Microsoft Azure

Azure IoT Academy

Transforming your business

Month 2, Day 2: Azure Digital Twins

Rebekah Midkiff
Technical Specialist
Microsoft

Alan Blythe
Sr. Technical Specialist
Microsoft

Eric Johnston
Sr. Technical Specialist
Microsoft

IoT Academy Journey

Month 2

- IoT Edge
- EFLOW
- Grafana
- Azure Monitor
- Azure Logic App
- Azure Digital Twins
- Azure Functions
- Partner Showcase

Month 3

- IoT Security
- Azure Sentinel
- Defender
- Partner Showcase
- Awards Ceremony

Day One, In Review

- Azure Bicep
- Azure IoT Edge
  - Debugging
  - Modules
  - Containerization
- Grafana
- Azure Logic App
- Alerts with Azure Monitor

IoT Academy Expectations

- *We have a very large audience, so please keep yourself on mute except when called.*
- *Please raise your hand and wait for acknowledgement before unmuting to ask a question.*
- *Use Teams reactions to ease interactions of a large audience*
- *We want this to be interactive so please don't hesitate to let us know if you have a question (comment in chat or raise hand).*
- *If you're stuck on a hands-on lab, we request that you notify us in chat and raise your hand so we can move you to a breakout meeting for assistance.*

Day Two Agenda (All times are in ET)

- 10:05am – 10:20am Introduction/Expectations Kickoff - Team
- 10:20am- 11:00pm Presentation
- 11:00pm – 11:15pm Coffee Break
- 11:15pm – 12:00pm HOL 1
- 12:00pm - 12:45pm Lunch Break
- 12:45pm - 1:15pm HOL 2/3
- 1:15pm – 3:00pm HOL 3
- 3:00pm - 3:15pm Coffee Break (Flexible timing)
- 3:15pm - 4:00pm HOL 3/Close/Q&A

Please fill out the bring your workload form ☺

Bring your own workload: collaboration session

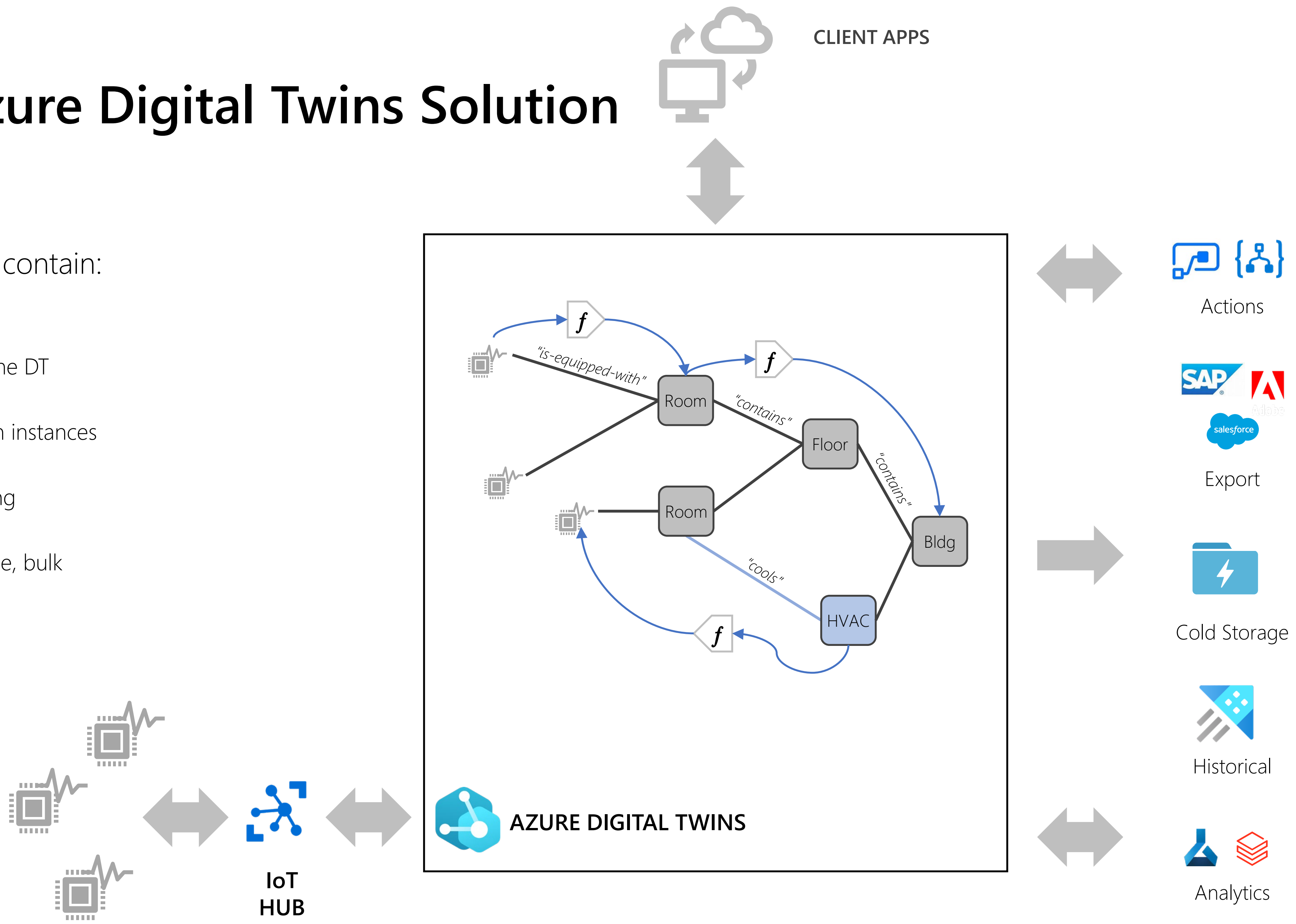
This will help us collaborate with each of you and give us insight in what areas you need assistance.

Azure Digital Twins Technical Overview

Anatomy of an Azure Digital Twins Solution

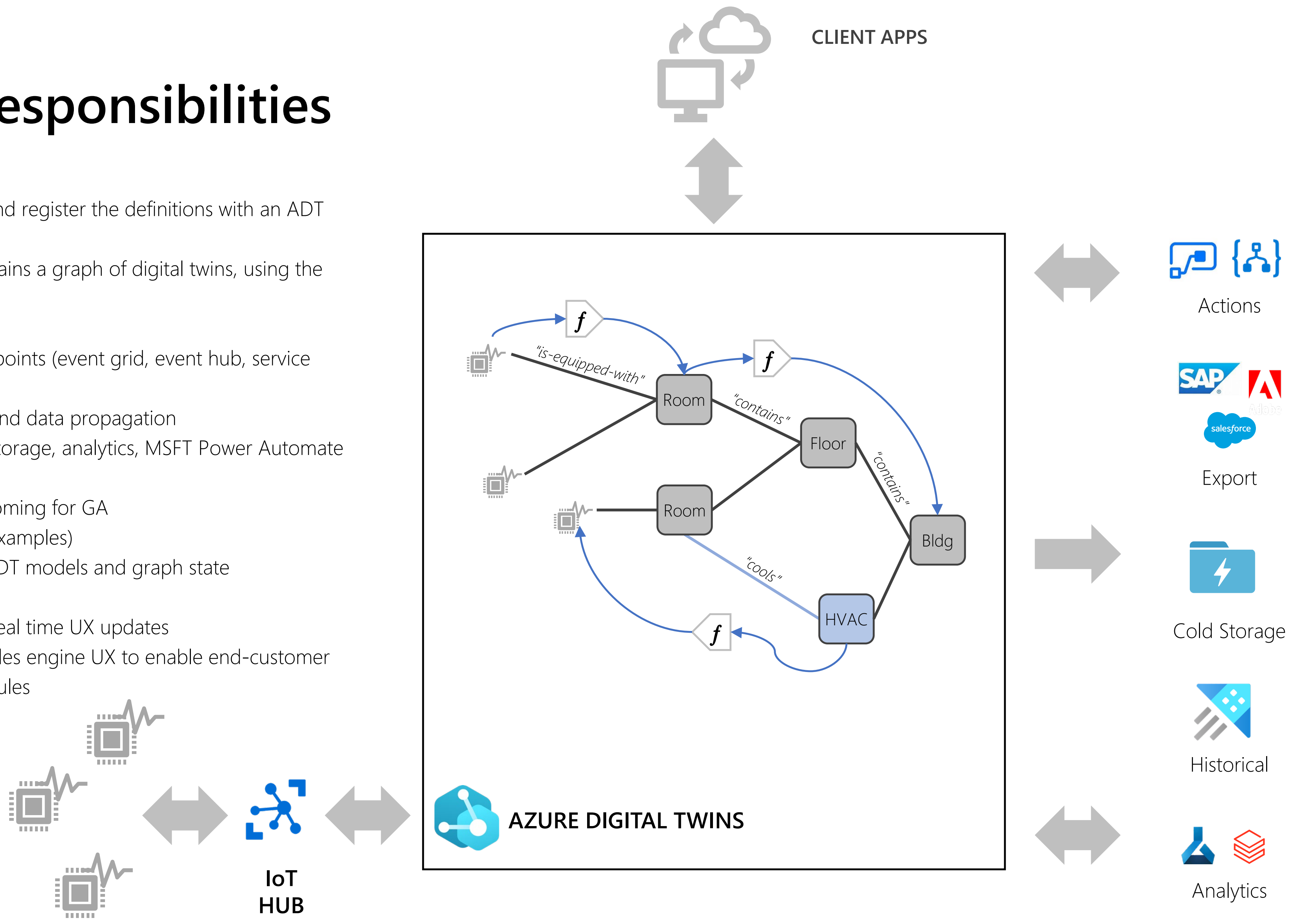
Digital Twins solution will typically contain:

- A Digital Twins instance
- IoT Hub with connected devices
- One or more client apps that connect to the DT instance:
  - Create and manage a graph of twin instances
  - Extract insight from twin state
- A set of Azure functions or other processing resources for custom processing
- Downstream services for long-term storage, bulk data analytics, etc.



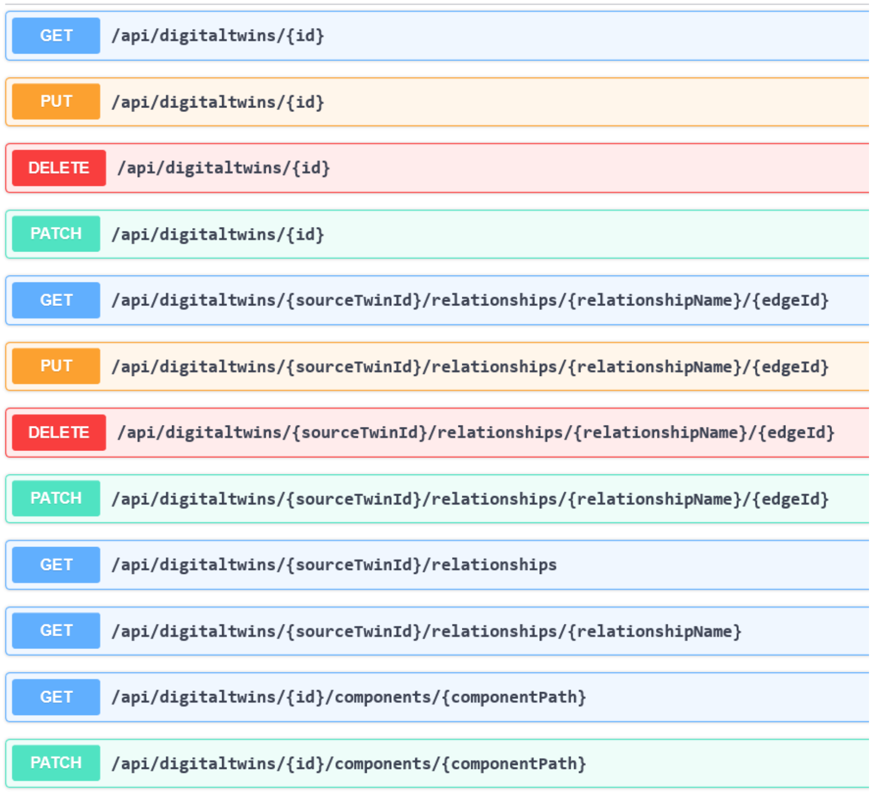
Core Developer Responsibilities

- Model the environment:
    - Create twin definitions using DTDL and register the definitions with an ADT instance
    - Write code that generates and maintains a graph of digital twins, using the previously defined models
- Process data and update state
    - Create egress routes to external endpoints (event grid, event hub, service bus)
    - Apply processing for business logic and data propagation
    - Send data to downstream services (storage, analytics, MSFT Power Automate etc.)
    - Additional event handling features coming for GA
- Write solution-specific frontend / UX code (examples)
    - Build monitoring dashboards from ADT models and graph state
    - Create visualizations of the graph
    - Wire up UX event handlers to drive real time UX updates
    - Build domain-specific query UX or rules engine UX to enable end-customer defined, domain-specific queries or rules
    - And many more…



Leveraging the Azure Digital Twin Service

- PaaS Service Offering
- Setup via Azure Portal, comfortable Azure CLI support
- Programmed via REST API surface
- SDKs available for C# at Public Preview
  - Additional SDKs for supported Azure languages will follow
  - For Public Preview, additional SDKs can be generated by customers using Autorest



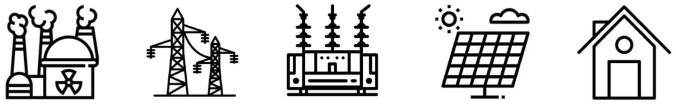
GET	/api/digitaltwins/{id}
PUT	/api/digitaltwins/{id}
DELETE	/api/digitaltwins/{id}
PATCH	/api/digitaltwins/{id}
GET	/api/digitaltwins/{sourceTwinId}/relationships/{relationshipName}/{edgeId}
PUT	/api/digitaltwins/{sourceTwinId}/relationships/{relationshipName}/{edgeId}
DELETE	/api/digitaltwins/{sourceTwinId}/relationships/{relationshipName}/{edgeId}
PATCH	/api/digitaltwins/{sourceTwinId}/relationships/{relationshipName}/{edgeId}
GET	/api/digitaltwins/{sourceTwinId}/relationships
GET	/api/digitaltwins/{sourceTwinId}/relationships/{relationshipName}
GET	/api/digitaltwins/{id}/components/{componentPath}
PATCH	/api/digitaltwins/{id}/components/{componentPath}

Digital Twins Modeling

Modeling with Azure Digital Twins

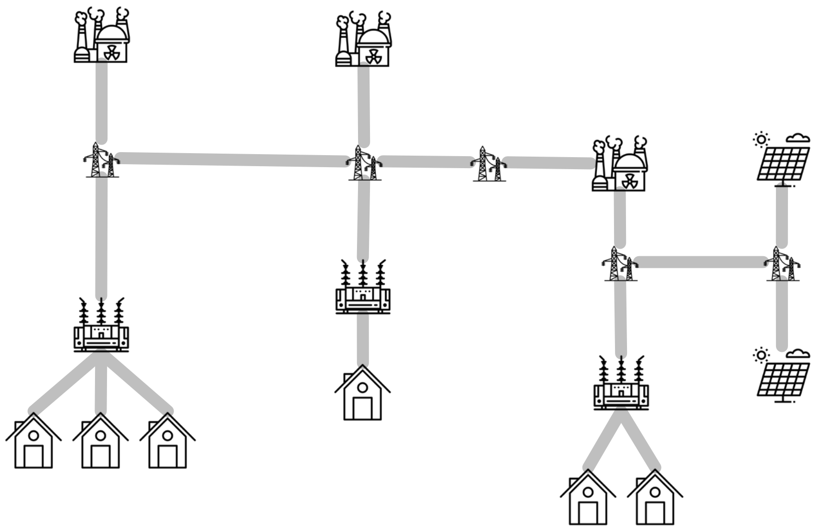
Create a domain vocabulary

- Describe the entities and concepts important for your business
- Describe how entities relate and connect to each other
- Use the Digital Twins Definition Language (DTDL) to author entities
    - Open-source specification
    - Programming language independent
    - Based on JSON-LD
- DTDL is also used to describe IoT devices
    - Aligned with IoT Plug and Play and Azure Data Explorer data model
    - Enables Plug and Play connectivity for device
    - Consistent programming model from ADT in the cloud to devices



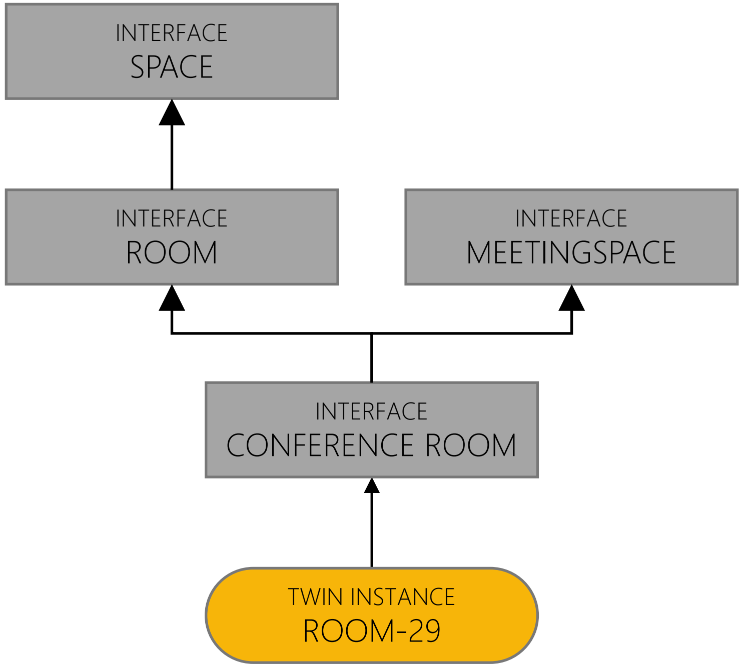
Build a model of your environment

- Create instances of the specific entities in your real world
- Connect the instances into a topology graph that represents relationships in your environment
- Define event processing and routing for your environment



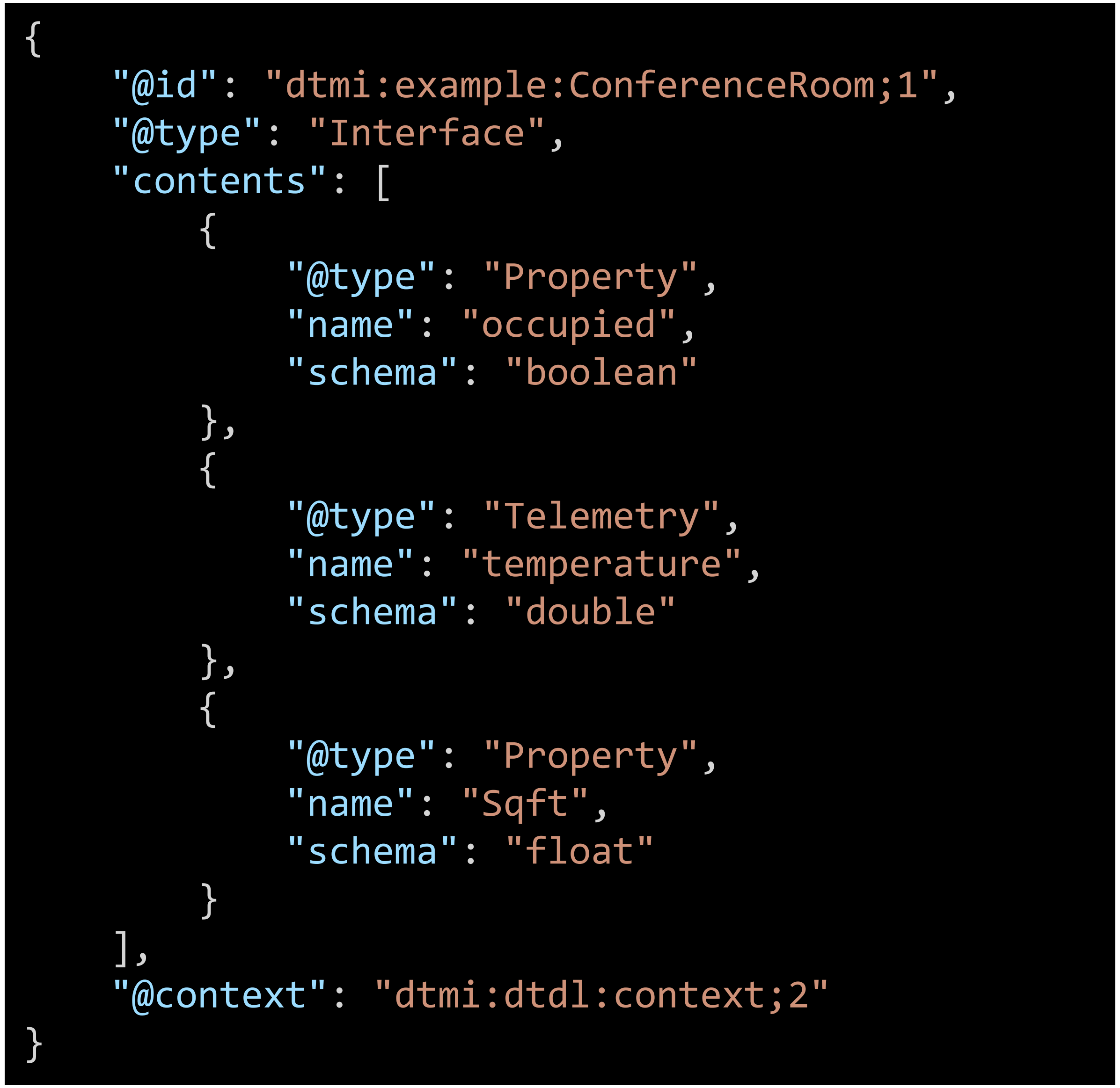
Twins Modeling

- With DTDL, you can describe your TWIN in terms of:
  - Properties
    - Properties are data fields that represent some state of a digital twin
    - Use properties to represent durable state
    - Read-only or writable
  - Telemetry
    - Telemetry fields represent measurements
    - Measurement are typically used for the equivalent of sensor readings.
  - Commands
    - Commands represent methods can be executed on the digital twin
    - Example: Reset command, or a command to switch a fan on or off.
  - Relationships
    - Relationships lets you model how a given twin is related to other twins
    - Relationships can represent different semantic meanings, e.g. “floor contains room”, “hvac cools rooms”, “Compressor is-billed-to user” etc.
  - Components
    - Component lets you build your model from other interfaces. Use a component to describe something that **is a part** of your model. Use a relationship to describe something that **relates** to your model.
    - E.g. a phone that is made up of a front camera and a back camera. Phone has an interface which has two components: front camera and back camera.



DTDL supports inheritance

Simple Interface Example



```
{
    "@id": "dtmi:example:ConferenceRoom;1",
    "@type": "Interface",
    "contents": [
        {
            "@type": "Property",
            "name": "occupied",
            "schema": "boolean"
        },
        {
            "@type": "Telemetry",
            "name": "temperature",
            "schema": "double"
        },
        {
            "@type": "Property",
            "name": "Sqft",
            "schema": "float"
        }
    ],
    "@context": "dtmi:dtdl:context;2"
}
```

- The following example shows a simple room
- The top level of any twin description is called an **interface**.
- At the top level, the twin description has a number of necessary fields
    - @id
    - @type
    - contents
        - @type
        - name
        - schema
    - @context

Types

Properties and telemetry fields can use simple and complex types

```
"contents": [
    {
        "@type": "Telemetry",
        "name": "rotation",
        "schema": {
            "@type": "Object",
            "fields": [
                {
                    "name": "roll",
                    "schema": "double"
                },
                {
                    "name": "pitch",
                    "schema": "double"
                },
                {
                    "name": "yaw",
                    "schema": "double"
                }
            ]
        }
    }
]
```

- Schema attribute for properties, telemetry and command arguments defines data type
- Simple types are primitives:
  - Integer, Boolean, double, string, etc.
- Optional displayUnit attribute to indicate a unit for display
- Complex Types
  - A complex type holds an array of fields
  - Fields can be of simple or complex types themselves
  - Complex types can be defined inline or as re-usable types

Properties

Properties are used to describe state that can be read or optionally written at any time

```
"contents": [
    {
        "@type": "Property",
        "name": "serialNumber",
        "schema": "string"
    },
    {
        "@type": "Property",
        "name": "fanSpeed",
        "writable": true,
        "schema": "double"
    }
]
```

- Properties are defined by a name and a schema
- Properties can be of simple or complex types
- From a client's point of view, properties can be read-only or writeable

Telemetry

Telemetry is used to represent sensor data that is not stored as state

```
"contents": [
    {
        "@type": “Telemetry",
        "name": “temperature",
        "schema": “double"
    },
    {
        "@type": “Telemetry",
        "name": “oilPressure",
        "schema": "double"
    }
]
```

- Telemetry is defined by a name and a schema
- Telemetry can be of simple or complex types

Relationships

Relationships lets you model how a given twin is related to other twins.

```
{
    "@id": “dtmi:example:Floor;1",
    "@type": "Interface",
    "contents": [
        {
            "@type": "Relationship",
            "name": "contains",
            "target": “dtmi:example:Room;1"
        },
        {
            "@type": "Relationship",
            "name": “isAssociatedWith",
        },

        ...
    ],
    "@context": "dtmi:dtdl:context;2"
}
```

- Relationships are uni-directional
- Bi-directional relationships can be modeled as relationship pairs
- Relationships can be queried
- Relationship attributes:
    - Name: Identifies the meaning of the relationship
    - Target: the type of interface the relationship targets
- Interfaces may define many different relationships

Inheritance (Specialization)

A twin might need to be specialized for a given use case – This can be done via simple interface inheritance

1

```
{
    "@id": “dtmi:example:Room;1",
    "@type": "Interface",
    "contents": [
        {
            "@type": "Property",
            "name": "occupied",
            "schema": "boolean"
        }
    ],
    "@context": “dtmi:dtdl:context;2”}
```

- Interfaces can inherit from each other
- Interface inheritance works largely like in C#
  - Names inherited from multiple interfaces coalesce
- For example, we'd like to have conference rooms, focus rooms, and office spaces that all share common properties.

2

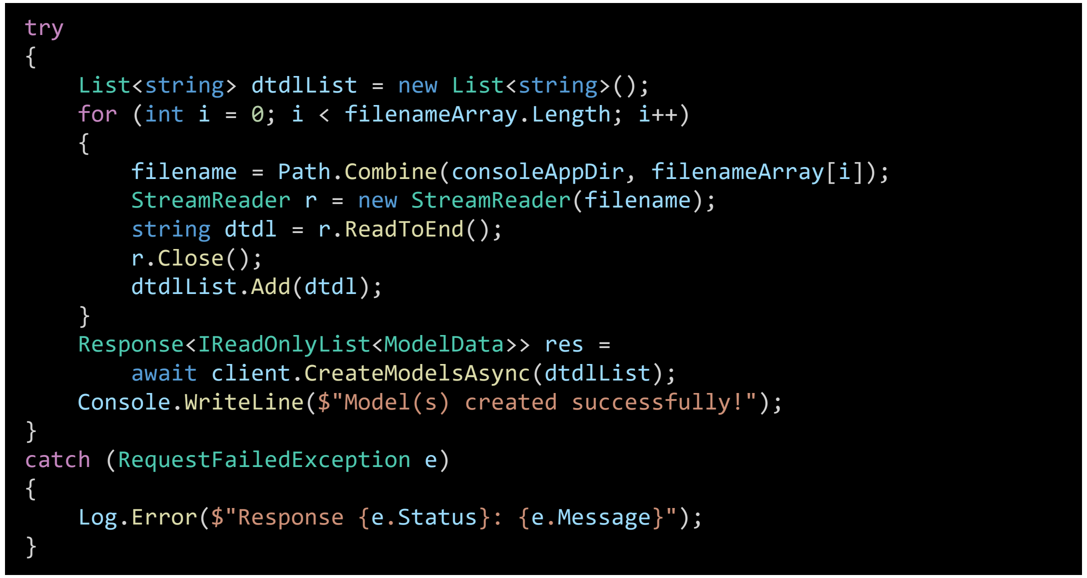
```
{
    "@id": “dtmi:example:ConferenceRoom;1",
    "@type": "Interface",
    "extends": “dtmi:example:Room;1",
    "contents": [
        {
            "@type": "Property",
            "name": "capacity",
            "schema": "integer"
        }
    ],
    "@context": “dtmi:dtdl:context;2”
}
```

3

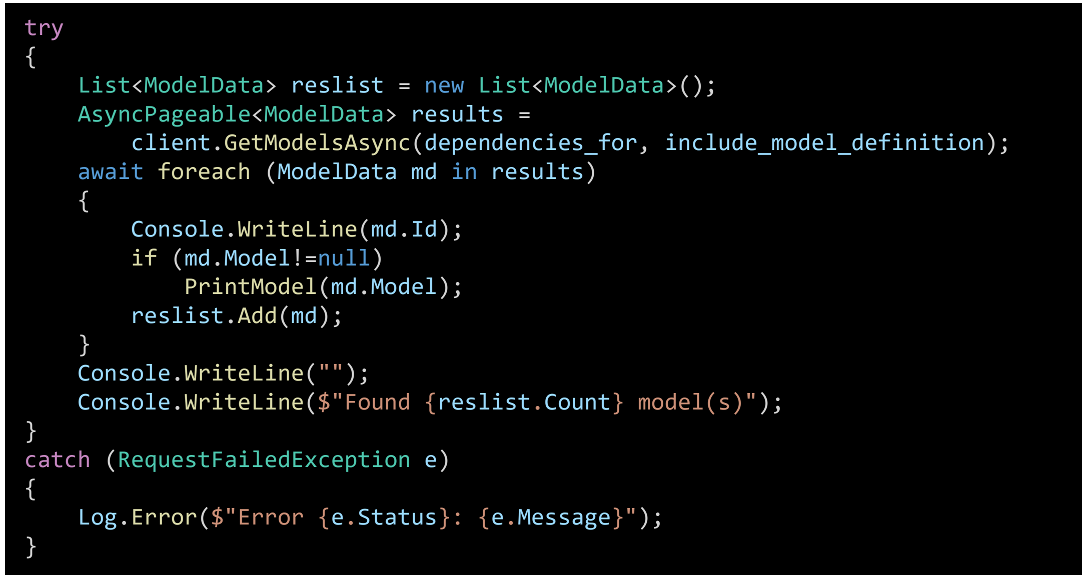
```
{
    "@id": “dtmi:example:ExecutiveConferenceRoom;1",
    "@type": "Interface",
    "extends": “dtmi:example:ConferenceRoom;1",
    "contents": [
        {
            "@type": "Property",
            "name": "vpName",
            "schema": "string"
        }
    ],
    "@context": “dtmi:dtdl:context;2”}
```

Model API Overview

- Create (Upload) models
- List models
- Decommission and delete models



```
try
{
    List<string> dtdlList = new List<string>();
    for (int i = 0; i < filenameArray.Length; i++)
    {
        filename = Path.Combine(consoleAppDir, filenameArray[i]);
        StreamReader r = new StreamReader(filename);
        string dtdl = r.ReadToEnd();
        r.Close();
        dtdlList.Add(dtdl);
    }
    Response<IReadOnlyList<ModelData>> res =
        await client.CreateModelsAsync(dtdlList);
    Console.WriteLine($"Model(s) created successfully!");
}
catch (RequestFailedException e)
{
    Log.Error($"Response {e.Status}: {e.Message}");
}
```



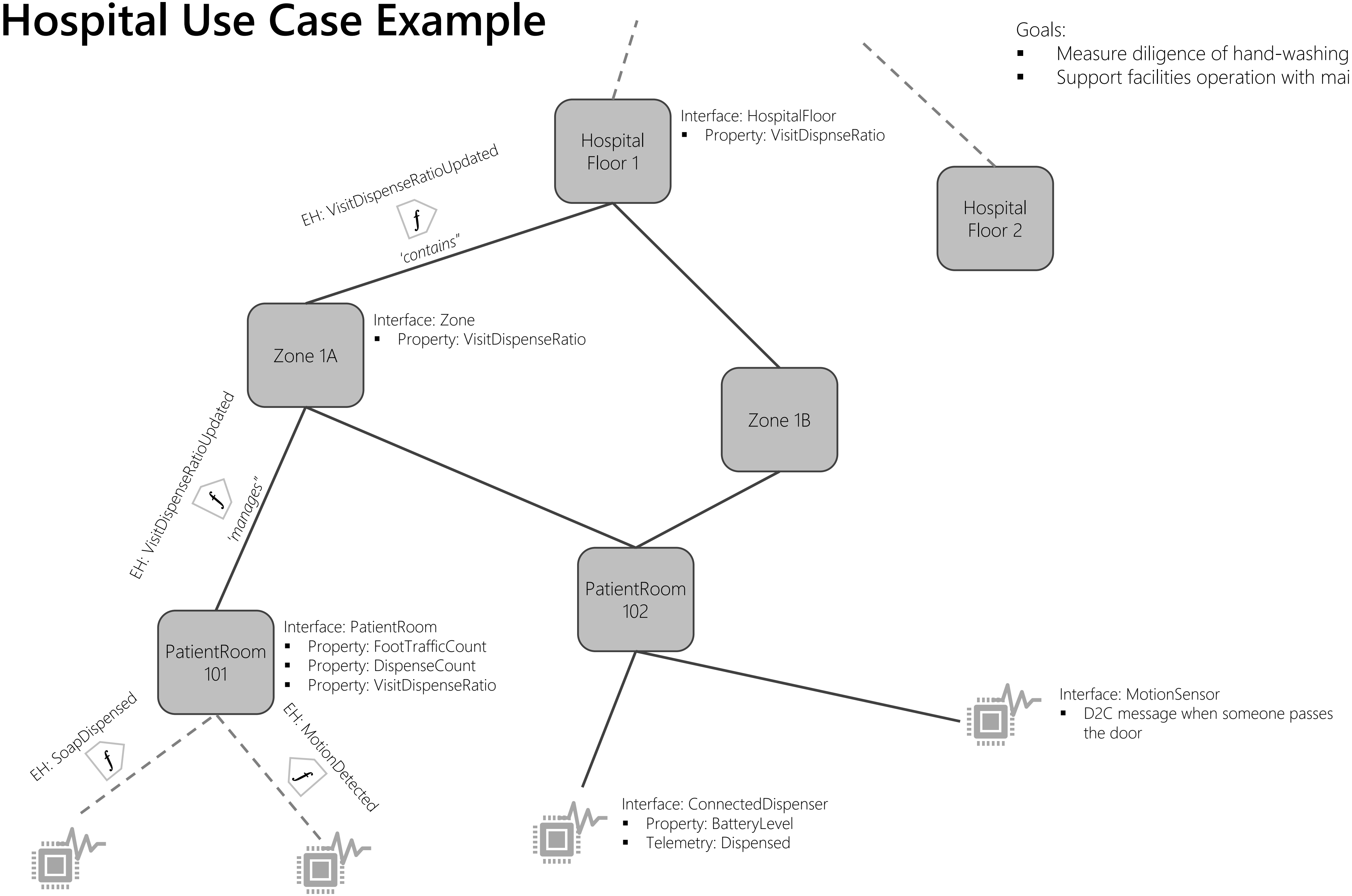
```
try
{
    List<ModelData> reslist = new List<ModelData>();
    AsyncPageable<ModelData> results =
        client.GetModelsAsync(dependencies_for, include_model_definition);
    await foreach (ModelData md in results)
    {
        Console.WriteLine(md.Id);
        if (md.Model!=null)
            PrintModel(md.Model);
        reslist.Add(md);
    }
    Console.WriteLine("");
    Console.WriteLine($"Found {reslist.Count} model(s)");
}
catch (RequestFailedException e)
{
    Log.Error($"Error {e.Status}: {e.Message}");
}
```

Digital Twins Graph

Hospital Use Case Example

Goals:

- Measure diligence of hand-washing on a hospital floor
- Support facilities operation with maintenance information



Hospital Floor 1

Interface: HospitalFloor

- Property: VisitDispnseRatio

Hospital Floor 2

EH: VisitDispenseRatioUpdated

'contains"

Zone 1A

Interface: Zone

- Property: VisitDispenseRatio

Zone 1B

EH: VisitDispenseRatioUpdated

'manages"

PatientRoom 101

Interface: PatientRoom

- Property: FootTrafficCount
- Property: DispenseCount
- Property: VisitDispenseRatio

PatientRoom 102

EH: SoapDispensed

EH: MotionDetected

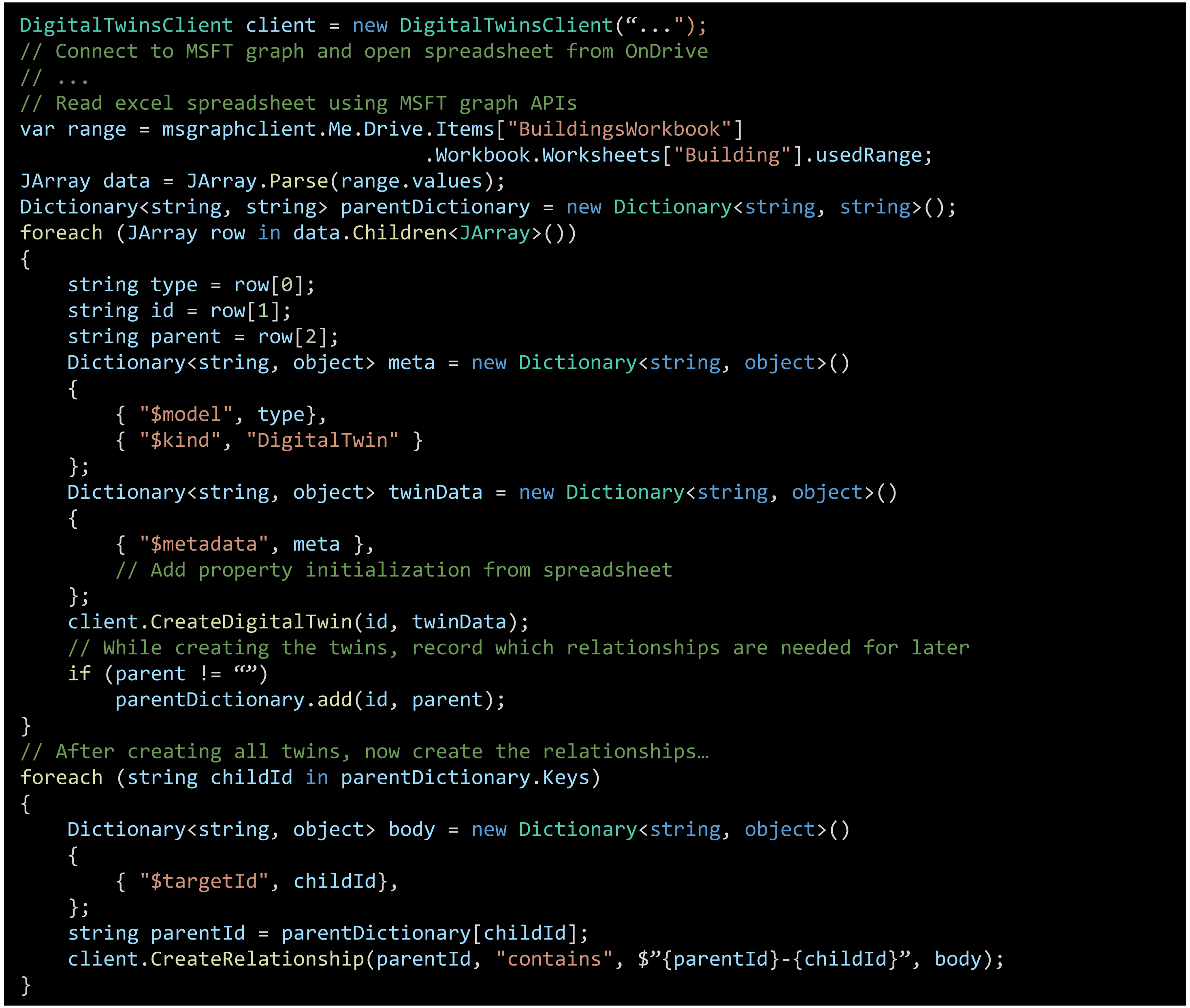
Interface: MotionSensor

- D2C message when someone passes the door

Interface: ConnectedDispenser

- Property: BatteryLevel
- Telemetry: Dispensed

Building Topology Sample



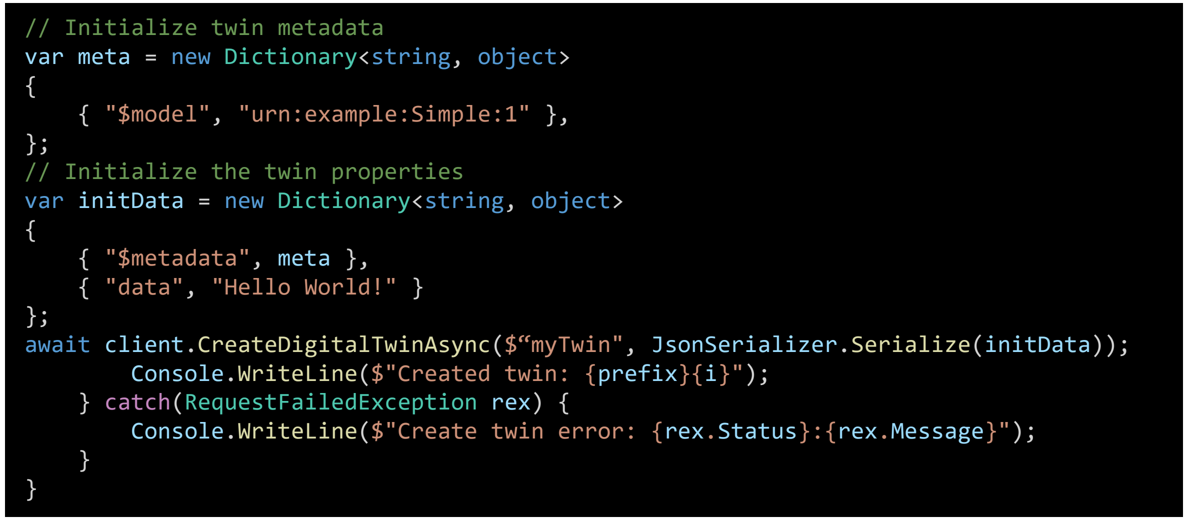
```
DigitalTwinsClient client = new DigitalTwinsClient(“...");
// Connect to MSFT graph and open spreadsheet from OnDrive
// ...
// Read excel spreadsheet using MSFT graph APIs
var range = msgraphclient.Me.Drive.Items["BuildingsWorkbook"]
                                  .Workbook.Worksheets["Building"].usedRange;
JArray data = JArray.Parse(range.values);
Dictionary<string, string> parentDictionary = new Dictionary<string, string>();
foreach (JArray row in data.Children<JArray>())
{
    string type = row[0];
    string id = row[1];
    string parent = row[2];
    Dictionary<string, object> meta = new Dictionary<string, object>()
    {
        { "$model", type},
        { "$kind", "DigitalTwin" }
    };
    Dictionary<string, object> twinData = new Dictionary<string, object>()
    {
        { "$metadata", meta },
        // Add property initialization from spreadsheet
    };
    client.CreateDigitalTwin(id, twinData);
    // While creating the twins, record which relationships are needed for later
    if (parent != “”)
        parentDictionary.add(id, parent);
}
// After creating all twins, now create the relationships…
foreach (string childId in parentDictionary.Keys)
{
    Dictionary<string, object> body = new Dictionary<string, object>()
    {
        { "$targetId", childId},
    };
    string parentId = parentDictionary[childId];
    client.CreateRelationship(parentId, "contains", $”{parentId}-{childId}”, body);
}
```

- The example code to the right builds a topology from data in an excel file (example below)
- This is a common use case for customers

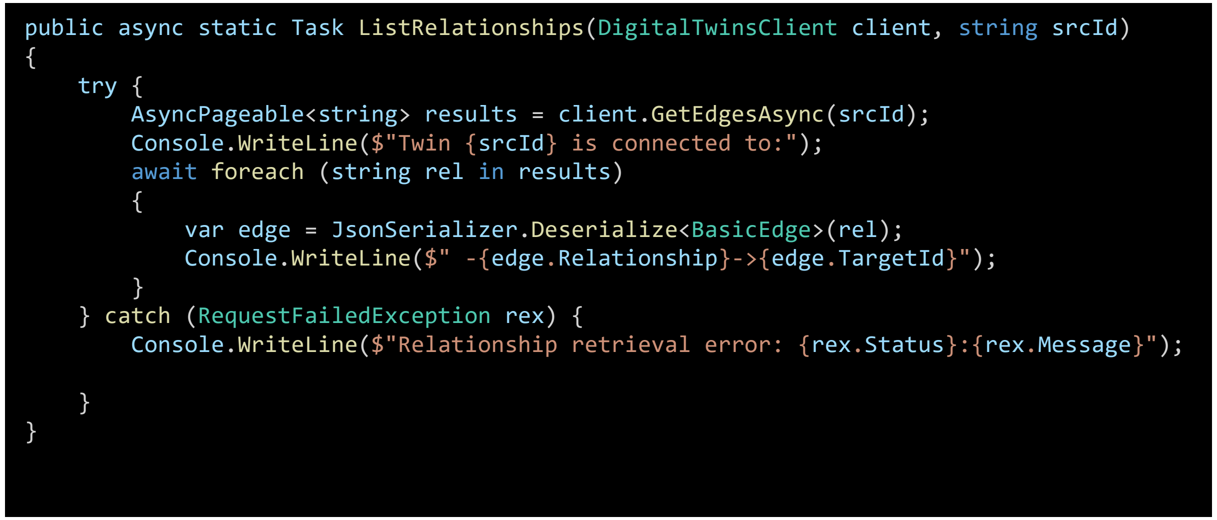
Type	Id	Parent	OtherData	OtherData	
floor	Floor01		…	…	
room	Room10	Floor01	…	…	
room	Room11	Floor01	…	…	
room	Room12	Floor01	…	…	
floor	Floor02		…	…	
room	Room21	Floor02	…	…	
room	Room22	Floor02	…	…	

Twin API Overview

- Twins
  - Create, Read, Patch/Update, Delete
- Relationships
  - Create, Read, Patch/Update, Delete
  - List incoming and outgoing relationships on twins



```
// Initialize twin metadata
var meta = new Dictionary<string, object>
{
    { "$model", "urn:example:Simple:1" },
};
// Initialize the twin properties
var initData = new Dictionary<string, object>
{
    { "$metadata", meta },
    { "data", "Hello World!" }
};
await client.CreateDigitalTwinAsync($“myTwin", JsonSerializer.Serialize(initData));
        Console.WriteLine($"Created twin: {prefix}{i}");
    } catch(RequestFailedException rex) {
        Console.WriteLine($"Create twin error: {rex.Status}:{rex.Message}");
    }
}
```



```
public async static Task ListRelationships(DigitalTwinsClient client, string srcId)
{
    try {
        AsyncPageable<string> results = client.GetEdgesAsync(srcId);
        Console.WriteLine($"Twin {srcId} is connected to:");
        await foreach (string rel in results)
        {
            var edge = JsonSerializer.Deserialize<BasicEdge>(rel);
            Console.WriteLine($" -{edge.Relationship}->{edge.TargetId}");
        }
    } catch (RequestFailedException rex) {
        Console.WriteLine($"Relationship retrieval error: {rex.Status}:{rex.Message}");

    }
}
```

Digital Twins Ingress

Ingress

- Drive ADT using the REST API surface
- Any source that can call REST APIs can drive ADT
- Typical scenario:
  - Event handler function attached to IoT Hub
  - Set properties on twins in response to telemetry messages
- Can be used with sources such as Event Hubs or Event Grids
- Can be used with Logic Apps and other tools that call REST APIs

Programming Example: Ingest Telemetry

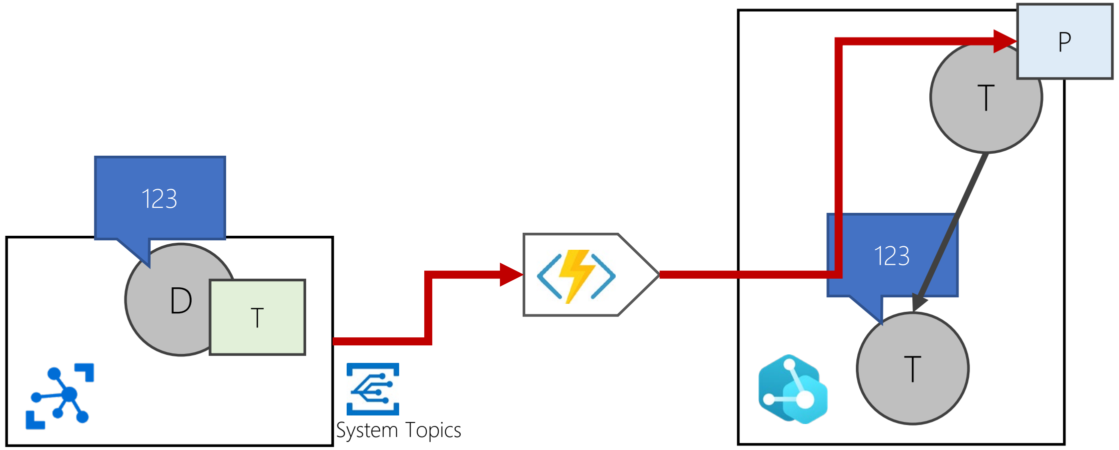
Use case: Set a property on a "logical" twin room based on telemetry from a connected device
Example: Set room temperature in response to thermostat telemetry
Not shown: Common items such as function creation, function security setup, etc

Graph Creation

```
Authenticate();

// Create init data with type and initial prop values;
result = await client.CreateTwinAsync(“MyRoom”, initData);
result = await client.CreateTwinAsync(“Thermostat-123”, initData);
result = await client.AddEdgeAsync(“MyRoom”, “Thermostat-123, Guid.NewGuid().ToString(), “contains”);
```

Azure Function

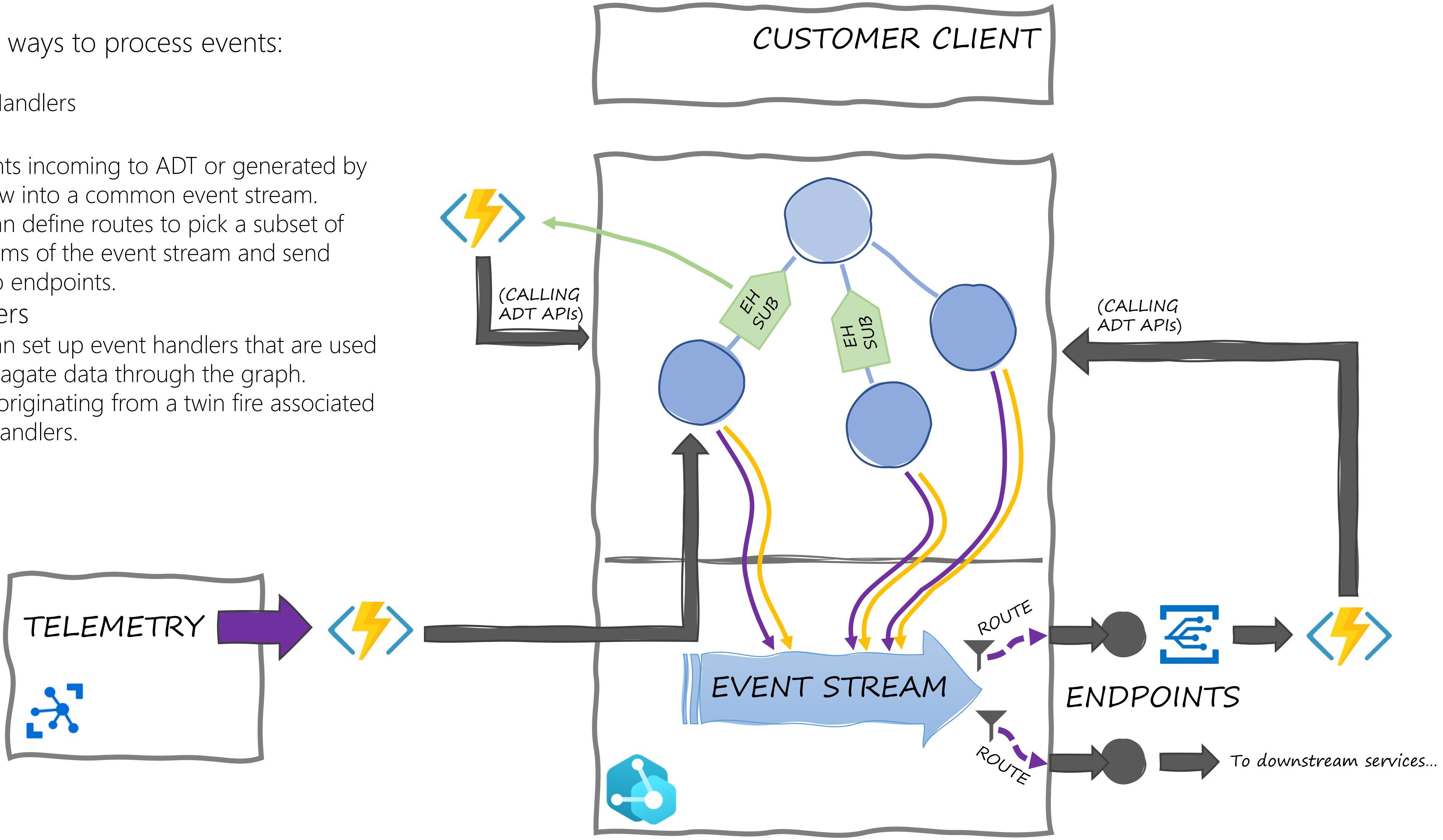
```
[FunctionName("ProcessHubToDTEvents")]
public async void Run([EventGridTrigger]EventGridEvent eventGridEvent, ILogger log)
{
  ManagedIdentityCredential cred = new ManagedIdentityCredential(adtAppId);
  client = new DigitalTwinsClient(new Uri(adtInstanceUrl), cred);

  if (client != null
  {
    if (eventGridEvent != null && eventGridEvent.Data != null)
    {
      // Reading deviceId from message headers
      JObject job = (JObject)JsonConvert.DeserializeObject(eventGridEvent.Data.ToString());
      string deviceId = (string)job["systemProperties"]["iothub-connection-device-id"];

      // Extracting temperature from device telemetry
      byte[] body = System.Convert.FromBase64String(job["body"].ToString());
      var value = System.Text.ASCIIEncoding.ASCII.GetString(body);
      var bodyProperty = (JObject)JsonConvert.DeserializeObject(value);
      var temp = bodyProperty["Temperature"];

      // Find parent using incoming relationships and update parent twin
      string parentId = await FindParent(deviceId, log);
      await AdtUtilities.UpdateTwinProperty(client, parentId, "/Temperature", temp, log);
    }
  }
}
```



Digital Twins Routes & Event Handling

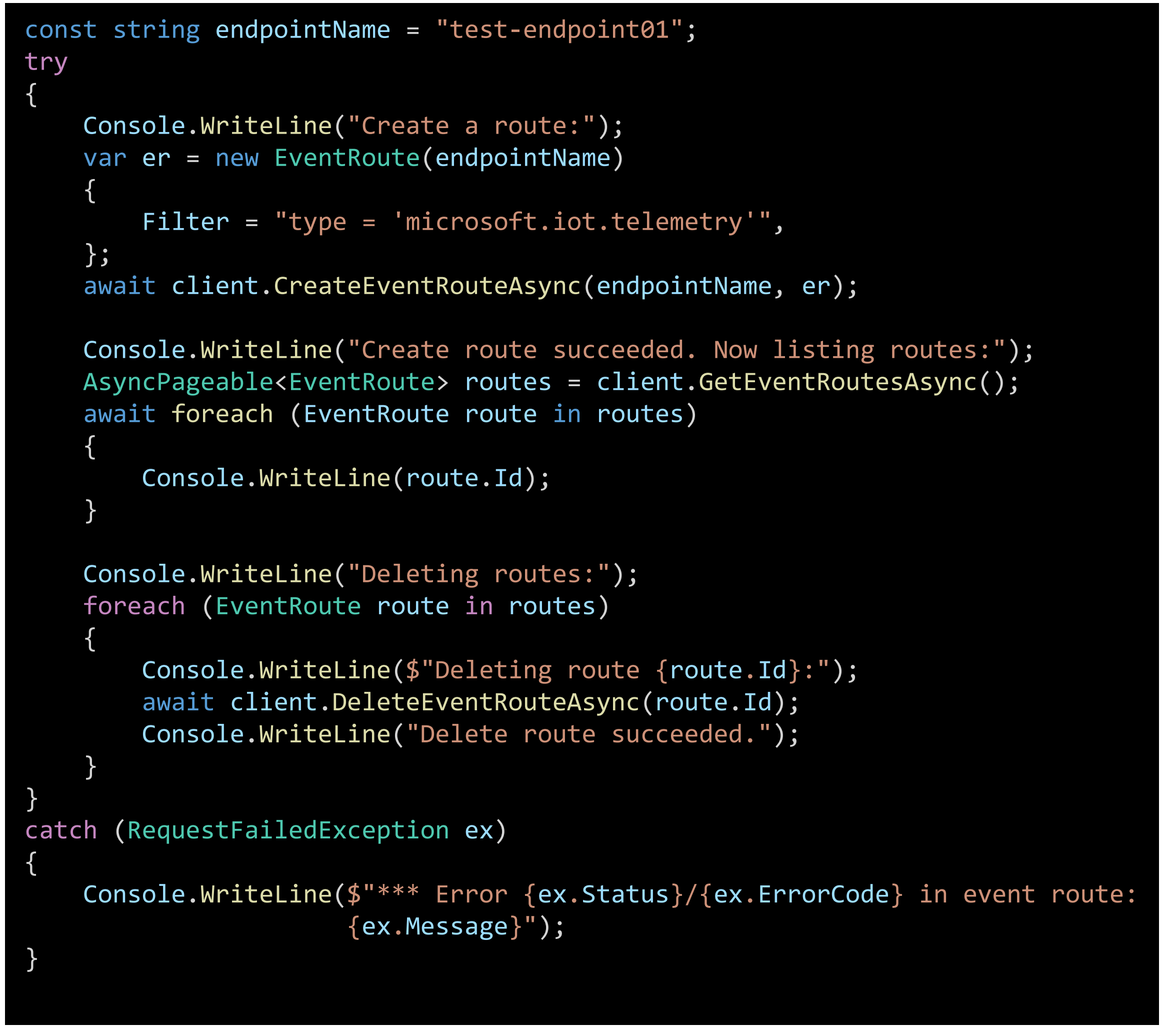
An Overview of Event Processing in ADT v2

- ADT has two ways to process events:
  - Routes
  - Event Handlers
- Routes
  - All events incoming to ADT or generated by ADT flow into a common event stream.
  - Devs can define routes to pick a subset of data items of the event stream and send them to endpoints.
- Event Handlers
  - Devs can set up event handlers that are used to propagate data through the graph.
  - Events originating from a twin fire associated event handlers.



Routes Example & Route Filtering

Creating, listing and deleting Routes



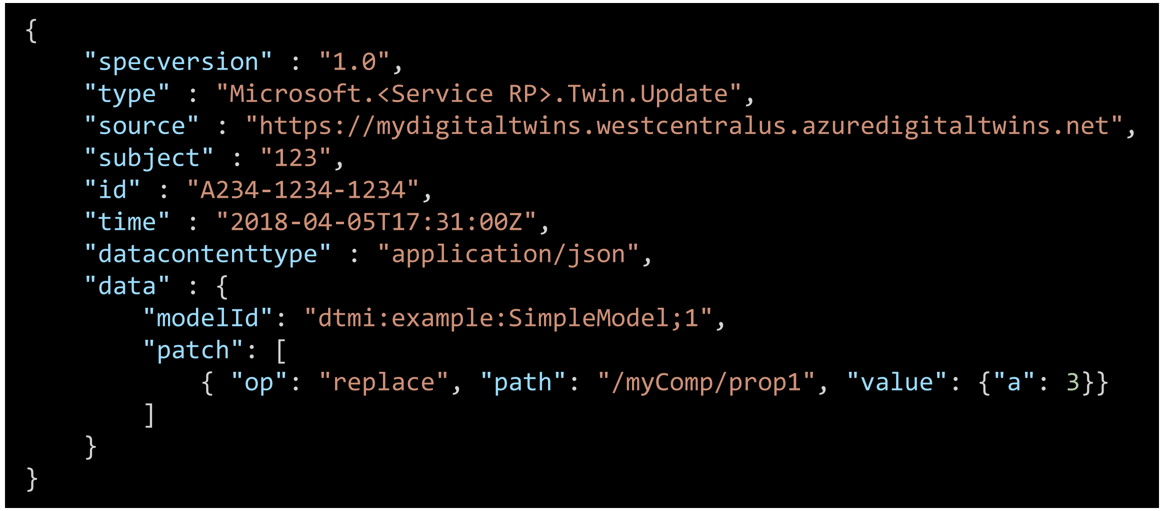
```
const string endpointName = "test-endpoint01";
try
{
    Console.WriteLine("Create a route:");
    var er = new EventRoute(endpointName)
    {
        Filter = "type = 'microsoft.iot.telemetry'",
    };
    await client.CreateEventRouteAsync(endpointName, er);

    Console.WriteLine("Create route succeeded. Now listing routes:");
    AsyncPageable<EventRoute> routes = client.GetEventRoutesAsync();
    await foreach (EventRoute route in routes)
    {
        Console.WriteLine(route.Id);
    }

    Console.WriteLine("Deleting routes:");
    foreach (EventRoute route in routes)
    {
        Console.WriteLine($"Deleting route {route.Id}:");
        await client.DeleteEventRouteAsync(route.Id);
        Console.WriteLine("Delete route succeeded.");
    }
}
catch (RequestFailedException ex)
{
    Console.WriteLine($"*** Error {ex.Status}/{ex.ErrorCode} in event route:
                      {ex.Message}");
}
```

Route Filtering

- ADT Messages use cloud event format
- Route filters today can filter against any combination of CE header fields

- Example of a filter to only include telemetry
  - `type = 'microsoft.iot.telemetry'`
- Example of a filter to only include one component
  - `Subject = 'thermostat67'`

Example Message (Twin Update)



```
{
    "specversion" : "1.0",
    "type" : "Microsoft.<Service RP>.Twin.Update",
    "source" : "https://mydigitaltwins.westcentralus.azuredigitaltwins.net",
    "subject" : "123",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "datacontenttype" : "application/json",
    "data" : {
        "modelId": "dtmi:example:SimpleModel;1",
        "patch": [
            { "op": "replace", "path": "/myComp/prop1", "value": {"a": 3}}
        ]
    }
}
```

Ingest and Event Processing Summary

Area	Public Preview	GA+
Ingest	**API-Driven**: Clients call ADT APIs to drive data into the service Example: Use EventHub or EventGrid on IoT Hub with an attached Azure Function	
Events	Use **routes** to send ADT events to downstream services and processing Use **EventGrid filtering** to select events for processing	Additional support for **twin-to-twin event handlers** (more efficient coding pattern for event propagation through the graph) Additional support for **simple event filtering in ADT routes**

Digital Twins Graph Queries

Public Preview Query Overview

- SQL-LIKE query language to search twins and relationships

- Get twins by properties
- Get twins by Model type
- Get twins by relationship properties
- Get twins using relationship traversal (limited to single hops for public preview)
- Any combination of above (AND, OR, NOT operator) of properties, interfaces, relationship properties and traversing
- Continuation support with variable page size in REST API. SDK provides continuous access (no need for explicit paging)
- Support for query comparison operators: AND/OR/NOT, IN/NOT IN, STARTSWITH/ENDSWITH, =, !=, <, >, <=, >=
- Scalar Functions support: IS_BOOL, IS_DEFINED, IS_NULL, IS_NUMBER, IS_OBJECT, IS_PRIMITIVE, IS_STRING, STARTS_WITH, ENDS_WITH

Query API Example

```
AsyncPageable<string> result = client.QueryAsync("Select * From DigitalTwins");
await foreach (string twin in result)
{
    Console.WriteLine(twin);
}
```

Queries Examples

Get twins by properties	Get twins by model	Get twins by traversing relationships

SELECT *
FROM DigitalTwins
WHERE $dtid in [‘123’, ‘456’]
AND firmwareVersion = ‘1.1’
	SELECT *
	FROM DigitalTwins
	WHERE IS_OF_MODEL
	(‘dtmi:contosocom:DigitalTwins:Space;3’)
	AND roomSize > 50
		SELECT device
		FROM DigitalTwins space
		JOIN device RELATED space.has
		WHERE space.$dtid = ‘Room 123’
		AND device.$metadata.model =
		‘dtmi:contosocom:DigitalTwins:MxChip:3’
		AND has.role = ‘Operator’

Additional Queries Examples

Description	Query

Get twins which property “Location” is defined	SELECT *
	FROM DIGITALTWINS WHERE IS_DEFINED(Location)

Get twins of this model and id which property Temperature is a numeric value	SELECT * FROM TWINS
	WHERE IS_OF_MODEL(‘dtmi:contosocom:DigitalTwins:Space;10')
	AND IS_NUMBER(T.Temperature)

Get twins which have a relationship named “Contains” with another twin with id = ‘id1’	SELECT Room
	FROM DIGITIALTWINS Room
	JOIN Thermostat ON Room.Contains
	WHERE Thermostat.$dtId = ‘id1'

Get rooms with this model which are on this floor and the floor has a “Contains” relationship with the room	SELECT Room
	FROM DIGITALTWINS Floor
	JOIN Room RELATED Floor.Contains
	WHERE Floor.$dtId = 'floor11'
	AND IS_OF_MODEL(Room, ’dtmi:contosocom:DigitalTwins:Room;1')

Digital Twins Access Control

Access Control

- Role Based Access Control support in ADT aligned with Azure RBAC
  - Coarse-grain access control with two built-in roles for authorizing access to ADT resources
    - Azure Digital Twins Owner
    - Azure Digital Twins Reader
  - Support for custom roles to meet specific access control needs
    - Customize roles by allowing actions (CRUD) from different permissions scopes

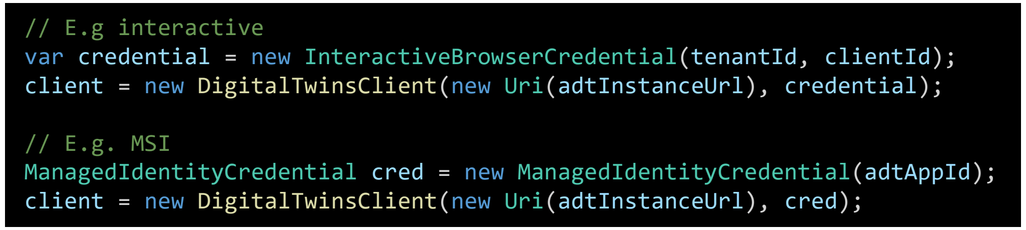
- Permission scope for control over:
  - **Model** **C**reate **R**ead **U**pdate **D**elete
  - **Digital Twins** CRUD
  - **Digital Twins Relationships** CRUD
  - **Query** operations
  - **Event routes** CRUD
    - Capability used for directing events to an endpoint
    - E.g. Event Hub, Event Grid, or Service Bus.

SDK and CLI

SDK Details

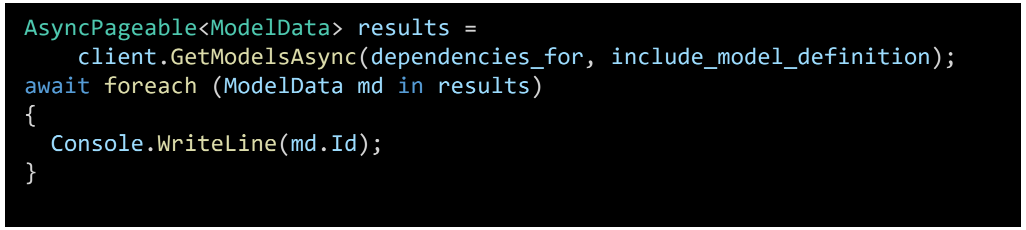
- Based on Azure SDK guidelines
  - General: https://azure.github.io/azure-sdk/general_introduction.html
  - .NET: https://azure.github.io/azure-sdk/dotnet_introduction.html
- Initial support for C#, additional SDKs for supported Azure languages will follow
- Idiomatic support for each language
- Comfortable authentication
- Consistent return types and error handling
- Automatic paging

Comfortable authentication



```
// E.g interactive
var credential = new InteractiveBrowserCredential(tenantId, clientId);
client = new DigitalTwinsClient(new Uri(adtInstanceUrl), credential);

// E.g. MSI
ManagedIdentityCredential cred = new ManagedIdentityCredential(adtAppId);
client = new DigitalTwinsClient(new Uri(adtInstanceUrl), cred);
```

Automatic Paging



```
AsyncPageable<ModelData> results =
    client.GetModelsAsync(dependencies_for, include_model_definition);
await foreach (ModelData md in results)
{
  Console.WriteLine(md.Id);
}
```

DTDL Parser

- Parses and validates DTDL
- Provides a C# object model for DTDL models (“reflection” over DTDL)
- Handles DTDL model sets (e.g. inheritance, etc.)
- Can be used by client software:
  - To validate models before service update
  - To find out which properties or methods are defined in a model (taking into account inheritance)
  - To drive UX generation code for model-driven UX
- C# code library distributed via NuGet

Azure CLI with Extensions for ADT

- Comfortable CLI commands to create and manage ADT instances
- Create, list and delete instances
- Manage RBAC
- Manage Endpoints
- Supports simple data plane commands operations too

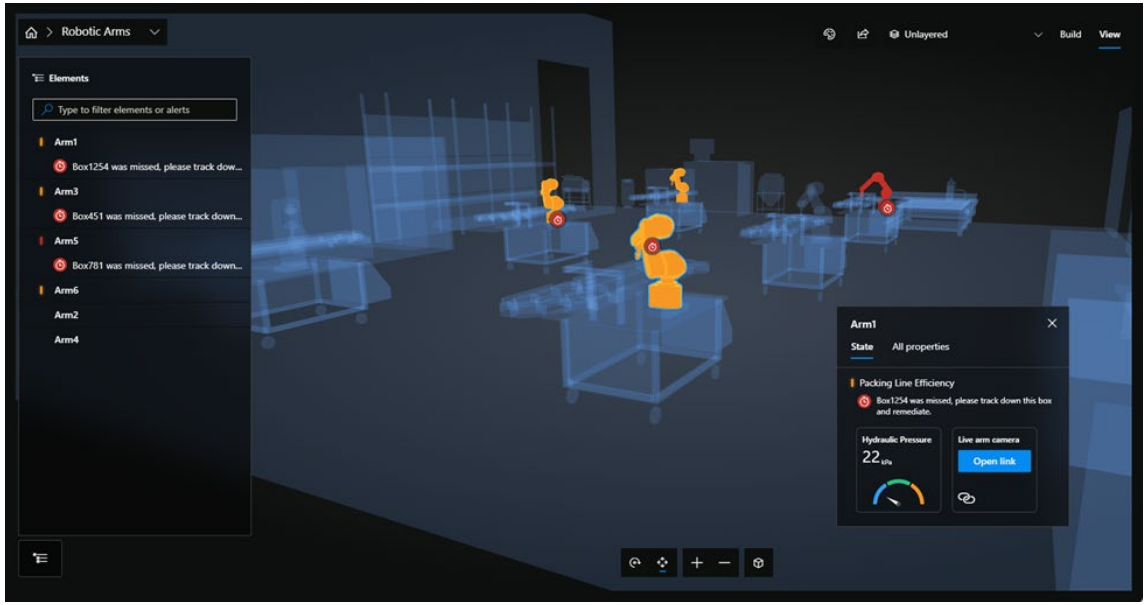
Example: AZ CLI commands to set up an ADT instance first time

```
az login
az account set -s <your-approved-subscription-ID>
az configure --defaults location="West Central US"
az provider register --namespace 'Microsoft.DigitalTwins'
az group create -n <your-resource-group-name>
az dt create --dt-name <name-for-your-Azure-Digital-Twins-instance>
             -g <your-resource-group-name>
az dt rbac assign-role -n <your-instance-name> --role owner
           -g <your-resource-group> --assignee <service-principal-to-grant-access>
```

3D Scenes Studio

3D Scenes Studio

- A tool for building 3D representations of entities and environments modeled with Azure Digital Twins.
- Provides a live or simulated visualization.
- Allows business stakeholders to better understand operations.
- Article
- IoT Show Video



Arm1

Box1254 was missed, please track dow...

Arm3

Box451 was missed, please track down...

Arm5

Box781 was missed, please track down...

Arm6

Arm2

Arm4

Arm1

Packing Line Efficiency

Box1254 was missed, please track down this box and remediate.

Hydraulic Pressure 22 kPa

Live arm camera

Hands-On Labs

Deploying Azure Digital Twins

The evolution of cloud + edge experiences

HORIZON 1 - TODAY

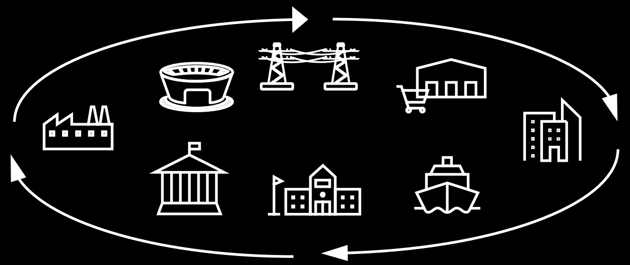
Intelligent Assets & Products

HORIZON 2 - EMERGING

Intelligent Environments

HORIZON 3 - FUTURE

Intelligent Ecosystems



What did we learn?

- Azure Digital Twins
- Azure Digital Explorer
- Azure Digital Twins SDK
- Azure Digital Twins C# Sample Project
  - Processed simulated telemetry from a IoT Hub device
    - Using an Azure Digital Twins Graph
    - Event Grid
    - Azure Functions

Send questions to iotacademy@microsoft.com
We start tomorrow at 10:10am ET