下，文件夹的名字就是工程名，如我创建一个工程为C:\Users\kid.li\SimplicityStudio\DMP\_workspace\konkesensor；**本章中出现的文件路径，可以在simplicity studio zigbee protocol SDK路径中找\SiliconLabs\SimplicityStudio\**

**<version>\developer\sdks\gecko\_sdk\_suite\<version>\protocol\zigbee）。**

**tool/appbuilder:**

Appbuilder使用的配置和模板文件，其位于SDK中，路径如下所示：（C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko\_sdk\_suite\v2.7\protocol\zigbee\tool\appbuilder），主要包含token&zcl&linke&pathlist&macros&library&ezsp&cli&gatt&platform&

layout等properties权证文件，**cli&callback&stackhandler的链接xml文件**，二进制镜像&典型回调信息info文件，host-unix makefile,板子模板的头文件等。

当Appbuilder安装于指定的协议栈时，AppBuilder会检查这个路径，以便最新的协议栈发布时，它能够找到最新的ZCL cluster实现的XML相关描述；**我的理解是新版本的zigbee协议栈会堆一些cluster进行修改，AppBuilder需要通过app\zcl的ZCL XML的描述来区别，于是在生成代码的时候会重写这一组配置和模板。**

你可以添加你自己的用户cluster .xml到工程中，通过Appbuilder的”zigbee stack”选项卡；如目录下的sample.xml，可以添加用户自己的cluster-属性和命令。详见Simplicity Studio AppBuilder Help at Help | Help Contents | Simplicity Studio AppBuilder | Creating custom clusters.

**总结：该目录与ZCL模板相关，包含了handler、callback、命令行处理模板。**

**app/framework:**

所有的应用框架代码包含于本路径中；他们各自的功能代码在各自的子目录中。包含了CLI命令行、插件、场景、安全、一组实用工具，具体请查看相关目录。

**app/framework/cli:**

实现了应用框架的命令行接口，

核心的命令行在app/util/serial/command-interpreter2.c.文件，这段代码需要看一下，它用来解析收到的串口命令，并返回操作结果；

1. 所有的命令都要正确的参数，如果参数不正确或者不全；函数返回用户正确的参数设置方法。
2. 传递给命令行的参数必须是以下格式：

<int>: 123(decimal) or 0x1ABC(hex)

<string>: "foo"(string) or {0A 1B 2C}(array of bytes)

**app/framework/include:**

应用程序框架使用的扩展API。

这个路径反映协议栈使用的include路径；它是所有外部接口的点位置。

**总结：这个目录包含了框架使用的一堆头文件，以及协议栈使用的数据结构定义，分别在af.h和aftype.h.**

**app/framework/plugin:**

ZCL处理代码，这个目录包含了silicon labs team所写的所有处理ZCL命令的代码；这些代码可以通过AppBuilder的plugin页签随意的添加到用户的代码中去；如果你没有选用这些plugin你可以自己通过callback实现所有的命令处理代码。

**app/framework/scenarios:**

Sample Code，不多解释。

**app/framework/security:**

zigbee安全相关的代码。

与密钥建立相关的代码在app/framework/cluster中，建议看一下。

**app/framework/util:**

应用框架的主函数、消息处理、其他实用（有用）代码等

这个目录包含了zigbee应用框架的核心；属性处理-多个ep；同时包含API用于访问、读、写属性；API位于文件attributetable.h, and attribute-storage.h.

**四 生成应用程序配置文件**

Zigbee应用框架用相同的预处理器指令配置代码，指示哪些代码包含或者不包含在应用框架内。处理**应用头文件**外，AppBuilder还生成了一个“**EP端点配置**”头文件，这个头文件以endpoint\_configuration.h作为后缀。如konkesensor\_endpoint\_config.h。

**以下文件位于workspace空间中，如我创建一个工程为C:\Users\kid.li\SimplicityStudio\**

**DMP\_workspace\konkesensor；**

**<DeviceName>\_endpoint\_configuration.h**

**这个文件配置应用框架的静态数据结构；**它允许属性元数据被多个EP共享，其中，每个EP有自己的存储空间；在<DeviceName>\_endpoint\_config.h中，使用#define定义的**属性相关**的**宏定义数据，**在文件app/framework/util/attribute-storage.c中使用；

每次修改AppBuilder设置后，这个文件都会被重新生成；silicon labs建议你不要修改这个文件或者手工定义，因为文件中的每个宏定义都有复杂的关系。

这个文件（端点配置文件）在工程中的重要角色或者用途，在9.1节ZCL属性配置中有更详细的描述。

**<DeviceName>.h**

如 konkesensor.h，是应用程序的主（main）头文件，它通过#define宏定义，**打开你需要的应用框架的特性；**例如你在应用框架中选择的网络类型（Zigbee Pro）、设备类型（COOR？ROUTER?ED？）、安全方式、ZCL段、选择的属性、endpoint、cli、网络段、callback段、全局开关、区域开关（打印）、Plugin宏开关、PLUGIN使用的API头文件以及用户自定义的宏。

**<DeviceName>\_callbacks.c**

应用框架产生的桩文件，它包含了你工程中选择的需要实现的回调函数；着就是你的代码要执行的部分；当然你可以不局限于使用该文件实现你的代码，你可以添加其他文件到工程中，实现同样的回调功能。

**<DeviceName>.hwconf**

生成的与你的芯片外设相关的配置文件；SSV4提供有好的界面去修改硬件配置；详见AN1115用simplicity studio配置32位器件外设，查看详细信息。

**<DeviceName>\_tokens.h**

如果你要将属性放在token中（NVM），那么AppBuilder将生成这个文件来配置token的存储。定义了属性刀token和从token都出属性的宏。

**<DeviceName>.ewp, eww, .xip, .xiw, .mak**

为您的应用程序生成项目文件。AppBuilder只生成与您选择的平台匹配的项目文件。这些文件可能会被加载到您的IDE中并进行编辑，以构建项目的其余部分。

**4.1应用框架文件**

除了以上设备特定的文件之外，AppBuilder产生的其他应用框架文件页保存在<User Workspace>/<Device-

Name>路径中。

产生的文件数量是变化的，主要基于你选择了支持哪些plugin和你对这些插件的要求；zigbee应用框架产生以下文件，但不止限于以下文件：

**af-structs.h:**应用框架定义的数据结构，**用于解析空中数据包-命令、请求等。**

**att-storage.h:** 定义应用框架的属性存储机制-属性读、写掩码/cluster修改功能掩码（初始化、属性改变、默认响应、消息发送、服务或客户等）/命令修改掩码（支持客户/服务发送命令？支持客户/服务接收命令等）

**attribute-id.h:** 定义了所有zcl定义的属性ID。

**attribute-size.h:**ZCL属性类型对应的字节数量。

**attribute-type.h:** ZCL定义的控制数据类型。

**call-command-handler.c: 处理所有空中接口过来的非通用命令，它安排空中来的cluster命令到回调函数；也处理没有回调函数的命令。**

**call-command-handler.h:**上面文件的头文件，声明其中的函数。

**callback-stub.c:**提供用户定义的回调函数的桩函数；只有在油壶应用程序没有定义的时候，这里的函数才有机会被编译。

**callback.h:提供所有应用框架和用户程序可能被实现的回调函数声明；它定义了完整的回调接口；是zigbee框架和用户程序通讯的主接口。**

**znet-cli.c:** 这个文件提供所有生成的命令行接口。

**client-command-macro.h:**作为zigbee应用框架的一部分，提供宏定义用于填充发往空中的数据包；用户应用程序配置的clusters支持的每一个命令都在这些宏定义中，它们将通过调用应用框架来填充发往空中的数据包。例如，填充on/off命令包宏定义如下：

#define emberAfFillCommandOnOffClusterOn()

**cluster-id.h:**定义从ZCL库载入应用框架的clusters。-看起来像是所有的cluster ids.

**command-id.h**:定义了从ZCL苦衷载入到应用框架的所有command ids.

**debug-printing-test.h:** 定义在应用框架中打开打印

**debug-printing.h:**

**enums.h:**zcl定义的相关枚举变量结构

**print-cluster.h:**基于zigbee应用框架定义用于打开打印的cluster。

**stack-handler-stub.c:**协议栈处理的桩函数，它可以在应用框架中重写。

针对协议栈处理，目前是空的；它同时包含了#include "app/framework/include/af.h"

程序中解释是，这个C文件提供所有callbacks的桩函数，如果一些情况下，用户没有实现的callback,这些桩函数将被调用；

**五 应用框架API**

**Zigbee应用框架API在app/framework/include/af.h中提供。**这个文件接口的提供方法与协议栈提供的emberZnet pro 提供的API方法是一致的；应用框架API参考手册和安装文件一起提供的，也可以在线查看<https://docs.silabs.com/>。

**许多应用框架（af.h中包含的在plugin,util等目录中）中的函数提供一个字节的endpiontid参数；**如下**（af.h中声明）**，第一个参数，尤其对**cluster初始化、cluster 周期tick调用、属性管理；**

EmberStatus emberAfDeactivateClusterTick(**uint8\_t endpoint**,

EmberAfClusterId clusterId,

bool isClient);

EmberAfStatus emberAfWriteServerAttribute(**uint8\_t endpoint,**

EmberAfClusterId cluster,

EmberAfAttributeId attributeID,

uint8\_t\* dataPtr,

EmberAfAttributeType dataType);

例如：函数emberAfReadAttribute位于app/**framework/util/**attribute-table.c中，使用endpointid作为第一个参数。

Zigbee应用框架包含的以下示例如下：

boolean **emberAf**ContainsCluster(int8u **endpoint,** EmberAfClusterId clusterId);

boolean emberAfContainsServer(int8u endpoint, EmberAfClusterId clusterId);

boolean emberAfContainsClient(int8u endpoint, EmberAfClusterId clusterId);

所有应用框架的API都是以“emberAf”作为前缀，以区分用户函数声明。

个人理解，应用框架实现了zigbee联盟定义的zigbee clusters library（ZCL），而CLUSTERS分为客户和服务端，所以，AF.H的API主要是操作clusters相关的函数。

获取endpiont的属性信息在app/framework/util/attribute-storage.h中；但入口仍然是在af.h中；如检测一个endpoint是否支持确定的属性，调用函数**emberAfContains**Attribute(int8u endpoint, ClusterId, AttributeId attributeId)，函数返回值标识是否存在支持。

**注意：**读、写属性需要一个endpint，如果没有给出，编译器会返回隐式声明的警告；所以，请注意警告。

**六 应用框架回调接口**

**Zigbee应用框架回调机制是有意设计的**，**为的是将所有用户程序从应用框架中移除；**如果试图将应用程序放进应用框架，silicon labs 认为这是一个bug，因为这意味着缺少一个回调来满足你的应用程序的需求；这种情况下，请发ticket通知silicon labs。

通常，当一个回调函数被调用时，Zigbee应用框架首先会向应用程序代码发出一些传入消息或请求一些应用程序数据；在回调函数API中，一些回调函数返回一个布尔值（TURE），标识消息已经处理了，应用框架不要再处理了；例如，**一个特定的消息，如果你的处理和应用框架的处理相冲突，就需要返回TURE代表消息已经处理完成了；这样就保证了应用框架不会干预你对消息的处理**。

**6.1 回调函数文件的生成**

AppBuilder已经产生了一个桩回调文件；默认的，如果AppBuilder发现默认路径已经存在了这个文件，它将不再生成装文件；你必须指定是否覆盖已经存在的文件（**貌似默认是覆盖**的，需要勾选callback的复选框才会提示）。

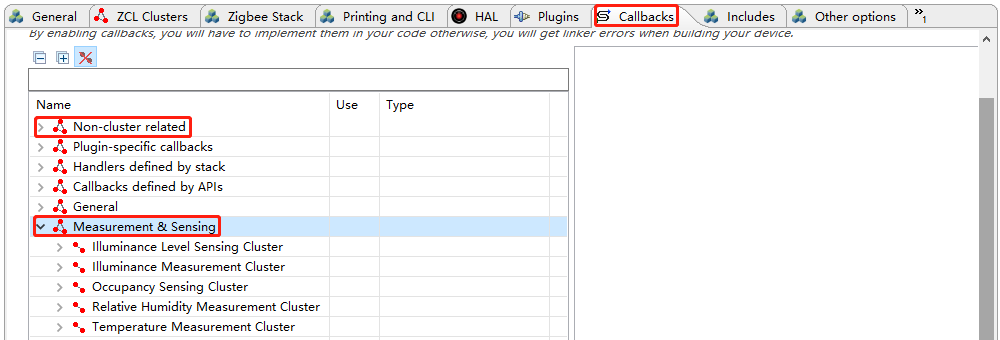
修改并再次生成工程时，会提示是否覆盖回调函数的文件；如果选择覆盖，AppBuilder会备份之前的文件。

**注意：**你可以在任何地方实现你的回调函数；它们不需要在生成的回掉函数文件中实现；然而，如果你在别的路径实现该函数，请将对应的函数从生成的文件中删除，这样Linker才不会检测到重复定义了函数（回调函数）。

**6.2 非clusters相关的回调**

为了方便使用，AppBuilder GUI将回调分成了及部分处理；其中，第一部分时非cluster相关的回调；由文件tool/appbuilder/callbacks.xml描述的回调函数构成；这些回调已经被手动插入到Zigbee应用程序框架代码中，插入的位置是用户希望接收关于Zigbee应用程序框架功能的信息的位置。例如：**按键中断回调函数被插入到中断处理上下文中。**

如下，AppBuilder的callback界面：



所有的全局命令归于此类，例如网络管理、安全等；zigbee应用框架包含处理全局命令的代码；如果任何全局命令回调函数返回TURE，代表着应用程序来处理命令，不需要应用框架处理；如果FALSE，则应用框架继续处理这条命令。

例如：

**“命令接收预处理”回调函数**（emberAfPreCommandReceivedCallback

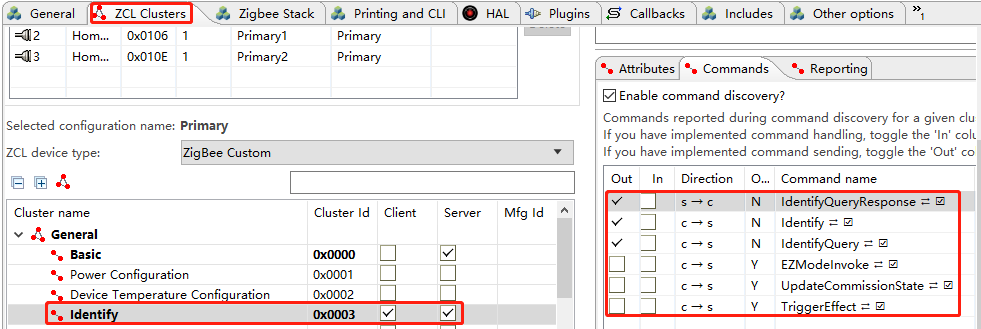
(EmberAfClusterCommand\* cmd,boolean isInterpan)），在接收到ZCL命令后，而该命令还没有被应用框架的命令处理代码处理前，这个回调函数被调用；这ZCL条命令被解析Ember

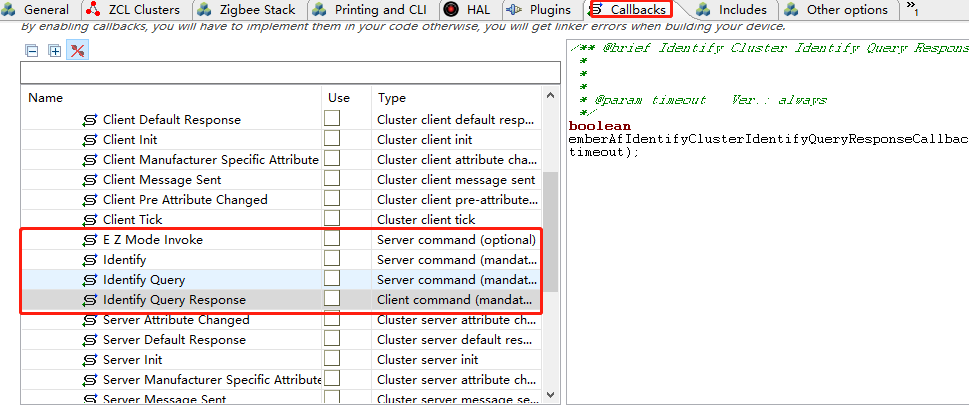
AfClusterCommand结构体，它提供一种简单的方法访问命令包含的相关数据，包括EmberAPSFrame、消息类型、源、buffer、长度、以及其他命令相关的标识；这个回调函数页返回一个布尔型变量代表命令是否已被处理；如果返回TRUE，代表应用程序已经处理好了，不需要进一步处理。

**6.3 cluster特定的命令处理回调**

Cluster相关的回调函数由应用框架产生；它可以接收预解析的空中命令包；通常ZCL命令和cluster特定的回调函数时一一对应的关系。**说明，请注意是命令->callback映射。**

例如：







Cluster特定命令回调函数全部返回布尔值；这个值允许旁路应用框架的处理；如果你实现了一个特定的cluster命令回调函数，并且返回TRUE给应用框架，则应用框架认为命令在框架外处理，并且要求的一些相应已经发送了；如果返回FALSE,应用框架认为不识别该命令并返回‘unsupported cluster command’.

**6.3.1命令回调上下文**

所有命令相关的回调函数在emberIncomingMessageHandler函数上下文中被调用；这意味着该上下文中对应用程序可用的zigbee API,在处理回调命令中也可用；这些API列表在协议栈API文件stack/include/message.h.中；在命令回调函数中有效的协议栈API也列在消息头文件stack/include/message.h中，包含如下：

emberGetLastHopLqi()

emberGetLastHopRssi()

emberGetSender()

emberGetSenderEui64()

emberGetBindingIndex()

emberSendReply() (for incoming APS retried unicasts only)

emberSetReplyBinding()

emberNoteSendersBinding()

另外还有：

**单播、广播、多播APIs、源路由和MTO路由请求APIs、重发、地址表、多播表、输入输出过滤等APIs**。

**6.3.2命令回调函数的数组操作**

任何包含参数数组的Zigbee消息都作为指向数组开头的int8u\*指针传递。即使框架知道数组中的参数可能是另一种类型(比如int16u或int32u)，也会这样做，因为在框架可能运行的各种处理器上存在字节对齐问题。实现回调的开发人员必须解析数组并为其硬件适当地转换其元素。

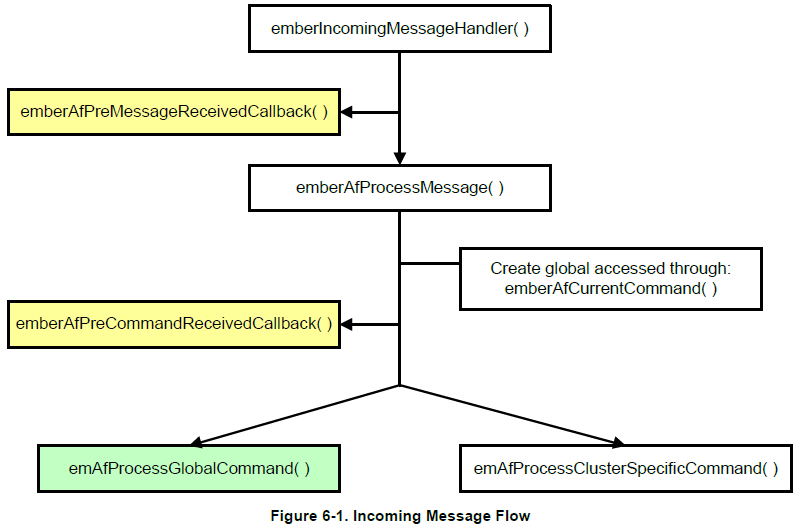
**6.3.3全局命令回调函数**

Zigbee全局命令也包含在应用框架回调接口中；这些回调函数用于接收和响应全局命令；例如，如果你接收到别的设备发过来的全局读属性命令，可以通过实现回调函数emberAf

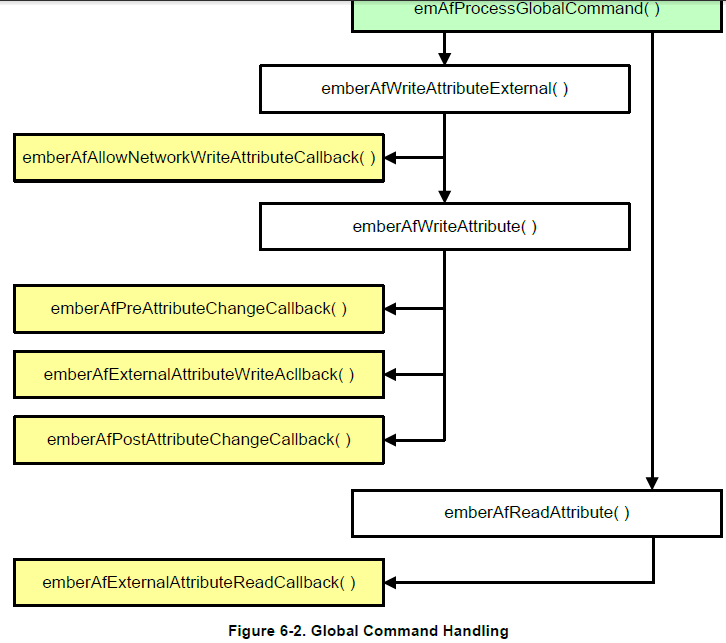
ReadAttributesResponseCallback处理和响应命令。

**6.4 调用流程**

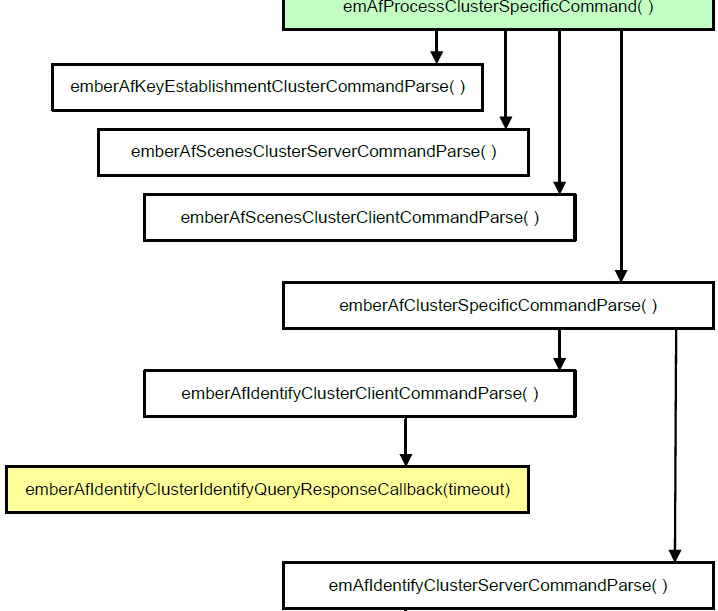
下图显示了emberIncomingMessageHandler如何实现从接收到消息->应用框架代码处理->应用程序的回调函数的流程。

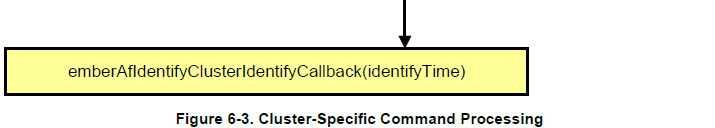


如果输入的消息检测出来是全局命令，那么它将传递给全局命令处理函数，流程如下：

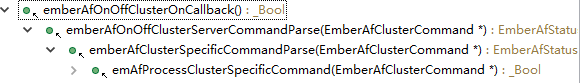


否则，如果发现是lcuster特定的命令，传给cluster特定的函数处理，如下：





如下，是on/off cluster对应的on/off命令调用过程，说明，以上流程是举例，具体参考代码。



**6.5 回调引用**

无

**七 时间处理**

Zigbee应用框架提供一个API访问当前的系统时间(int32u emberAfGetCurrentTime()，app/framework/util/time-util.h.中有详细描述；这一章描述该函数是如何实现的，参考framework/util/time-util.c.:

如果zigbee time cluster sever在系统中已经实现，那么这个函数通过调用 int32u emberAfTimeClusterServerGetCurrentTime(),从服务端获得当前时间，这种情况是通过读取服务端（远端设备）的time cluster sever的time属性并返回给客户端(本地）；如果系统中time cluster sever没有实现，那么emberAfGetCurrentTime调用emberAfGetCurrentTime

-Callback获得当前时间。

如果你的设备想获得当前时间，但是系统没有实现time cluster sever插件，那么可靠的做法是在系统中找一个设备来维护它自己的时间，当它收到时间请求的时候，通过emberAfGet

CurrentTimeCallback返回时间；对于SE设备，这个功能是非常重要的，例如一个IPD设备（in-premise display），本质上在时间管理上，它是独立的；它需要为别的设备提供time cluster sever输出；因此，IPD设备管理一个自己的时间，并且当框架请求的时候，通过函数emberAfGetCurrentTimeCallback提供给应用框架，

如果应用程序提供了time cluster sever,则服务代码通过emberAfGetCurrentTimeCallback初始化和更新time属性，如果emberAfGetCurrentTimeCallback返回0，则time cluster sever每秒将属性增加1；因此，如果在系统中，实际的时间值可以同步或者写到服务的属性，那么你的其他设备不需要再实现emberAfGetCurrentTimeCallback函数了；更多的通过绑定time cluster sever实现时间处理的方法，请参考app/framework/plugin/time-server/Time

-server.c。

**八 事件**

在soc和Host的应用中，zigbee应用框架和cluster代码使用zigbee协议栈的机制去调度事件；

使用zigbee协议栈事件机制保存Code（OTA升级？）和RAM，可以使睡眠设备能够更好的工作。

从高层面来看，事件机制为需要周期激活或者失活处理的行为提供一个中心位置，（激活或失活）触发源来自用户输入、空中命令、或者设备初始化；事件机制优于静态的滴答机制，因为它允许应用框架准确的知道设备何时（具体时间执行什么）执行的行为是什么（比如100毫秒后更新level值、250毫秒后扫描下一通道等）；这对于睡眠设备尤其重要，因为它们需要知道准确的唤醒时间，更重要的是事件处理时，它们不能睡眠；zigbee应用框架有两种类型的事件：用户事件和cluster事件；用户事件是应用框架的使用者创建的，它可以在应用程序中执行任何目的的应用；cluster事件和cluster的实现相关，它包含在应用框架的plugin中。

8.1 创建用户事件

Zigbee应用框架使用zigbee典型的事件机制去控制和运行“用户”应用事件在框架内；协议栈事件机制文档在vent.h 头文件中，位置在stack/include/event.h.

**这个文档写的比较详细，必要看一下……**

用于框架和AppBuilder提供有用的接口创建和添加用户事件到你的应用程序中；在AppBuild

Er中创建事件是在AppBuilder文件的“include”标签页的“Event Configuration”点击“ADD NEW”,这个操作添加一个事件到事件表中，它们将会被应用框架运行；同时AppBuilder也为用户事件在“callback”文件中产生一个“桩”。

**8.1.1 事件函数和事件控制**

事件包含两部分：时间函数，在事件出发后执行的函数；事件控制EmberEventControl结构，用于调度事件；事件控制结构体如下：

typedef struct {

/\*\* The event's status, either inactive or the units for timeToExecute. \*/

EmberEventUnits status;

/\*\* The ID of the task this event belongs to. \*/

EmberTaskId taskid;

/\*\* Indicates how long before the event fires.

\* Units are milliseconds.

\*/

uint32\_t timeToExecute;

} EmberEventControl;

应用框架的事件机制需要准确跟踪下一个事件什么时候发生，以便确定睡眠与否；另外还需要知道事件发生的时候执行的函数。stack/include/event.h.头文件中，提供了关于创建事件的更多的文档。

**可以看一下…**…

**8.1.2 用户事件示例**

Z3Light 参考应用使用一个用户事件管理它的状态；这个事件包含两部分：一个叫做commissioningLedEventControl的EmberEventControl结构体，和一个叫做commissioning

LedEventHandler的事件处理函数，事件出来函数在每次事件触发后执行；事件控制和事件处理函数都在Z3Light\_callback.c文件中；关于这个应用的文档，可以在AppBuilder的“通用”页前中找到。

**8.2 cluster事件如何创建**

每一个cluster包含一个服务和一个客户端的“滴答”回调（**说明，虽然都有，但是可以选择是否使用；如果使用的话就会产生一个对应的事件**）；AppBuilder产生一个事件表（如果选择使用tick回调），对应每一个cluster，会为每个cluster的服务或者客户端生成一个事件；实际的事件表在<DeviceName>\_endpoint\_config.h文件**（这里应该是错误的，实际工程中查看似乎在workspace/<DeviceName>/af-gen-event.h->EMBER\_AF\_GENERATED\_EVENTS）.**

它被应用框架的app/framework/util/af-event.c文件包含并使用。

说明：事件表是编译时生成并且是静态的；因此，在运行时无法随机创建和移除时间表；事件表入口必须存在，以便代码可以管理事件的调度。

**8.3 cluster事件如何调度**

插件和应用代码可管理cluster事件表中的那些事件，通过使用应用框架的事件管理API;这些API包含两个函数，emberAfScheduleClusterTick 和emberAfDeactivateCluster-Tick。

Tick 单位大约是1mS（0.9765625mS）;当然，这个值和时钟的精度是有关的。

**8.3.1 emberAfScheduleClusterTick**

这个函数使用EPID，cluster id,sever/client标识寻找时间表中相关的事件；事件表在**work**

**space/<DeviceName>/af-gen-event.h->EMBER\_AF\_GENERATED\_EVENTS中，原文档貌似有错误；**如果找不到，返回EmberStatus EMBER\_BAD\_ARGUMENT给调用者；如果找到的话，它调用通过调用者传递的时间安排调度事件，并返回EMBER\_SUCCESS。

EmberStatus emberAfScheduleClusterTick( int8u endpoint,

int16u clusterId,

boolean isClient,

int32u timeMs,

EmberAfEventSleepControl sleepControl);

EmberAfEventSleepControl参数允许调用者指示在事件表中事件处于活动状态时设备可能执行的操作（**我理解，周期等待+事件Fire+调用处理函数这三个时间就是所谓“活动状态”，直到事件被deactive函数去活动状态为止**）。此值仅与休眠设备相关;它对不休眠的设备没有影响。EmberAfEventSleepControl的可能值在app/framework/include/af-type.h中枚举,如下所示：

* EMBER\_AF\_OK\_TO\_HIBERNATE：意味着事件在“活动状态”下，应用程序可以进入深度睡眠状态，直到事件Fire；如果调度代码不关心这个设备的完整性，可以使用这个睡眠控制值。
* EMBER\_AF\_OK\_TO\_NAP：意味着该事件“激活状态”期间，设备可以睡眠，并周期性的唤醒poll需要的数据；如果调度代码需要设备周期性POLL的时候，需要使用这个睡眠控制参数值；在超时等待其他设备的回复的应用中这个参数是很有用的，如果是超时等待时，超时前你不希望设备进入睡眠状态，因为这样的话就无法搜到其他设备发过来的等待的消息了，那么超时等待将没有意义。
* EMBER\_AF\_STAY\_AWAKE：意味着事件“激活状态”期间，设备不会进入休眠状态；如果您正在安排一个非常频繁的事件，并且不希望设备在很短的时间内小睡，那么可以使用此事件，因为设备将在每次醒来时进行轮询。如果设备完全处于休眠状态，它将每秒轮询一次（短Poll周期可以在插件sleeply中修改）。

**8.3.2 emberAfDeactivateClusterTick**

该函数关闭一个事件；在调用调度事件时应调用此函数，以确保事件代码不会继续调用该事件。如果不再需要事件，也可以在调用事件之前调用它。

说明：在事件Fire之前，zigbee应用框架会自动调用该函数，以确保该事件不要再调用；

DeactivateClusterTick和ScheduleClusterTick类似，使用相同的参数；因为它也要定位到cluster tick的事件表中寻找对应的事件以关闭它。

EmberStatus emberAfDeactivateClusterTick(int8u endpoint,

int16u clusterId,

boolean isClient);

**九 属性管理**

**9.1 ZCL属性配置**

在zigbee应用框架中，属性存储管理通过两个C文件（app/framework/util attribute-storage.c和attribute-table.c），他们使用共同的头文件（appname\_endpoint\_config.h），APPBUILDER从应用配置文件中生成的这个头文件；endpoint配置头文件创建属性元数据和实际的属性存储。

关于属性存储，你有一些选项可以选择：

* 外部（存储）属性
* 连续存储（NVM）
* 独立存储
* 属性绑定
* 属性报告

**9.1.1 属性存储的端格式**

所有非字符串类型的属性存储的端格式，都应该与平台的端格式相同；对于Cprtex-M3内核的芯片，例如EM35X系列和确定的EZSP host架构，这意味著非字符串格式的属性是最低位在前的，即小端格式。

**9.1.2 属性存储端格式的实现**

Zigbee协议要求所有非字符串格式的数值在空中接口中以小段格式传输；对于EM35X EFR32和其他小端格式平台，在存储和空中数据传输时，属性是不需要进行字节交换的；然而，如果zigbee应用框架运行在打断格式的处理器中，例如确定的UNIX HOST系统通过EZSP-UART，在属性通过空中接口传输时将执行字节交换，以保证空中接口中的数据是小端格式；

上一节说，属性应该以正确的格式存储，因为在从本机类型或字节数组写入属性表时，不会执行任何字节交换。因此，由用户负责确保那些表示Zigbee整数类型但不直接映射到诸如int16u或int32u等本机类型的字节数组是按照应用程序平台的字节顺序表示的。

如果你的应用程序可能运行在不同端格式的平台中，你可能需要通过#define BIGENDIAN\_CPU检查端格式，它由随同zigbee协议栈打包的HAL层提供。

例如：考虑到简单测量服务插件的测试代码（位于app/framework/plugin/dimple-metreing-sever/simple-metering-test.c）；这个测试代码从属性表中拉出每天总结的测量属性，更新他们并且放回属性表；不行的是，每天总结属性是一个zigbee-48位的无符号整形数，它不是原生的数据类。

Zigbee HAL针对EM35X 和EFR32家族的处理前，如果非原生的数据类型（例如int48u）可以读和简单的操作；造成的结果是，属性必须先读到细节数据，并且字节数据必须被操作，以便可以写回属性表；在这些操作中，开发者需要记住的重要一点是在EM35XX SoC中，属性的存储方式是低位在前，所以操作过程必须准寻这个规范，否则，另一个网络中的设备要通过空中接口读属性将会造成错误。

注意：对于EZSP接口主机的应用程序，因为所有属性的存储基于host处理器，所以对属性存储的影响主要是主机的端模式。

**9.1.3 外部属性（E）**

你可能希望存储一些属性值在应用框架之外的位置；这种类型的属性最常用的场景是它们每次的读操作都要从硬件中读取；这种情况下，没有理由需要将属性再拷贝到应用框架中的RAM空间，因为它们浪费了宝贵的RAM。

通过在AppBuilder GUI属性后面，选择“E”复选框可以实现属性存贮在外部位置；属性元数据将被标记指示应用框架将不保留存储器RAM空间存储属性；等属性读、写时应用框架访问他通过调用emberAfExternalAttributeReadCallback和emberAfExternalAttributeWrite

Callback。

**说明：一旦设计一个属性作为“外部属性”，那么这两个回调函数会自动包含到你的call**

**back.c文件中**。

对于外部属性的操作请求，应用程序将立即响应请求；目前没有任何状态机与访问外部属性相关；例如开始一个读操作，然后一分钟后再次调用回调函数去看读操作的执行情况。

所有不能立即返回或者立即更新的属性，都不适合外部化；对于这种情况，siliconlabs建议你将属性包含在应用框架存储中，在应用框架中指定一段时间更新属性值，通过调用回调emberAfMainTickCallback实现。