Enabling Research Extensions in Matter via Custom Clusters

Ravindra Mangar, Jared Chandler, Timothy J. Pierson, David Kotz
Dartmouth College, Hanover, NH
{ravi.gr, jared.d.chandler, timothy.j.pierson, david.f.kotz}@dartmouth.edu

Abstract—**Matter is a recent interoperability standard that aims to address fragmentation in smart homes by providing a common system for integrating disparate smart-home devices. As Matter adoption grows, it also creates a shared platform on which new smart-home mechanisms can be implemented and evaluated end-to-end across realistic deployments.**

However, turning a research idea into a runnable prototype in a Matter-based deployment is tedious. We address this shortcoming by presenting a practical template for implementing custom clusters in the open-source Matter SDK and invoking it from a widely used smart-home controller. Using a running example, we add a simple cluster that erases sensitive data stored on a smart device. We view this template as an enabling step for the community. While Matter's open reference implementation provides common ground, the concrete steps required to add and exercise experimental functionality remain scattered. Our template and walkthrough consolidate the necessary steps needed for a reproducible workflow that researchers can adapt for exploring new security and privacy mechanisms.

I. Introduction

Since its release in October 2022 by the Connectivity Standards Alliance (CSA), the Matter protocol [1] has rapidly gained backing from major smart-home platforms and device manufacturers as a standard method of providing interoperability across device types and across vendors. Matter version 1.0 supported 8 different device types (e.g., smart plugs or smart air conditioners) while the latest release as of December 2025 expands this catalog to 13 different device types.

Matter has already proven to be a growing commercial ecosystem. For some device types, millions of devices have already been deployed. For instance, Midea reports shipping over 2 million Matter-enabled air conditioners [2].

While Matter's commercial ecosystem has expanded, it remains difficult for researchers to leverage the protocol for exploring new functionality. The published research to date has primarily emphasized measurement, interoperability studies, and ecosystem characterization, rather than exploring extensions to the Matter ecosystem for new functionality or new device types. Many smart-home research ideas depend on behaviors that are not yet part of Matter's standardized feature set, creating a practical barrier to using Matter as an experimental medium. Our work aims to alleviate the burden from researchers from having to scour code repositories and documentation and introduces a reproducible template for prototyping new features in both the device and controller [3].

Workshop on Security and Privacy in Standardized IoT (SDIoTSec) 2026
23 February 2026, San Diego, CA, USA
ISBN 978-1-970672-01-5
https://dx.doi.org/10.14722/sdiotsec.2026.23064
www.ndss-symposium.org

In this paper, we make two important **contributions**:

- A guide for implementing new functionality on any Matter-compliant device which has a mechanism to allow firmware to be reflashed (e.g., development devices).
- A guide for invoking such new functionality from a smart-home controller.

Motivating Example: To motivate and illustrate our contributions, this paper uses a running example. Consider a researcher interested in the important task of *sanitizing* a smart-home device for reasons of security and privacy, that is, to erase any sensitive information stored within the device (e.g., identity of the home residents, their personal health information, or keys and certificates allowing access to cloud resources). The Matter standard does not include an "erase all sensitive data" functionality, so the researcher wishes to add this new functionality to the open-source Matter SDK and experiment with this function by flashing the customized Matter implementation on a Matter-complaint smart device.

To demonstrate how to extend Matter to include this erasure functionality, we use a smart microwave oven as an example. The oven may contain sensitive information, such as network credentials, that should be erased if the device is sold or recycled. The new Matter extension provides code to delete this sensitive information and supplies a standardized way for applications to invoke it.

II. Background

In this section we introduce Matter-specific terminology, used in the remainder of the paper, and illustrate their organization in Figure 1.

- Node: A single Matter device such as a light bulb, smart lock, or in our example, a microwave oven.
- Cluster:[1] A cluster specifies a group of related *commands* and *attributes*. In the Matter data model, a node exposes one or more *endpoints*, and each endpoint hosts instances of one or more clusters; thus, commands are invoked

[1]Matter's data model reuses the 'cluster' abstraction popularized by the Zigbee wireless specification.

on a particular cluster. Clusters are the primary unit of interoperability. They are also the unit a researcher/developer will most often use when designing custom functionality. In our example, we create a `Sanitize` cluster which includes functions to do the actual data erasure (*commands*).

- Endpoint: A logical interface within a node that groups a set of clusters (features). For example, an electrical power strip might have multiple outlets, and each outlet has a unique endpoint.
- Attribute: A piece of information the device can report when requested (like a status or sensor value), or a configuration parameter that can be set when requested. For example, a thermostat will have an attribute for the current temperature; the microwave oven in Figure 1 has an attribute `CookTime`.
- Command: A request message that asks the device to perform an action. Like attributes, commands have a defined data format (a typed payload with specific fields), and may return a typed response. An example of a command is `StartCooking` for a microwave oven.
- Controller: An application (most commonly, a smartphone app) that commissions and controls Matter devices. The controller uses the Matter protocol to contact device endpoints, issue commands, and read or set attributes.

III. Creating New Cluster

Continuing with our running example, we introduce a new cluster which defines one new command. Specifically, we add a custom cluster (`Sanitize`) that exposes a single command (`SanitizeDevice`). When a controller invokes the `SanitizeDevice` command, the device erases sensitive data stored locally and returns a success/failure status.

Importantly, the addition of this new cluster and command (or in other cases, multiple commands and attributes) does not alter other device functionality. As such, the device can still be commissioned and operated normally after the addition – including any Matter controllers that are unaware of the extended functionality. We provide our implementation as a template that researchers can reference when implementing their own custom clusters. We plan to open-source this template to facilitate reuse.

Although the new cluster and encompassing command we build follows Matter's data model and can be commissioned and exercised like a Matter node, we intentionally wrote our template for a research/testing audience using the open-source Matter SDK and reference implementation rather than a product-certification pipeline.[2] The key distinction is that our template produces an experimental research prototype rather than a CSA-certified commercial device; as a result some controllers may surface warning messages indicating the lack of certification.

[2]We used v1.5.0.1 of the Matter reference implementation found at github.com/project-chip/connectedhomeip

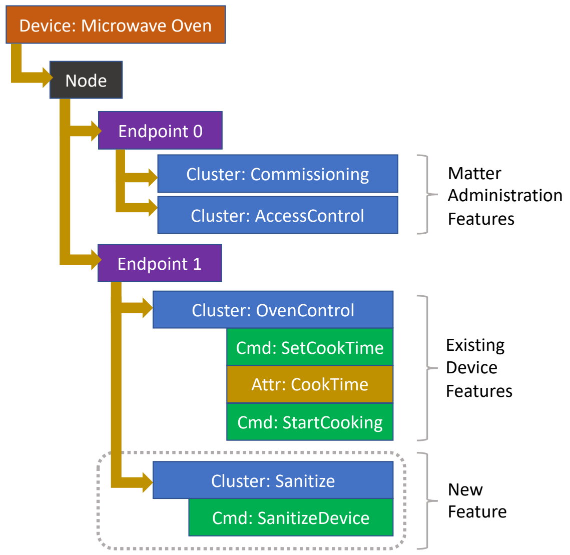


Fig. 1: Diagram of Matter's data model for a Microwave Oven. Dashed gray line indicates new functionality to be added. Adapted from [4].

A. On the Device

Here we outline the steps required to define a new cluster, implement its server-side behavior [3] (the code running on the device that receives requests from a controller, i.e., the logic for the command), and expose it on an endpoint.

We use the Linux `microwave-oven-app` example from the Matter SDK to embed our new functionality (i.e., the `Sanitize` cluster). While it runs on a generic Linux system (such as a Raspberry Pi), it appears logically to controllers as a smart microwave oven.

Step One: Define the Cluster: First, we extend the Matter SDK's data model to include a new `Sanitize` cluster. This extension signifies to the SDK the commands and attributes associated with our new cluster as well as the permissible data types. Concretely, we add a new cluster definition under `src/app/zap-templates/zcl/data-model/chip/` so that the cluster is recognized by the SDK and can be included as part of an endpoint. A snippet of our cluster definition is shown in Listing 1. After rebuilding the SDK is now aware of the new cluster and its capabilities.

Step Two: Implement the cluster on the device: The next step is to add the server-side code that handles the new command. In our case, the device must perform three steps: (1) receive a `SanitizeDevice` request, (2) erase whatever data the researcher designates as sensitive, (3) send a clear "success/failure" response back to the controller.

[3]Here, *server-side* refers to the device-side; we use this terminology for consistency with Zigbee's cluster model.

Listing 1: Matter cluster definition fragment defining the mandatory `SanitizeDevice` command in `SANITIZE_CLUSTER`.

```
<define>SANITIZE_CLUSTER</define>
  <description>Cluster to remove sensitive
    data from device.
  </description>
  <!-- Commands -->
  <command source="client" code="0x00" name="
    SanitizeDevice"  optional="false">
    <description>
      Deletes sensitive data.
    </description>
    <access op="invoke" privilege="manage"/>
    <mandatoryConform/>
  </command>
```

The cluster is implemented across the following files:

```
src/app/clusters/
└─sanitize-server/
    ├─BUILD.gn
    ├─sanitize-server.h
    └─sanitize-server.cpp
```

BUILD.gn is the build file that tells the project's build system to include this new `Sanitize` cluster in the firmware, which source files to compile and other parts of Matter it requires (for example Matter provides its own implementation of some cryptographic functions). It can also declare any extra C/C++ libraries this code needs (for example, the Boost library). The file `sanitize-server.h` is a header that lists the functions this cluster code provides; for example, a function to set it up when the device starts and a function that runs when the controller sends the `SanitizeDevice` command. The file `sanitize-server.cpp` provides the code for the functions that implement the new command. When a controller invokes `SanitizeDevice`, this code triggers a routine to erase whatever data is deemed sensitive by the programmer and returns if the operation was a success or failure (see Listing 2).

Step Three: Attach the new cluster: We next expose the cluster on a specific endpoint so the device advertises it and routes requests to the corresponding handler. This is done using the Zigbee Cluster Library (ZCL) tool as shown in Figure 2.

Concretely, we update the device's endpoint configuration to include the `Sanitize` cluster on a specific endpoint so incoming commands get handled by `sanitize-server.cpp`, as shown in Listing 2.

B. On the Controller

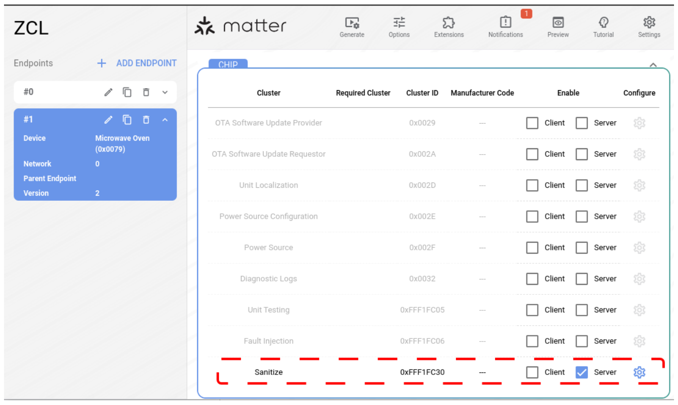
To exercise our custom cluster from a real smart-home ecosystem, we use Home Assistant [5], a popular open-source platform with native Matter support.[4] We first commission the modified device into Home Assistant as usual [8]. After commissioning, Home Assistant automatically discovers

[4]We use Home Assistant Core v2025.12.3 [6] with Python Matter Server v8.1.1 [7]

Listing 2: Command Handler that does the erasure of sensitive data.

```
void SanitizeCluster::InvokeCommand(
    HandlerContext & ctxt)
{
    switch (ctxt.mRequestPath.mCommandId)
    {
    case Commands::SanitizeDevice::Id:
        HandleCommand<Commands::
    SanitizeDevices::DecodableType>(
            ctxt, [](HandlerContext & ctx,
    const auto & ) {
                const EndpointId ep = ctx.
    mRequestPath.mEndpointId;
                CHIP_ERROR err = chip::
    DeviceLayer::PlatformMgr().ScheduleWork(
                    DoSanitizeDevice,
    static_cast<intptr_t>(ep));
                ctx.mCommandHandler.AddStatus(
                    ctx.mRequestPath, (err ==
    CHIP_NO_ERROR) ? Status::Success : Status
    ::Failure);
            });
        return;

    default:
        ctxt.mCommandHandler.AddStatus(ctxt.
    mRequestPath, Status::UnsupportedCommand);
        return;
    }
}
```



#1	
Device	Microwave Oven (0x0079)
Network	0
Parent Endpoint	
Version	2

Cluster	Required Cluster	Cluster ID	Manufacturer Code	Enable	Configure
OTA Software Update Provider		0x0029	---	☐ Client ☐ Server	⚙
OTA Software Update Requestor		0x002A	---	☐ Client ☐ Server	⚙
Unit Localization		0x002D	---	☐ Client ☐ Server	⚙
Power Source Configuration		0x002E	---	☐ Client ☐ Server	⚙
Power Source		0x002F	---	☐ Client ☐ Server	⚙
Diagnostic Logs		0x0032	---	☐ Client ☐ Server	⚙
Unit Testing		0xFFF1FC05	---	☐ Client ☐ Server	⚙
Fault Injection		0xFFF1FC06	---	☐ Client ☐ Server	⚙
Sanitize		0xFFF1FC30	---	☐ Client ✓ Server	⚙

Fig. 2: Once the Matter SDK is aware of the new cluster, through our earlier changes to the data model, we can add the new cluster to a device using the Zigbee Cluster Library (ZCL) tool. The change is applied via the GUI by updating build configuration files, so rebuilding the firmware incorporates it without any code edits. This makes the cluster easily reusable across Matter devices by enabling it through configuration rather than device-specific source changes.

Listing 3: Telling the controller of our new cluster. Values for the IDs come from the data model XML.

```
@dataclass
class SanitizeCluster(Cluster,
    CustomClusterMixin):
    """New Sanitize Cluster"""
    class Commands:
        """Commands for the Sanitize cluster.
    We only define the one to erase data"""

        @dataclass  """Erase Sensitive Data""
        class SanitizeDevice(ClusterCommand):
            cluster_id: ClassVar[int] =
    SANITIZE_CLUSTER_ID
            command_id: ClassVar[int] = 0x0000

            #This is a client to server
    command.
            is_client: ClassVar[bool] = True
            # Response just has a status
            response_type: ClassVar = None

            @ChipUtility.classproperty
            def descriptor(cls) ->
    ClusterObjectDescriptor:
                return ClusterObjectDescriptor
    (Fields=[])
```

the device's endpoints and standard clusters, but it will not understand our new experimental `Sanitize` cluster unless we add explicit support for it. In practice, this requires small changes across Home Assistant's Matter stack: we (1) register our custom cluster in the Python Matter Server backend and (2) add a custom component under `config/custom_components/` that exposes the command as a Home Assistant service (so it can be triggered from automations and displayed as a dashboard button).

Step One: Modify Home Assistant's Matter Integration: Home Assistant's Matter support has two parts. `Home Assistant Core` provides the UI, automations and the commissioning flow, while `Python Matter Server` is the backend service that is the logical Matter controller and sends commands to devices. To make Home Assistant aware of our custom `Sanitize` cluster, we update `Python Matter Server's` list of custom clusters by editing `python-matter-server/matter_server/common/custom_clusters.py` to register the new cluster and its associated commands.

Step Two: Add the Integration: To expose our custom `SanitizeDevice` command to Home Assistant, we implement a custom integration under `config/custom_components` and the associated files as shown below.

```
config/custom_components
└─matter_sanitize/
    ├─manifest.json
    ├─__init__.py
    └─services.yaml
```

The integration is a new directory consisting of: (1) a `manifest.json` that declares metadata, (2) an `__init__.py` that registers a Home Assistant service (`matter_sanitize.sanitize_device`) and forwards requests to the Python Matter Server, and (3) a `services.yaml` that describes the service schema so it appears in the UI and can be bound to a dashboard button. The contents of `manifest.json` and `services.yaml` follow documented Home Assistant practices to add a custom integration [9]; unique to our case is the addition of a `matter_sanitize` domain. Listing 4 shows the contents of `__init__.py`.

Listing 4: Home Assistant forwards the request by invoking the command defined in Listing 3.

```
async def handle(call):
    node = node_from_ha_device_id(hass, call.
    data["device_id"])
    #Invoke SanitizeDevice cmd
    await get_matter(hass).matter_client.
    send_device_command(
        node_id=node.node_id,
        endpoint_id=call.data.get("endpoint_id
    ", 1),
        cluster_id=0xFFF1FC30,
        command_id=0x0000,
        payload={},
    )
```

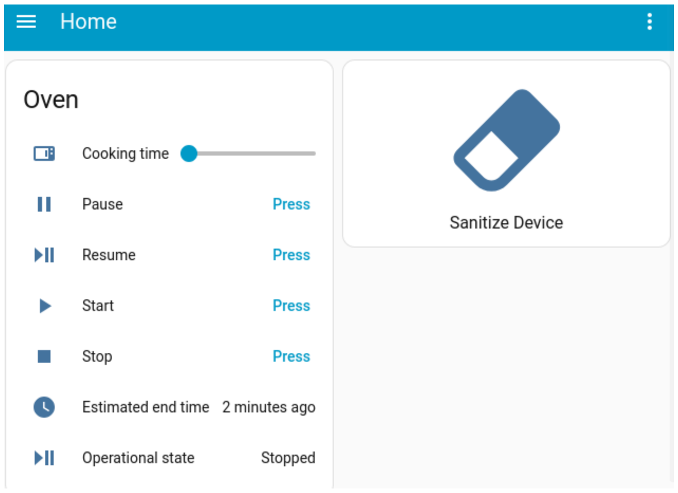


Fig. 3: Here we see our simulated microwave oven in Home Assistant with an accompanying button to remove sensitive data from the device.

Step Three: Create the UI element: We add a button to the Home Assistant dashboard (as shown in Figure 3) and configure its action to invoke backend service [10], [11]. The only project-specific detail is the service we expose and call (`matter_sanitize.sanitize_device`), which forwards the request and ultimately invokes our new `SanitizeDevice` command.

IV. Related work

Prior work has surveyed Matter's architecture adoption challenges, helping motivate research opportunities [12]. Mangar and Zegeye et al. complement this perspective by building a practical hardware testbed that enables deploying and exercising existing Matter applications end-to-end [13], [14], [15]. In security research, a growing body of work motivates the need for rapid prototyping and evaluation; Wang et al. identify vulnerabilities and attack scenarios that emerge from real smart-home Matter deployments and integrations [16]. These efforts motivate and enable experimentation, but they do not provide a reproducible workflow for implementing and invoking new experimental functionality within Matter. Our work fills this gap with an end-to-end template spanning both the device and a real controller.

V. Summary

We present instructions, and a template example, for adding a custom cluster in the reference implementation of Matter and invoking it from a real smart-home controller. Full instructions and code can be found at https://github.com/SPLICE-project/matter-custom-clusters.

Acknowledgements

This research results from the SPLICE research program, supported by a collaborative award from the SaTC Frontiers program at the National Science Foundation under award number CNS-1955805. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors. Any mention of specific companies or products does not imply any endorsement by the authors, by their employers, or by the sponsors.

References

[1] Connectivity Standards Alliance. (2023, Oct.) Matter 1.2 arrives with nine new device types & improvements across the board. https://csa-iot.org/newsroom/matter-1-2-arrives-with-nine-new-device-types-improvements-across-the-board/. Accessed Dec. 18, 2025.

[2] ——. Midea pioneering Matter adoption in smart appliances. Online at https://csa-iot.org/newsroom/midea-pioneering-matter-adoption-in-smart-appliances/.

[3] R. Mangar, J. Chandler, T. J. Pierson, and D. Kotz, "Enabling Research Extensions in Matter via Custom Clusters," Online at https://github.com/SPLICE-project/matter-custom-clusters.

[4] Google, "The device data model," https://developers.home.google.com/matter/primer/device-data-model, Apr. 2023, accessed Dec. 18, 2025.

[5] Home Assistant, "Home assistant," https://www.home-assistant.io/, accessed Dec. 18, 2025.

[6] Home Assistant Core, "Home Assistant Core release 2025.12.3," https://github.com/home-assistant/core/releases/tag/2025.12.3, Dec. 2025, release 2025.12.3 Accessed Dec. 18, 2025.

[7] Python Matter Server, "python-matter-server release 8.1.1," https://github.com/matter-js/python-matter-server, Sep. 2025, release 8.1.1. Accessed Dec. 18, 2025.

[8] Home Assistant Docs, "Matter," https://www.home-assistant.io/integrations/matter/, 2025, accessed Dec. 18, 2025.

[9] ——, "Integration file structure," https://developers.home-assistant.io/docs/creating_integration_file_structure/, accessed Dec. 18, 2025.

[10] ——, "Actions," https://www.home-assistant.io/dashboards/actions/, accessed Dec. 18, 2025.

[11] ——, "Button card," https://www.home-assistant.io/dashboards/button/, accessed Dec. 18, 2025.

[12] D. Belli, P. Barsocchi, and F. Palumbo, "Connectivity Standards Alliance Matter: State of the art and opportunities," *Internet of Things*, vol. 25, p. 101005, 2024.

[13] R. Mangar, J. Qian, W. Zegeye, M. Khanafer, A. AlRabah, B. Civjan, S. Sundram, S. Yuan, C. Gunter, K. Kornegay, T. J. Pierson, and D. Kotz, "Designing and Evaluating a Testbed for the Matter Protocol: Insights into User Experience," in *Proceedings of the NDSS Workshop on Security and Privacy in Standardized IoT (SDIoTSec)*. NDSS, February 2024.

[14] W. Zegeye, A. Jemal, and K. Kornegay, "Connected smart home over Matter protocol," in *Proceedings of the International Conference on Consumer Electronics (ICCE)*. IEEE, 2023, pp. 1–7.

[15] W. K. Zegeye, R. Mangar, J. Qian, V. Morris, M. Khanafer, K. Kornegay, T. J. Pierson, and D. Kotz, "Comparing smart-home devices that use the Matter protocol," in *Proceedings of the International Workshop on Intelligent Communication Network Technologies (ICNET'25)*. IEEE, January 2025, DOI 10.1109/CCNC54725.2025.10976049.

[16] H. Wang, Y. Liu, Y. Fang, Z. Jin, Q. Liu, and L. Xing, "WIP: Security vulnerabilities and attack scenarios in smart home with Matter," in *Proceedings of the NDSS Workshop on Security and Privacy in Standardized IoT (SDIoTSec)*. NDSS, February 2024, DOI 10.14722/sdiotsec.2024.23048.