Weave Architecture Overview

Weave is an end-to-end Internet of Things (IoT) platform that enables devices, services, and clients, such as mobile apps, to seamlessly interact and form a complete IoT system and thus create a thoughtful home experience.

- Overview
- Weave design philosophy
- Theory of operation
    - Weave schema organization
    - Resource distribution
    - Resource proxy controller
    - Fanout
    - Complete example
    - Hosting controllers on devices
- Resource model
    - Properties
    - Commands
    - Events
    - Putting it all together
    - Devices
        - Device traits
    - Groups
        - Location
        - General-purpose groups
    - Users
- Real-time protocols
    - Weave/WDM
    - Weave Web API
- Actors
    - Devices
        - Constrained endpoints
        - Hubs
        - Border routers
    - Nest Service
    - Clients

System components
- Pairing
  - Pairing process
- Data management
- Controllers
- Logging
- Security
- Accounts
- Firmware update

Implementations
- Weave SDK
- Nest Apps SDK
- Weave developer tools

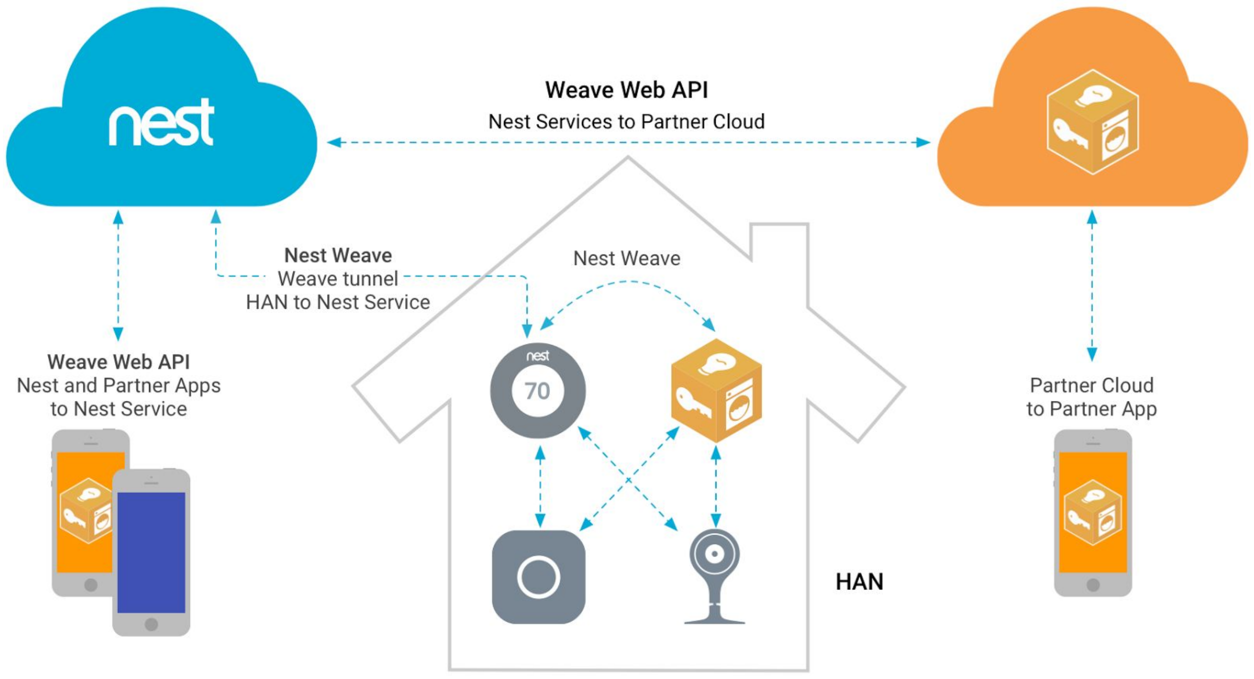
Overview

Weave is a distributed computing IoT platform targeting the thoughtful home that comprises low-power and less-constrained devices, along with the services to support them.

The Weave platform includes:

- A Home Area Network (HAN), where devices built on the Weave device SDK can intercommunicate via the Weave low-power protocol.
- The Nest Service, which facilitates communications between the HAN and internet clients, manages the home graph, and hosts controllers that together create the thoughtful home experience. The Nest Service communicates with the HAN using Weave and to web clients using the Weave Web API.
- Clients—for example the Nest mobile app, built on the Nest App SDK—which communicate with the Nest Service or directly with the HAN to provide front-end experiences to users.

All three entities and their relationships are illustrated in the following figure.



The Weave system works as a heterogeneous distributed computing system implemented on multiple types of hosts, such as battery-powered microcontroller-based devices, line-powered Linux-class devices, and powerful cloud-hosted services. The protocols interoperating between these hosts support a number of important capabilities including:

- Real-time distributed data management and synchronization
- Command-and-response control
- Real-time eventing
- Historical data logging and preservation
- Cryptographically controlled security groups
- Time synchronization
- Software updates

Underlying the components and protocols is a strong data model that formally describes the various entities and their capabilities. The resource model drives code generation for devices, services, and client apps to provide an unambiguous understanding of the various entities in the home and ensure correct-by-construction communications between them.

Through these mechanisms, the various components of the Weave system work in concert to provide a secure, reliable, comprehensive platform upon which the thoughtful home experience is built.

Weave design philosophy

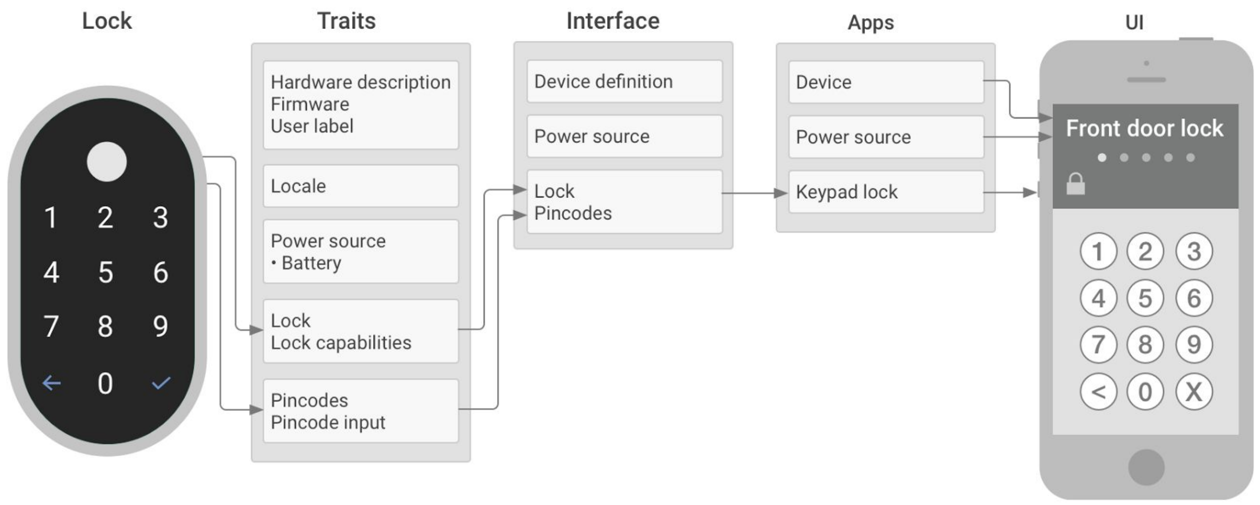
- Extensibility
- Sensitivity to low-power device considerations
- Ability to flexibly handle various configurations of devices with graceful degradation of capabilities
- Heterogeneous distributed computing system
- Security considered from the ground up
- Development experience allows for rapid prototyping and development of new device types and services

Theory of operation

The foundation of Weave is a comprehensive formal data schema for describing the functionality of devices and the services and clients that interact with those services. The schema describes small composable units of functionality called traits, which comprise properties, commands, and events, along with a semantic understanding of how these operate to implement the functionality. Traits can be composed into interfaces, which indicate how the basic units of functionality work together to form a higher level of functionality. For example, burners, controls, and on/off indicators may together form a stove interface. Traits and interface instances are collected into resources, which describe all the functionality for an actor in the system, such as a device or room in the home.

Weave schema organization

The following example illustrates the Weave schema organization.



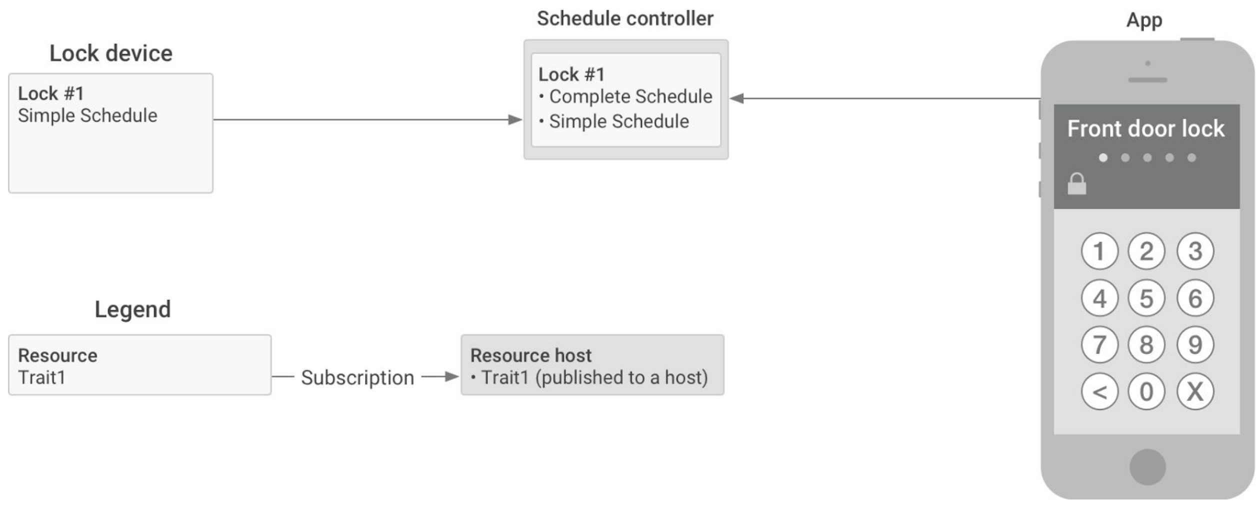
The front door lock on the left is composed of a list of individual traits that define the various capabilities of Lock, such as its hardware description, power source, and pincodes to unlock the door. Interfaces are then combine selected traits that operate together to provide a specific function; for example, Lock plus Lock capabilities combine to form a Lock interface. These can be arbitrarily mixed together, so a Lock interface, Pincodes interface, and a Schedule trait can form the integrated functionality of a Keypad Lock. The apps user interface can then be developed against these higher level interfaces. This enables tremendous flexibility, as a particular device is defined in terms of its individual capabilities, while a client can easily understand how these low-level capabilities work together through the interfaces.

The example above considers two resource types, a device and a structure (generally a location resource). Other types of resources that can have traits include users, fixtures (doors, windows), and annotations (well-known names for things with localized UX text, icon, and voice representations). Almost all functionality in the Nest ecosystem for everyday operation is mapped onto resources and their traits, with the exception being non-data model protocols such as streaming media. For example, instead of creating a specific API in the Nest Service to expose weather information, a weather trait would be added to a structure resource for use by any other resource needing to understand weather.

For both devices and apps, Nest provides an SDK that manages real-time communications as well as pairing, security, logging, software updates, and other aspects. The resource definition is used to generate code on top of these SDKs, which further simplifies construction of a device while ensuring correct-by-construction real-time networking. A lock device will have C++ classes defined for all of its traits. This includes the tags needed for profile and field identification in Weave communications. Similarly, an app can generate code for a particular interface that includes classes for all of the traits associated with that interface. In both cases, application code is written on top of the generated classes, and the networking is taken care of automatically.

Resource distribution

The implementation of a resource described by the schema is often distributed across the Weave system at the trait level. This is accomplished through distributable units of functionality called controllers, which expose and consume various resource traits to perform their function. For example, a lock implemented on a basic microcontroller might only be able to use a simple scheduling trait for access times, while a more comprehensive schedule trait is used for a high-quality app experience. A controller translates between the simple schedule for consumption by the lock device and the comprehensive schedule for the app. When the device is brought online, the Nest Service invokes the controller, which then provides its portion of traits for the device resource.

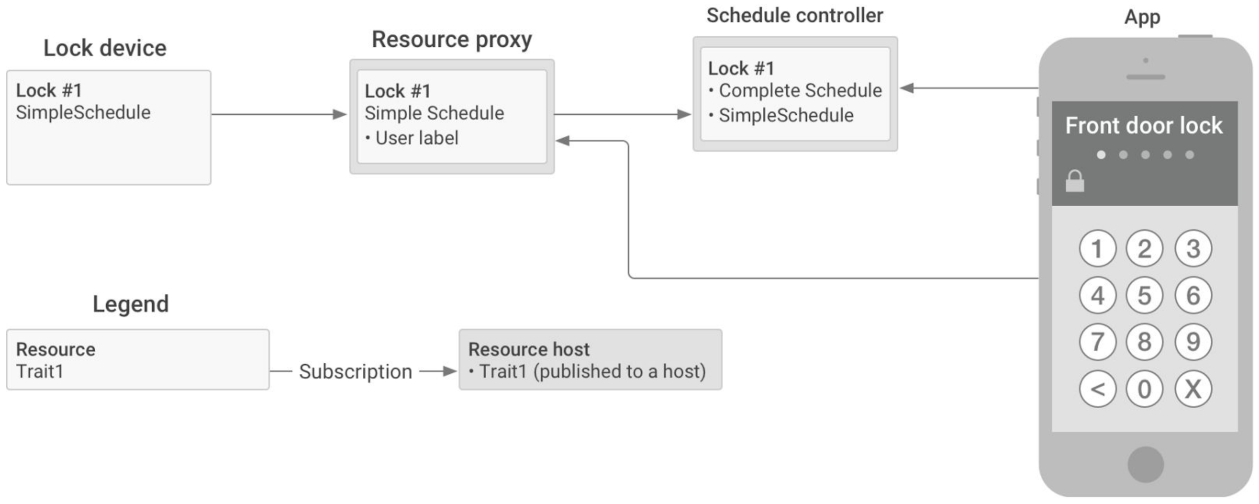


In this diagram, the boxes represent resources with their various traits listed inside, while the surrounding box indicates a resource host. Traits marked with an bullet specify that the trait is published on the given host, while arrows indicate subscriptions to the trait from consumers. The lock device is a host as is the Schedule controller, and they are publishing various traits that make up the Lock #1 resource. In this simple case, the Schedule controller hosts both the Complete Schedule trait, which is eventually consumed by the App, and the Simple Schedule trait, which is consumed by the lock device.

Resource proxy controller

Several different types of controllers provide functionality and intelligence in the home to augment what the devices are capable of doing. One type of controller, the resource proxy, is important for both sleepy low-capability devices and for remote control of HAN devices. This controller proxies the traits of a device, providing reliable access to the device's data, which may otherwise be inaccessible because the device is sleeping, remote on an unreliable WAN connection, or not capable of servicing requests for its data.

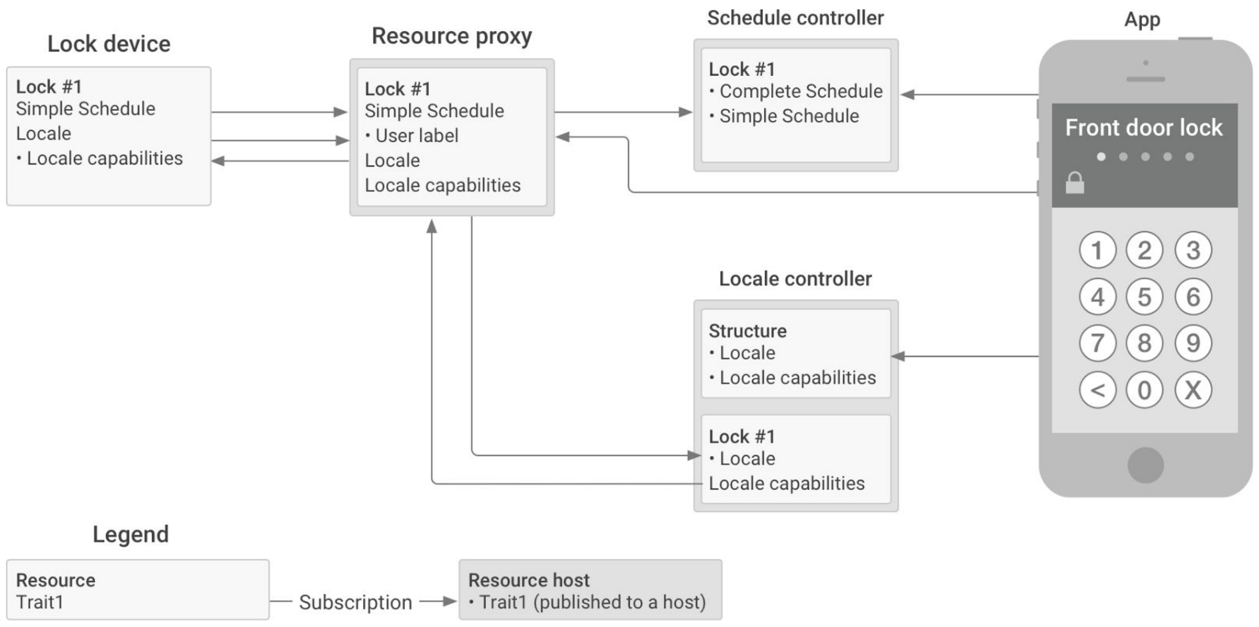
In general, the Nest Service creates a resource proxy for all devices in the home, and is able to manage device properties and command queues regardless of the device's connectivity. The Nest Service also gives the simple device a single point of connection to the Nest Service without the device having to understand the topology of additional controllers. The Nest Service can also publish simple traits that the device may not care to manage, such as the user's label for the device. This proxy can be inserted into the previous diagram, and the result is shown here.



Fanout

Another use of a controller is to manage fanout. For example, a device may require a locale trait to determine which language to speak or display. Normally a user would manage this on each device, but a fanout controller can manage locale for different areas within a structure or across the entire structure.

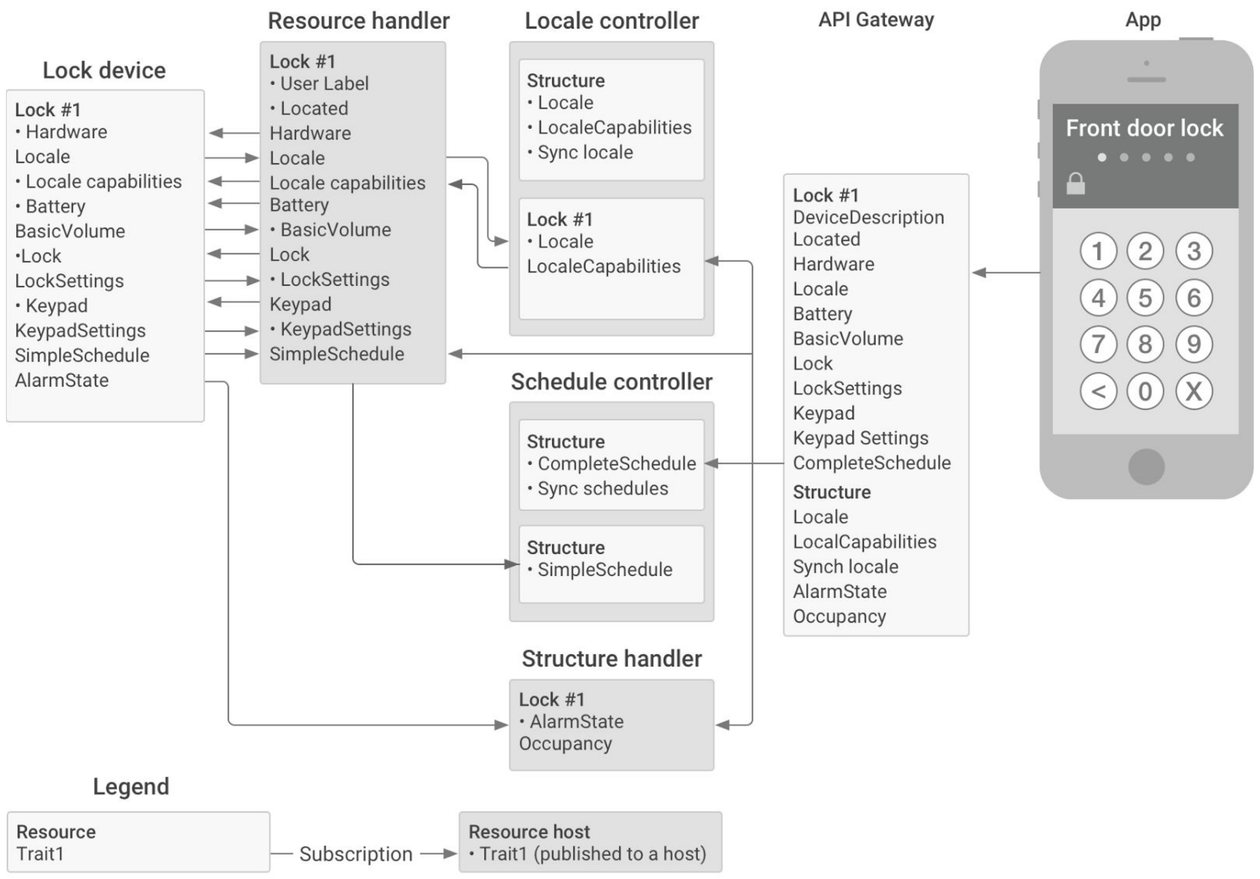
The controller provides a locale trait for the structure resource and a locale trait for each device resource that is part of the structure. The resource handler for the device points to the controller for the locale trait, and without any change to the device, its locale is now managed at the structure level. This arrangement is shown here.



In the diagram, the App is communicating with different hosts to access the various traits of the lock and structure resource. The Nest Service collects all of these traits into a single unified resource view, and presents the resource view to clients. The app doesn't need to understand the distributed system servicing each resource, but it can talk to a single API exposing all the resources to which it has access.

Complete example

A typical complete example is illustrated as follows.



The distribution of traits between a device and various controllers is maintained on the resource definition using options that describe the configuration. The configuration defines which traits of the resource are code-generated for the device, and which controllers must be invoked to complete the implementation of the resource.

Hosting controllers on devices

In the previous examples, controllers reside in the Nest Service. As a distributed system, however, the controllers can also be hosted on hub-like devices that reside in the HAN. The hub-like devices can advertise their controller capabilities to the Nest Service, which dynamically coordinates where the controllers reside for a specific resource. For example, an HVAC zone controller that is hosted in the Nest Service can be dynamically moved to a newly installed capable thermostat device.

This description covers the general real-time communications model used for day-to-day normal device, Nest Services, and apps interactions. The Weave architecture also includes a number of infrastructural components needed for IoT operation, including pairing, security layers, and software update, which are described in the rest of this document and other Weave documents.

Resource model

A resource describes an entity in the Weave system that can have traits associated with it. Traits are collections of properties, commands, and events that describe a small unit of functionality. They also include ancillary data model such as enums, statuses, and complex types.

Properties

Properties are characteristics of a trait that represent the state of a resource, for example, the brightness of a light or whether someone is home. Properties are strongly typed. A label and numeric tag are used to communicate each property. The properties may also have metadata indicating constraints, readability, and serialization specifics. Properties are operated on using the state management commands and three events: `observe`, `update`, and `notify`. State management is the primary means of command and control for Weave devices.

Commands

Commands are requests for action sent to a trait, with an expected response. While most operations are managed using properties with the standard state commands `observe` and `update`, a subset of operations are not amenable to property management. For example, `NextTrack()` on a media player trait requests changes to properties that the client could not compute, and `ScanNetwork()` on a Wi-Fi trait is requesting results with no property effects. A command has request parameters and expected response parameters.

Events

An event is a unidirectional assertion of truth about a trait at a specific time. This is the mechanism used to asynchronously communicate occurrences on the trait to other actors in the Weave system. Most events are related to property changes and communicated with the standard `notify` event. Other events can communicate arbitrary non-property-based occurrences, such as ringing a doorbell. Multiple events may be correlated to enable composition across traits; for example, a pincode entry event can be correlated with an unlock event, indicating that the pincode caused the door to unlock. Events consist of parameters and a timestamp.

Putting it all together

Properties, commands, and events work together to define a comprehensive mechanism for interacting with traits. Settings and actions are primarily handled through property management, with commands for handling requests that are not amenable to properties.

For example, a brightness property on a lighting trait is updated by a client to change the brightness on a physical device. An operation such as scanning Wi-Fi networks or skipping a track on a music player uses a command. Notification of changes on a specific entity are conveyed through property notifications and general-purpose events, which clients may subscribe to and monitor.

Devices

The main resource in the Weave platform is the device. The schema for a device is composed of the operations controlling or reporting on its physical operation, settings that guide how it operates, and general device properties.

Device traits

A typical device resource schema includes multiple trait instances, both general and specific.

General capabilities common to most devices:

- Device, hardware, and firmware description
- User label and location
- Network interface configuration
- Heartbeat and liveness
- Locale and timezone settings
- Power sources configuration

Specific functionality of the device:

- Sensors
- Operational settings
- Control of the various physical capabilities of the device

Groups

A group is a resource that holds a collection of other resources. Groups may contain traits of their own, which are hosted by a controller in the Nest Service. For example, a group of ceiling lights that are intended to work together are collected into a group resource that operates as a virtual light, which can then be controlled by the app client.

Location

The most common use of groups is to define the home topology, including the structure, floors, rooms, and fixtures. Location group resources can include traits that apply globally to devices. The Structure group, for example, contains a number of traits that apply across a home:

- Locale—language to be used on all devices

- Timezone—timezone to be used by all resources
- Occupancy—the home/away status of the home

For traits such as locale and timezone (device traits), a controller can host the device resources' traits to manage fanout from the structure trait. For occupancy, which is a property of the structure, a controller manages the trait for the structure resource.

General-purpose groups

Other groups are general in nature and can be used for various purposes. In the example of a group of lights, the group resource includes the same light operation traits as the individual lights, which are controlled at the group level by an app client. The associated controller is responsible for fanning out the light operations to the individual devices. A more complex heterogenous group (such as an HVAC zone resource containing thermostats, sensors, shades, and fans) contain HVAC-related traits. A complex controller coordinates the various devices.

Users

Users are considered resources for the purpose of managing properties specific to the user and referencing the user in events. For example, a user might have a trait indicating their pincode for access to various devices in the home, and a temporary guest user may include a schedule trait indicating when they are allowed access to the home with the pincode.

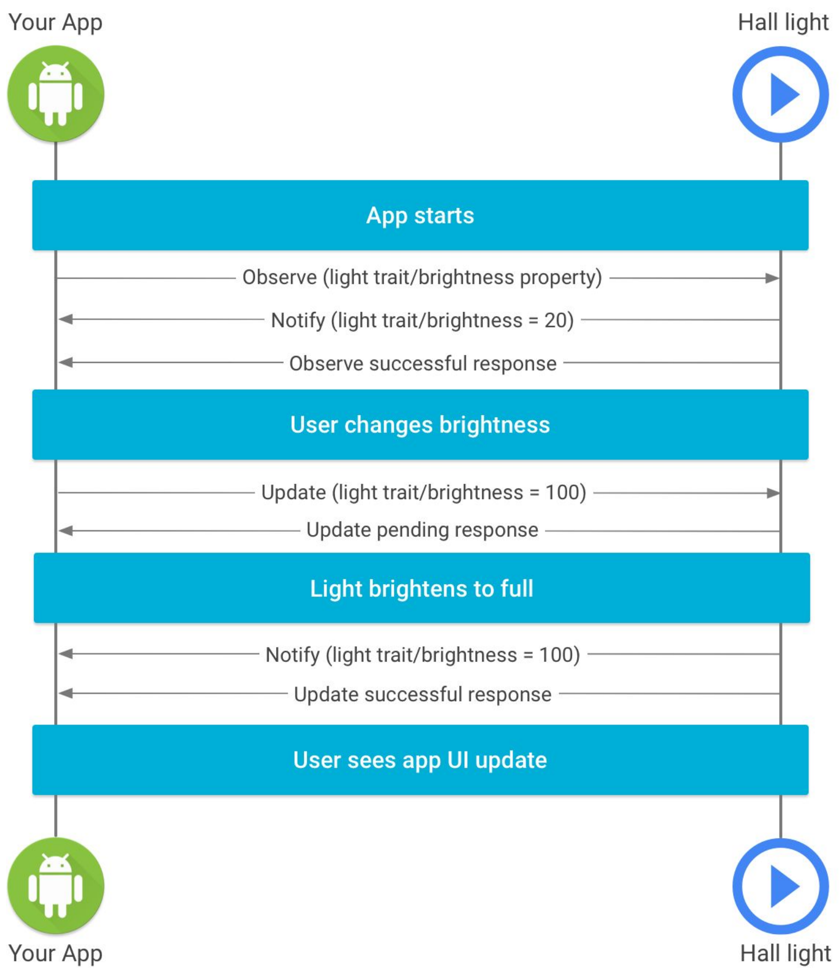
Real-time protocols

Communications across various hosts is managed using Weave real-time protocols. These include two real-time protocols: WDM on Weave for low-power HAN communications, and Weave Web API for messages with the Nest Service. These are asynchronous, fast, and relatively reliable. Both protocols include a similar set of capabilities that are bi-directionally translated within the Weave Tunnel. The capabilities include:

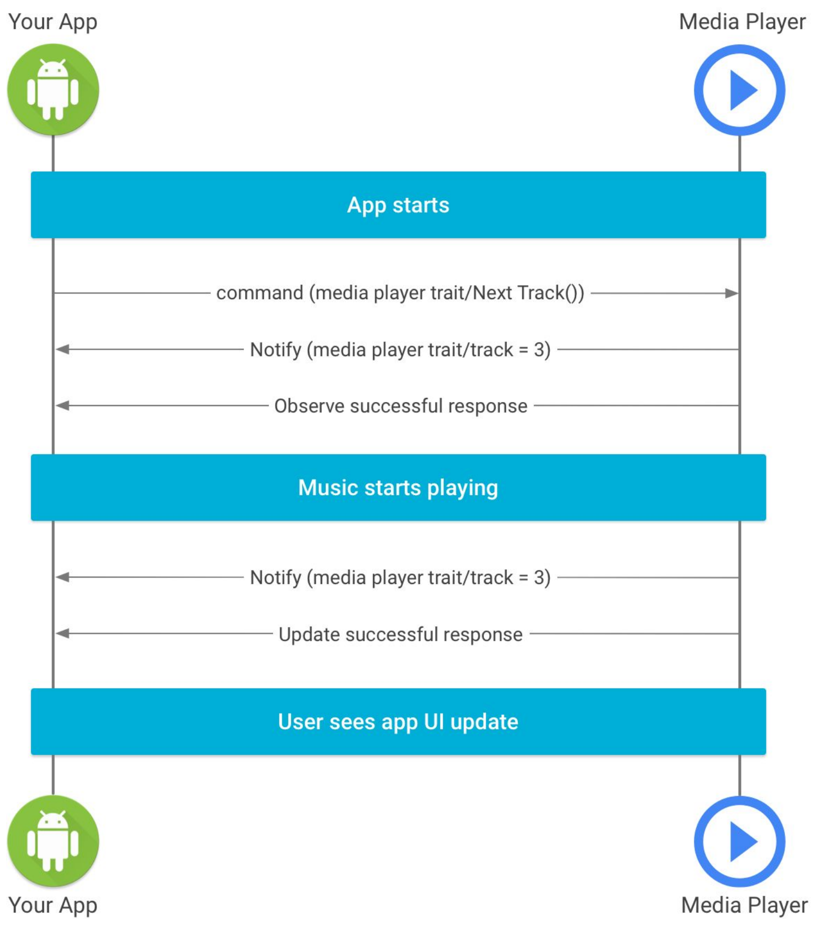
- Commands—requests for an action to be taken, with a life cycle ending in a returned result
  - Property management commands
    - `Observe`—fetch current property values and future changes
    - `Update`—request change of property values
- Events—assertions of truth or occurrences at a given time, with no lifecycle
  - Property management events
    - `Notify`—communicate a property change

The majority of interactions are accomplished through property management commands and events. Custom commands and events are available when property manipulation is not sufficient to convey an operation or an occurrence.

A general property manipulation sequence from a client to a light device:


Your App
Hall light
App starts
Observe (light trait/brightness property)
Notify (light trait/brightness = 20)
Observe successful response
User changes brightness
Update (light trait/brightness = 100)
Update pending response
Light brightens to full
Notify (light trait/brightness = 100)
Update successful response
User sees app UI update
Your App
Hall light

A general custom command sequence from a client to a music player:


Your App
Media Player
App starts
command (media player trait/Next Track())
Notify (media player trait/track = 3)
Observe successful response
Music starts playing
Notify (media player trait/track = 3)
Update successful response
User sees app UI update
Your App
Media Player

Weave/WDM

Nest's network protocol used within the HAN is Weave. Weave is optimized for secure operation on low-power networks and constrained, sleepy devices. Several application protocols called Weave profiles are implemented with Weave. The profile used for everyday real-time trait management is called Weave Data Management (WDM). WDM provides compact operations for the `update` and `observe` commands, providing asynchronous notifications of property state changes.

For the compact encoding of data, WDM uses a binary format called TLV (Type Length Value), which is optimised for effective deserialization on constrained devices. This format hierarchically addresses multiple properties within traits and indicates values for the properties depending on the operation.

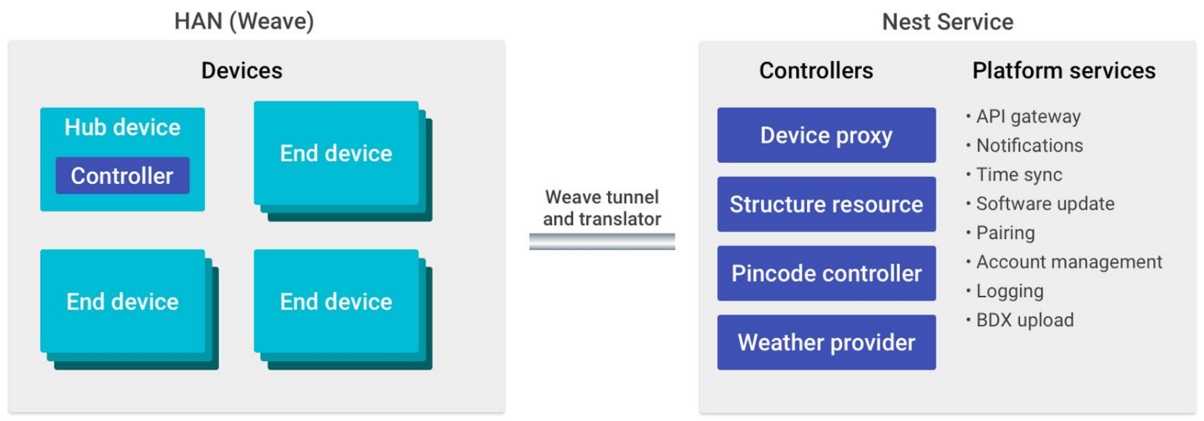
Weave Web API

Nest provides a comprehensive cloud-based API for real-time communications called the Weave Web API. This provides a fully asynchronous gRPC-based API for securely communicating with devices in the HAN and with the Nest Service. The API also contains interfaces for handling accounts, pairing, and other capabilities required for managing the home.

The Nest Service operates as the bridge between Weave in the HAN and the gRPC-based Weave Web API, and provides endpoints for Weave's low-power security and reliability mechanisms.

Actors

The protocols described above are used for communicating among various actors in the system. The actors have computational and network constraints that inform how the system is configured and where various capabilities are hosted.



Devices in the HAN on the left provide non-real-time services and communicate with Nest Service on the right through the Weave tunnel and translator. The Nest Service on the right provides real-time services using controllers and platform services to manage devices in the HAN.

The Weave ecosystem is designed to be flexible allowing users to install any combination of devices as they create their thoughtful home and enjoy the full functionality of their devices. Where devices are less capable, the Nest Service hosts controllers to augment the device with graceful degradation when the internet is down. When more powerful devices are installed, controllers can be moved into the home to provide better latency and reliability. The only dependency in the devices is that at least one border router device must be installed to enable Thread-only constrained devices.

Devices

Devices are physical products installed by the user into the HAN. Devices interact with the physical world and provide a number of capabilities but may be limited in various ways. The limitations can be categorized based on network capability and compute capability.

Constrained endpoints

Constrained endpoints are the most limited devices. For a number of reasons, such as battery operation, cost, and physical size, these devices tend to be constrained in network and compute capacity. They operate on a low-power network stack, such as Thread or BLE. They tend to run on RTOS with limited RAM and flash available. Finally, they tend to be sleepy, connecting only periodically to the network for updates to conserve power. Examples include battery-operated keypads, door locks, and remote sensors that are expected to operate for years on a single set of batteries.

Constrained endpoints are dependent on border routers to communicate outside the HAN. This implies that a border router device is present in the home to pair and operate the device remotely. Constrained endpoints may also depend on externally hosted controllers to supplement their functionality. For example, a lock that is only able to implement a simple scheduling trait for access may depend on an external controller to translate to a more comprehensive scheduling trait.

Hubs

Compared with constrained endpoints, hub devices contain more computing power than is necessary for their functionality. Hubs can host trait handlers and proxies for other devices in the home when the devices would normally be hosted in the Nest Service. Hosting functionality in the home can improve reliability when the user's internet connection is unreliable, improve latency of operations that would normally have to connect to the Nest Service, and potentially satisfy system and regulatory constraints around local access between devices.

Border routers

Border routers are devices that bridge the Thread network to the user's local area network for constrained endpoints, and create a Weave tunnel connecting the HAN to the Nest Service. A border router connects to both the Thread network and the user's LAN, generally via Wi-Fi. It has a consistent power source, so it is always online. Border routers typically also serve as hubs.

Nest Service

The Nest Service is a cloud-based infrastructure that connects the HAN devices into a comprehensive user experience. The Nest Service provides several key capabilities:

- Remote connection to HAN devices through a Weave tunnel
- Translation of Weave/WDM HAN protocols to gRPC
- Coordination of pairing between app clients and devices
- Management of accounts, users, and ACLs
- Management of relationships between resources (the resource graph)
- Hosting of non-device resources, such as locations and users
- Hosting of generic trait handlers
- Hosting of various special-purpose controllers and algorithms
- Connection to other cloud-based systems, such as weather providers
- Hardware and software certificate validation

Clients

There are client types that can interact with the Weave system. Mobile apps create a compelling user experience for setting up, observing, and interacting with the various components in the home. Cloud-based services can augment Nest-provided controllers, provide valuable additional inputs to the Nest experience, and bridge non-Weave device networks into the Nest ecosystem. Typically, clients interact with the Nest Service using the Weave Web API, although some clients may be able to connect locally directly to the HAN network. Most apps should be built using the Nest Apps SDK.

System components

Pairing

Weave pairing involves securely joining a new device to a user's home network. While pairing is intended to be as easy as possible, there are two crucial requirements. First, it is critical to ensure that the owner of the HAN intends to pair the device. Second, the device cannot be hijacked and paired away from its intended network. While there are several special-purpose pairing flows used in specific circumstances, this section describes the typical pairing flow.

All devices that connect to a Weave network must be factory provisioned with a hardware certificate, which is used to authenticate the device and authorize its access to various security domains. The certificate is provided by Nest when a device is certified, and is used to establish secure connections with services and other devices in the network when the device is paired.

Pairing process

1. **Establish that an app client user has the physical possession of the device being paired, and the app client wants the device to be paired to their network.**

   This is accomplished using a QR code affixed to the exterior of the device.

   The QR code contains the Weave Device Descriptor. The user scans the QR code or enters the pairing code, and this enables Password-Authenticated Session Establishment (PASE) between the app client and the device. The session is used for subsequent secure data transfer, ensuring that the pairing process cannot be subverted.

2. **Perform network provisioning, which provides the credentials needed to join the HAN wireless networks.**

   This is handled by the Weave Network Provisioning profile, which provides the tools for scanning, configuring parameters, and enabling network connections.

   For Wi-Fi, this involves transmitting the SSID, pre-shared key, and security type.

   For Thread, this involves getting the network name, PAN ID, and key to the device. As this information is available to other existing Thread devices on the network, the information is obtained from an existing Thread device and transmitted to the new device. If there is no Thread network in the home, the first Thread device creates a new network.

3. **Perform fabric provisioning, which involves joining the device to an existing Weave fabric.**

   As with Thread, an existing Weave device passes the fabric configuration to the new device, including the fabric ID and key. The new device’s hardware certificate is authenticated to ensure that only certified devices are able to join the Weave fabric.

4. **Provision the device to the Nest Service.**

   This involves the client app contacting the Nest Service to obtain a pairing token. The token and account and service IDs are passed to the device. The device connects through the network to the Nest Service and establishes a Certificate-Authenticated Session Establishment (CASE) session using the hardware certificate. The device requests provisioning over the secure session using the pairing token and account information. When this is complete, the device is associated in the Nest Service with the user's account. The client app is then notified that the device has been paired.

After the device is paired, it usually requires setup of the initial configuration. This includes a variety of general settings, such as structure/room location and other device-specific configuration and calibration that make the device operational. These are handled through standard data management mechanisms. Once the device has been configured, it is marked as operational and becomes a member of the home's device network.

Data management

Once a device is operational, its everyday operations are handled through communications with other devices, with the Nest Service controller, and with the client app using real-time data management protocols. The protocols communicate commands, events, and property management-specific commands and events. The HAN uses the WDM protocol on Weave, while the Nest Service uses the Weave Web API. The Weave tunnel endpoint includes the translation between the two protocols.

A device’s traits can be published on other devices or by resource handlers and controllers hosted in the Nest Service. This allows for constrained devices to offload functionality, Nest Service-based handling of settings-like traits, and group management of common traits. The Nest Service collects all disparate trait handlers for a resource under a single addressing scheme, hides their distributed nature, and provides a single access point for the resource.

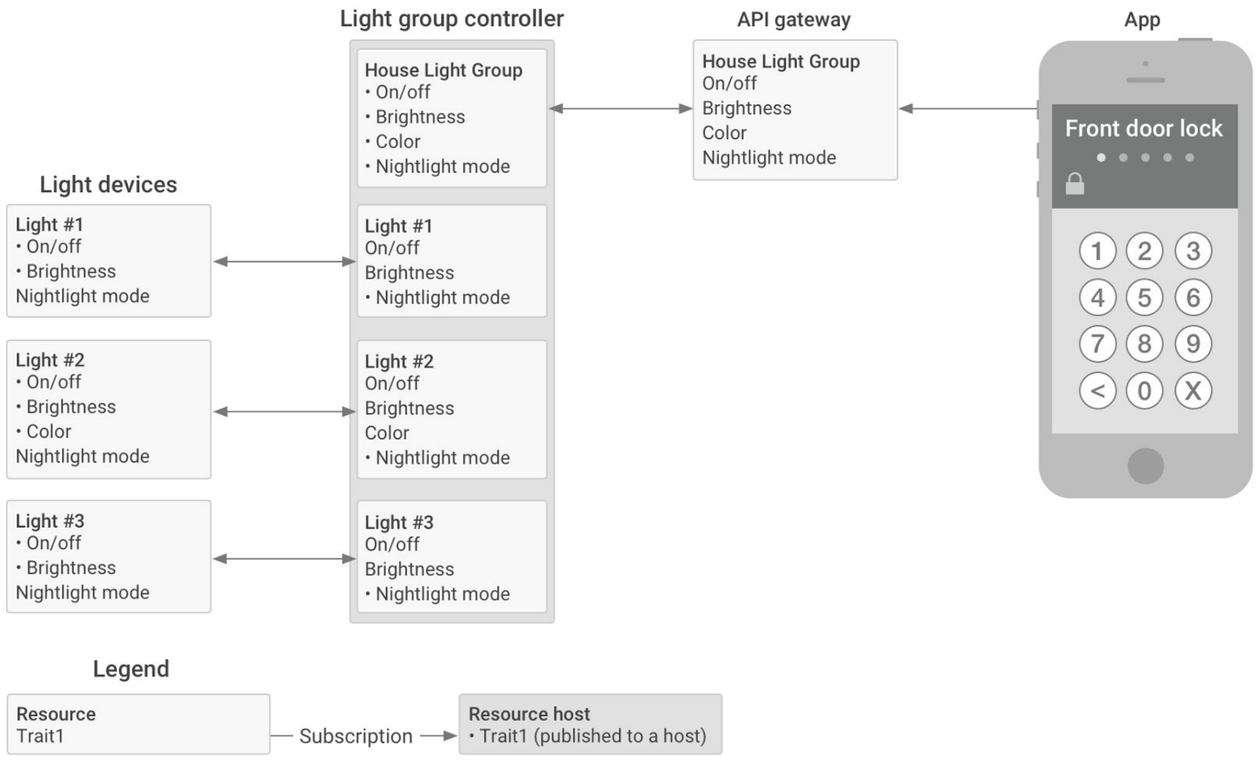
Controllers

Enabling devices to communicate with each other and with a client application is just the beginning of building a comprehensive home experience. A set of functionality outside of devices and app clients is necessary to build a fully automated home experience. This functionality is provided by controllers. Controllers fall into several broad categories:

- Arbiter
  - Coordinates disparate inputs to produce a common state
  - Example: Home Occupancy derived from many inputs
- Complexity
  - Provides intelligence for dumb devices
  - Example: Converting a basic schedule into a complex schedule
- Fanout
  - Coordinated control of common capabilities across a set of homogenous resources
  - Example: Controlling a group of lights together
- Coordinator

  - Provides upleveled capabilities, implemented by coordinating across multiple types of device
  - Example: HVAC zone controller
- Adapters
  - Bridge other network protocols' devices to Weave
  - Example: UPnP adapter
- Resource proxy
  - A properties and commands proxy for a resource whose primary host is not reliably available due either to device sleepiness or network unreliability
  - Example: Nest Service proxying of HAN device traits

Controllers expose their functionality through traits on resources, either hosting a trait for a particular resource or operating on/subscribing to a trait hosted elsewhere. For example, a lighting group controller would expose lighting traits for the lighting group's resource and fan out to the individual lights in the group, either by hosting remote traits for the devices or sending updates to device-hosted traits. A client would then talk to the lighting traits on the group resource to control the lights. The light devices operate identically whether they are controlled directly from an app or by a controller.



In this example, a light group controller manages a set of three lights in the HAN. The on/off, brightness, and (on some lights) color traits are managed by the controller passing update commands. The Nightlight mode setting for each light is published by the controller, and the lights subscribe to it. The app, however, simply deals with a House light group resource, which allows it to manipulate the same on/off, brightness, color, and nightlight traits together. This example shows a simple fanout controller; however, more complex logic can implement more interesting functionality, including exposing a different set of traits from those on the devices.

Controller functionality can equivalently be hosted on a device node or in a service host within the Nest Service. The Nest Service orchestrates where a specific controller is hosted based on the capabilities of devices available in the HAN for hosting, and any other requirements associated with a given user experience. In the example above, hosting the Light group controller in the home would likely lead to improved latency and synchronous control of the various lights.

Logging

In addition to real-time data management, Weave provides two mechanisms for managing historical operations data as well as traditional logging for debug and diagnostics data. While subject to device and network constraints, the mechanisms can reliably convey important events and preserve state edge to the Nest Service where they are persisted and accessible through the Weave Web API.

Properties and events within the Weave schema can be tagged with options indicating how they are logged and persisted. This includes a log level ranging from production to debug and an urgency level. The persistence of events marked production is considered to be fundamental to the operation of the device, and are transmitted and persisted to the best ability of the device and the network. In normal operating mode, debug events are not transmitted or persisted; however, the device can be temporarily put into debug mode where events are transmitted on a best-effort basis. These log events can be accumulated and transmitted in batch form to optimize between device RAM and network constraints. Once received, the events are persisted in the Nest Service, and are accessible through the Weave Web API.

Weave also supports the uploading of bulk data to the Nest Service using the BDX Weave protocol. This provides complete flexibility for creating arbitrary logging, diagnostics, and other information types to be uploaded in batch form and then persisted.

Security

Security is a fundamental component of the Weave architecture. While network protocols (such as Thread and Wi-Fi) include encryption, encryption is not sufficient for managing the nuances of securing different classes of devices and services communicating in an IoT system. Weave security must work efficiently on very constrained devices. To that end, Weave provides a layered security approach built on top of a few lightweight authentication and encryption technologies.

HAN security includes Weave certificates, which authenticate devices, Weave service endpoints, and device firmware updates. These are rooted in a trusted Nest certificate authority that enables devices to prove their authenticity when joining the Weave fabric and their identity to Nest Services.

The certificates establish secure sessions by exchanging keys using the Certificate-Authenticated Session Establishment (CASE) protocol. A session secures communications between two nodes in the system, generally two devices or a device and a Nest Service endpoint. The session can be a single transaction to several weeks of communications, depending on the security needs of the communications. Sessions are not intended to live indefinitely; a node at either end of the session can invalidate a session key at any time.

The second kind of key used by devices is the group key. These are multi-party keys that are long-lived and subject to rotation through a common epoch key. The most general group key is the fabric root key, which is shared by all devices in the Weave fabric, and which can be used for general communications across devices and with Nest Services endpoints. Other group keys are used to create secure domains, in which only device that are certified to be part of that domain can participate. For example, the physical access group is used to encrypt pincodes and sign secure physical lock-related commands between authorized devices and clients. Access to these keys is controlled by the Nest Service; however, the Nest Service does not have all of the key material required to participate in the security domains. This provides compartmentalized security. In this example, the Nest Service and devices that are not certified in the Physical Access domain cannot decrypt pincodes created on the app client for use on physical door locks.

Regardless of the provenance of the keys, they are used to encrypt and/or integrity check various fields of a Weave message, or portions of the application payload.

Accounts

The mechanism for client identification, authentication, and authorization with the Nest Service is the Nest Account. An account is authorized to access a given set of resources, including structures, devices, and users, and in most cases corresponds to a single Nest user. The Nest Account can also contain OAuth tokens that authorize external client services and applications to access a subset of the resources.

Firmware update

Weave provides over-the-air (OTA) firmware updates for conveying firmware update versioning as well as a Bulk Data Transfer (BDX) service for managing binary firmware downloads to devices. The Weave Software Update profile provides mechanisms to query devices for their current firmware version, and to specify the desired firmware version across a fleet of devices at any granularity. Once a firmware update has completed certification, the image is signed and uploaded to the Nest Service where the developer can manage its distribution to devices in the field.

While Nest does not prescribe an update mechanism on devices, it does require a mechanism that is resilient to interruptions or failures at any stage of the process. Usually this is an A-B image mechanism implemented in the device's bootloader. To help developers meet these requirements, Nest provides a reference implementation for updating firmware.

Implementations

Nest provides several implementations that simplify building devices, mobile apps, and services for the Weave system.

Weave SDK

Nest provides a C++ Weave SDK implementation for both RTOS and Linux-class devices. The SDK handles pairing, networking, real-time networking, logging, and other aspects of device operation and management. A device developer chooses traits to be implemented by a given device. Code generation creates a simple C++ object model for each trait. The developer then writes code against the objects to implement them on the hardware.

Nest Apps SDK

For mobile app development, Nest provides a comprehensive SDK for Android, iOS, and the web that handles most of the complex protocol interactions needed to set up and manage resources, control devices, and deal with error conditions. The SDK handles connections to the Nest Service API. This includes the networking and persistence layers needed to manage devices and other resources. A code generation layer uses the Weave schema definitions to create an object model for all traits and interfaces with which the app interacts. An app developer writes code that interacts with language-native object methods, abstracting away the underlying network specifics.

Weave developer tools

Nest provides a suite of developer tools and example apps to assist in developing, debugging, and testing devices and the platform. The primary tool for debugging the HAN and Nest Service network is the Weave Data Explorer (WDE). WDE is able to connect to the Weave fabric in the home as well as the Nest Service through the Weave Web API. The tool visualizes the network of devices and controllers and allows interaction and debugging of the various resources and their traits. WDE also contains automation capabilities for creating and running tests and Nest's certification suite.

Nest also provides the Nest Home Simulator, which enables virtual devices and other resources to be created in a virtual structure. These devices can be manipulated by WDE to run automatically, which can test different combinations of devices without requiring the physical hardware.