EFM Manager User Guide

Kinetic - Edge & Fog Processing Module (EFM) 1.7.0

Revised: July 9, 2019

© 2018 Cisco and/or its affiliates. All rights reserved.

Introduction

The EFM Manager is a component of the of Cisco Kinetic EFM that can detect and manages devices and message brokers throughout the EFM message system. In addition to managing the devices and brokers, the user has the ability to create dashboards for health monitoring of selected fields that can be presented in the most current value or a historical chart.

The discovery and onboarding of devices are accomplished through a set of definition files that specific how the devices are discovered in the EFM node system, which fields and values will be used, as well what the resulting output will be desired. Device fields are input from the original device node or through the EFM Manager UI, enabling the user to compliment detected values with user input. The discovery and onboarding of the DSA message brokers does not require user configuration.

The EFM Manager is extensible and each device type has its own unique set of definitions. These definitions files can be upgraded to support new fields or renaming as requirements change.

After discovery and a device is approved, the EFM Manager maintains a list of devices in the EFM Manager graphical interface, but also creates a node structure in the EFM data path that can be used as input for other applications. Each accepted device will be found under the kinetic-efm-asset-dslink/Assets, where the name is taken from user defined Label. Labels need not be unique.

EFM Manager Components and functions

The EFM Manager encompasses the following components:

- Application Server Backend, an Asset DSLink
- EFM Manager UI Web component
- A set of (example) device onboarding definition files and device simulation DSLink.

The Application server backend is responsible for discovering the devices to be onboarded using the DQL[1] based search pattern from the onboarding definition files. It also persists the device onboarding decision of the end-user. This package also contains the kinetic-efm-dslink-bridge module which acts as the connection point between the application server and the DSA network.

The Asset DSLink (kinetic-efm-asset-link) is implemented in Java and its purpose is to transform Low Level Device Driver DSLink data into normalized device/device structure. It is also exposing the describing dimensions that an end-user can enter inside the EFM Manager UI.

The EFM Manager UI web component is used in managing the devices and brokers (Accept, Reject, Delete and edit details) by the end-user.

The device onboarding definition, defines the device to be onboarded. The asset class definition, defines the normalized device data model. Example asset definitions are part of the installation and are used in automatically discovering

[1] DQL is a distributed query language for EFM. DQL allows for queries across a single broker or optionally across a multi-broker system. For more details on using DQL and examples see https://github.com/IOT-DSA/dslink-dart-dql/blob/master/README.md .

© 2018 Cisco and/or its affiliates. All rights reserved.

devices from Device simulation DSLink in EFM Manager UI web component. More details on the onboarding can be found in the Onboarding Configuration section.

The Device simulation DSLink is used to generate simulated device readings for Vibration and Temperature devices. This can be used for demonstration purposes. By default, the DSLink generates 5 Temperature Devices and 5 Vibration Devices. It is an optional component can be removed by the system administrator. More details on the device-simulator can be found in the Device Simulator section.

Logging in to the EFM Manager

In order to connect to the EFM Manager, enter default URL is the following `https://[Server IP Address]`. If not already logged into the EFM Manager, it is necessary the user insert their username and password and click “Log In” to continue. Once logged in, the authorized applications will appear on the landing page. After, select the “EFM Manager” to enter the application.

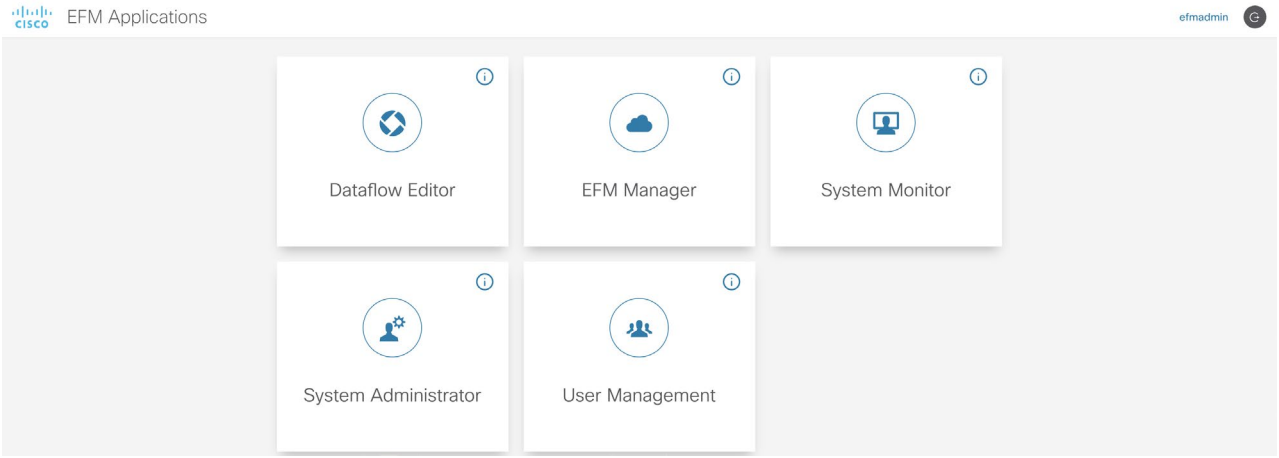


Figure 1. EFM Main applications landing page

© 2018 Cisco and/or its affiliates. All rights reserved.

User Interface Overview

The user interface of the Kinetic – EFM Manager contains two main pages. In the leftmost portion of the UI, a selector provides the ability to switch between two different views. The options are:

1. Devices – All the approved devices appear in the Devices page. This is the initial page when logging in. This page allows for viewing, editing and removing the approved devices. But it also displays (see below Accept/Reject Device) discovered devices pending acceptance or rejection.
2. Brokers – All the approved brokers appear in the Brokers page. This page allows for viewing, editing and removing the approved brokers. But it also displays (see below Accept/Reject Broker) discovered brokers pending acceptance or rejection.

Devices Page

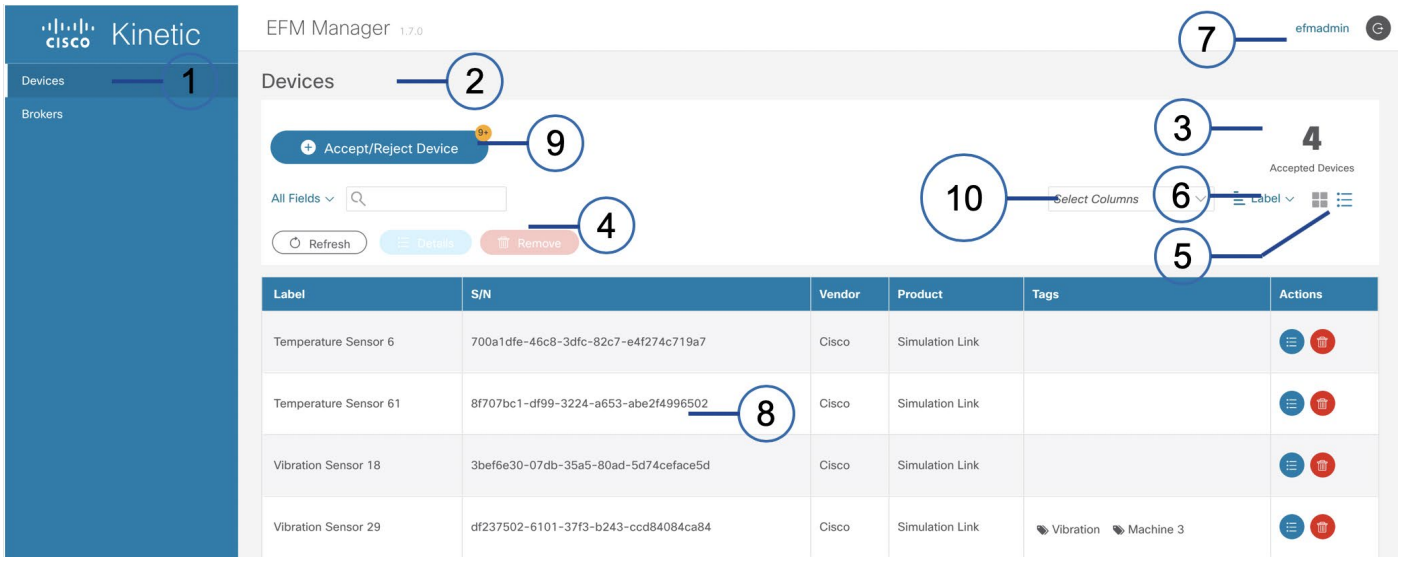


Figure 2. EFM Manager and View Selection (List Mode)

© 2018 Cisco and/or its affiliates. All rights reserved.

1	View Selection pane	2	Devices List Title
3	Summary counts of devices	4	Search –The text box allows for the user to input the search criteria. Refresh – Refresh the list Details – enter the details page of the selected item Remove – delete the accepted device from the list
5	Block (left) or detailed list (right) selector for devices	6	Ascending Sort criteria. Field selection is built dynamically based upon definitions.
7	Username (left) and logoff (right)	8	Devices List
9	Accept/Reject Device – once selected, enters the Devices Accept/Reject Device page. Devices pending will appear on this screen.	10	Select Columns (list mode only) – select one or more fields to appear on the list

Viewing and updating a Device

Select the device from the list, then select the “Details” or Details symbol for the specific device in the list to see the details. The page will be updated with the details defined in the onboarding definition files. For the simulated temperature device, the example is as follows.

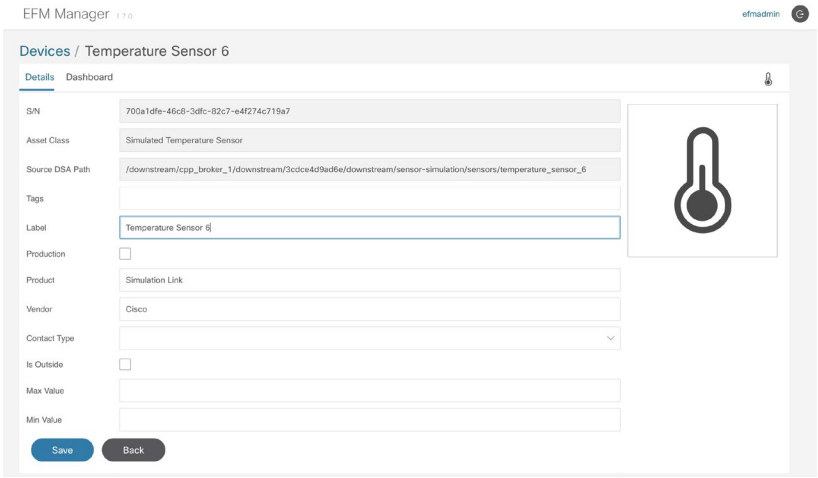


Figure 3. Device Details

The device detail allows for viewing and editing the device attributes that are permitted. Select Save to store the updated information.

© 2018 Cisco and/or its affiliates. All rights reserved.

Device Dashboard

Select the “Dashboard” tab allows the user to create a dashboard with streaming text or charts of the updated telemetry. The dashboard is specific to this device type.

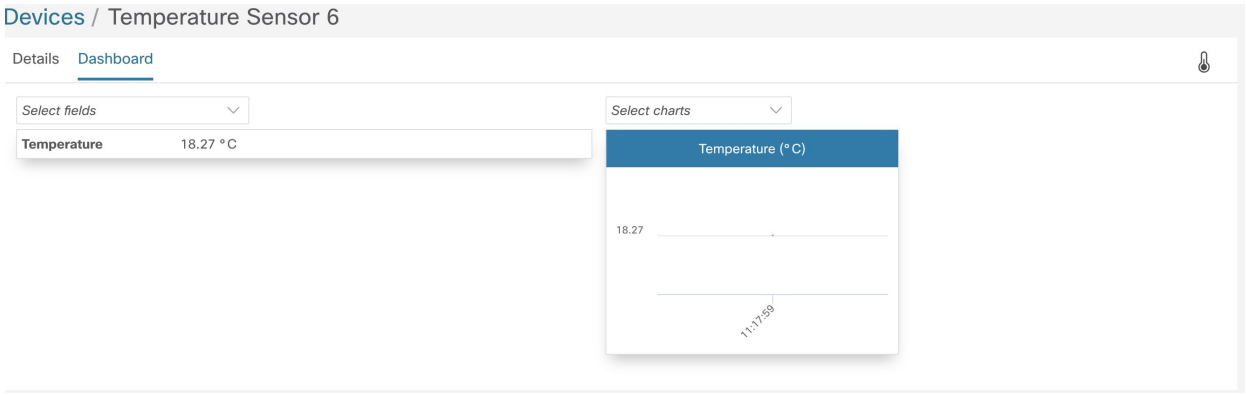


Figure 4. Device Dashboard

Select the fields using the dropdown selectors on the left for text-based output and on the right for charts.

Removing a Device

Select the “Remove” or trash can symbol for the specific device in the list to remove. It will be removed from the Devices list. A removed device may reappear as a rejected discovery entry.

© 2018 Cisco and/or its affiliates. All rights reserved.

Devices Page and Device onboarding

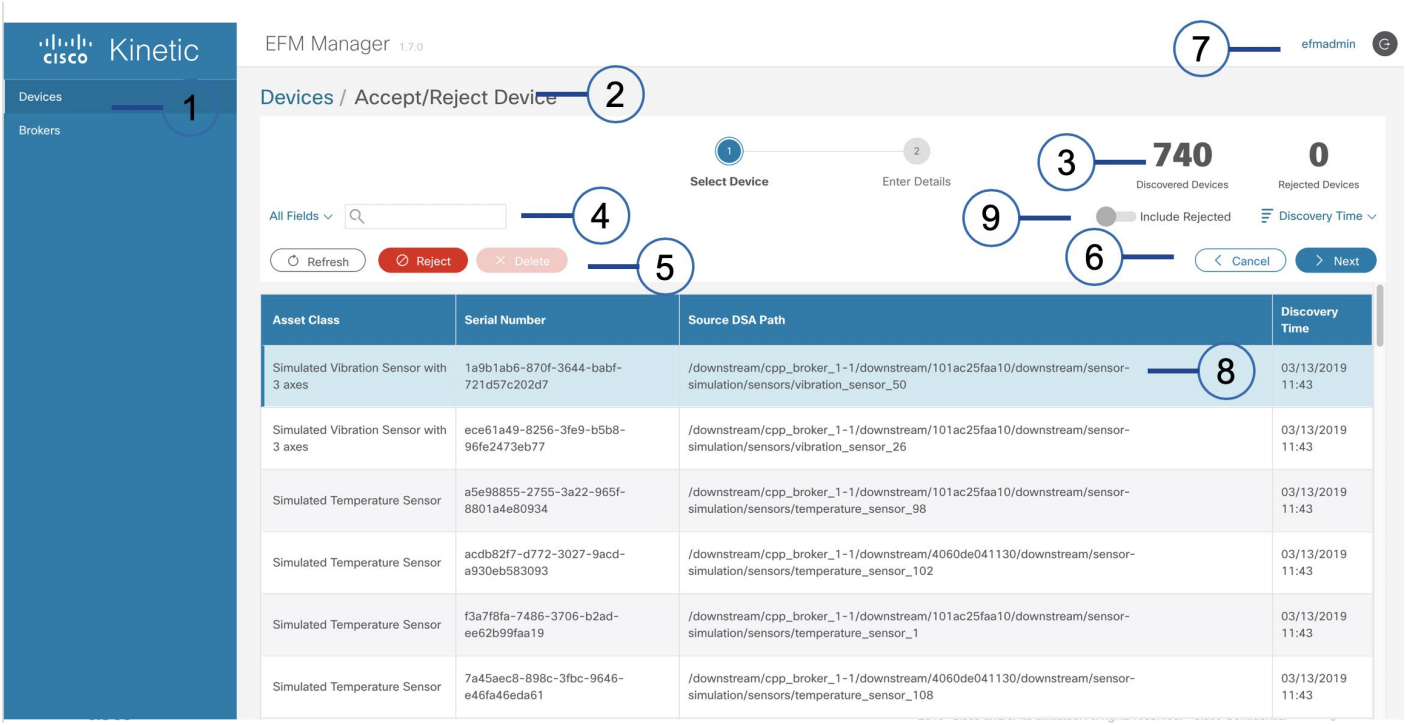


Figure 5. Onboarding Devices – Discovered Devices page

1	View Selection pane	-	**2**	Devices Accept/Reject Device Title
3	Summary counts of devices – Discovered and Rejected		**4**	Search – The text box allows for the user to input the search criteria.
5	Refresh – refresh list Reject – remove selected item from the discovered list. Does not delete entry. Delete – N/A		**6**	Cancel – Return to accepted devices list Next – proceed to details page for onboarding device
7	Username (left) and logoff (right)		**8**	Discovered Devices List
9	Include Rejected – adds rejected devices to list Descending Sort criteria. Field selection is built dynamically based upon definitions.			

© 2018 Cisco and/or its affiliates. All rights reserved.

From the Devices main page, select the “Accept/Reject Devices” to view the list of devices pending acceptance or rejection. Devices that have been deleted from the “Accept/Reject Devices” list can be viewed by toggling the “Include Rejected” switch. Rejected items appear highlighted in red.

The EFM Manager discovery is a continuous process. The EFM Manager queries the EFM message system for devices, that have been defined using the Cisco EFM Device Object Model Standard structure. When a new device is found and it is not already in the Approved Devices list nor the Rejected Devices, it is placed in the Discovered list section.

Accepting a Device

Select a device from the list and select the “Next” icon to proceed. A new page is displayed for user input. When accepting a device, the attribute fields that are defined in the onboarding template for the specific sensor appear. Some fields may be populated by the system such as S/N, Asset Class and DSA Path. Others may allow for user modification or user input.

Once the device is saved, it is placed in the Devices approved list and can be viewed by selecting the “Devices” tab on the left pane.

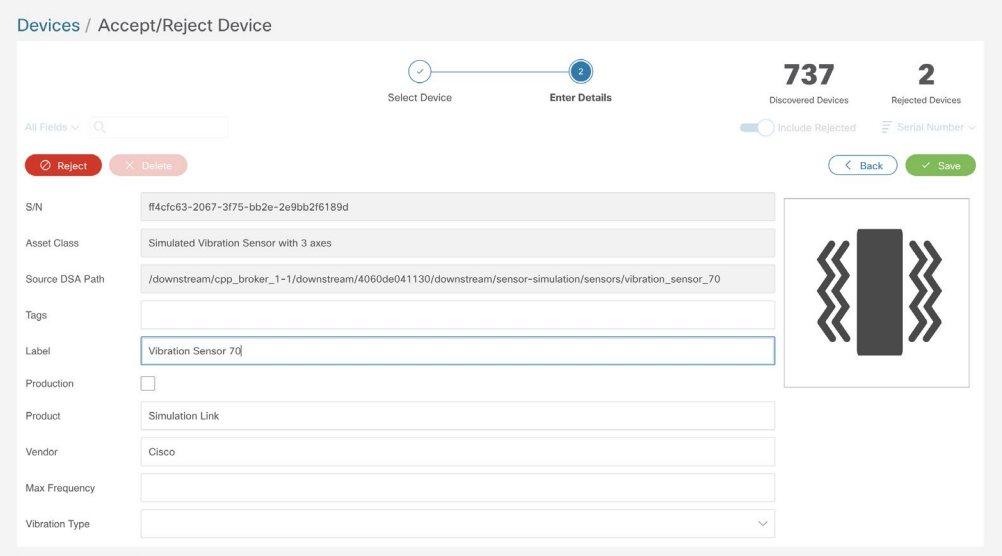


Figure 6. Onboarding a Device – Accepting a Device

Rejecting a Device in the Discovered Device list

If a device is rejected, it will stay in the discovered list, but may not appear on the list if the “Rejected Devices” toggle is off. Once a device is rejected, all rejected devices can be viewed by selecting the “Reject Devices” option on the screen. Even after rejecting a device, the user can decide to accept a device or delete.

© 2018 Cisco and/or its affiliates. All rights reserved.

Brokers Page

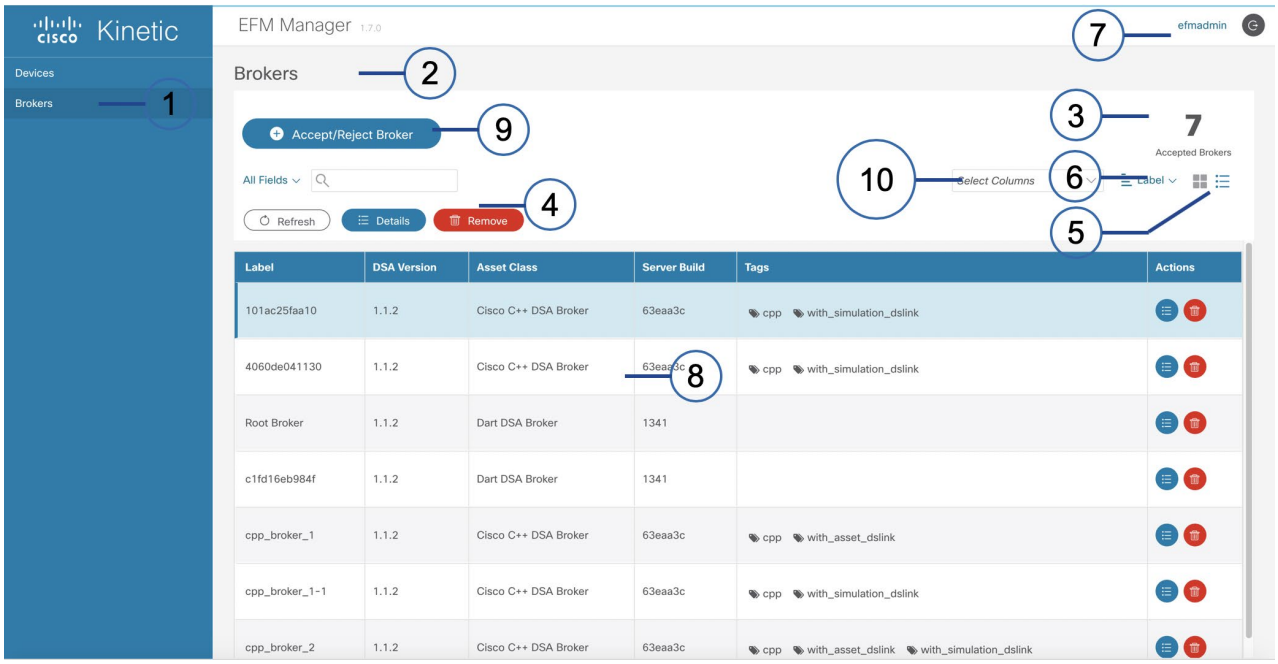


Figure 7. EFM Manager and Broker View Selection (List Mode)

1	View Selection pane	**2**	Brokers List Title
3	Summary counts of accepted brokers	**4**	Search –The text box allows for the user to input the search criteria. Refresh – Refresh the list Details – enter the details page of the selected item Remove – delete the accepted device from the list
5	Block (left) or detailed list (right) selector for brokers	**6**	Ascending Sort criteria: • DSA Version • Label • Server Build
7	Username (left) and logoff (right)	**8**	Brokers List
9	Accept/Reject Broker – once selected, enters the Brokers Accept/Reject Broker page. Brokers pending will appear on this screen.	**10**	Select Columns (list mode only) – select one or more fields to appear on the list

Selecting the Brokers options in the View Selection pane causes the graphical view to change to the Brokers page.

© 2018 Cisco and/or its affiliates. All rights reserved.

Viewing and updating a Broker

Select the “Details” or Details symbol for the specific broker in the list to see the details. The Details, Dashboard and Relationship tabs change the specific pages for the broker.

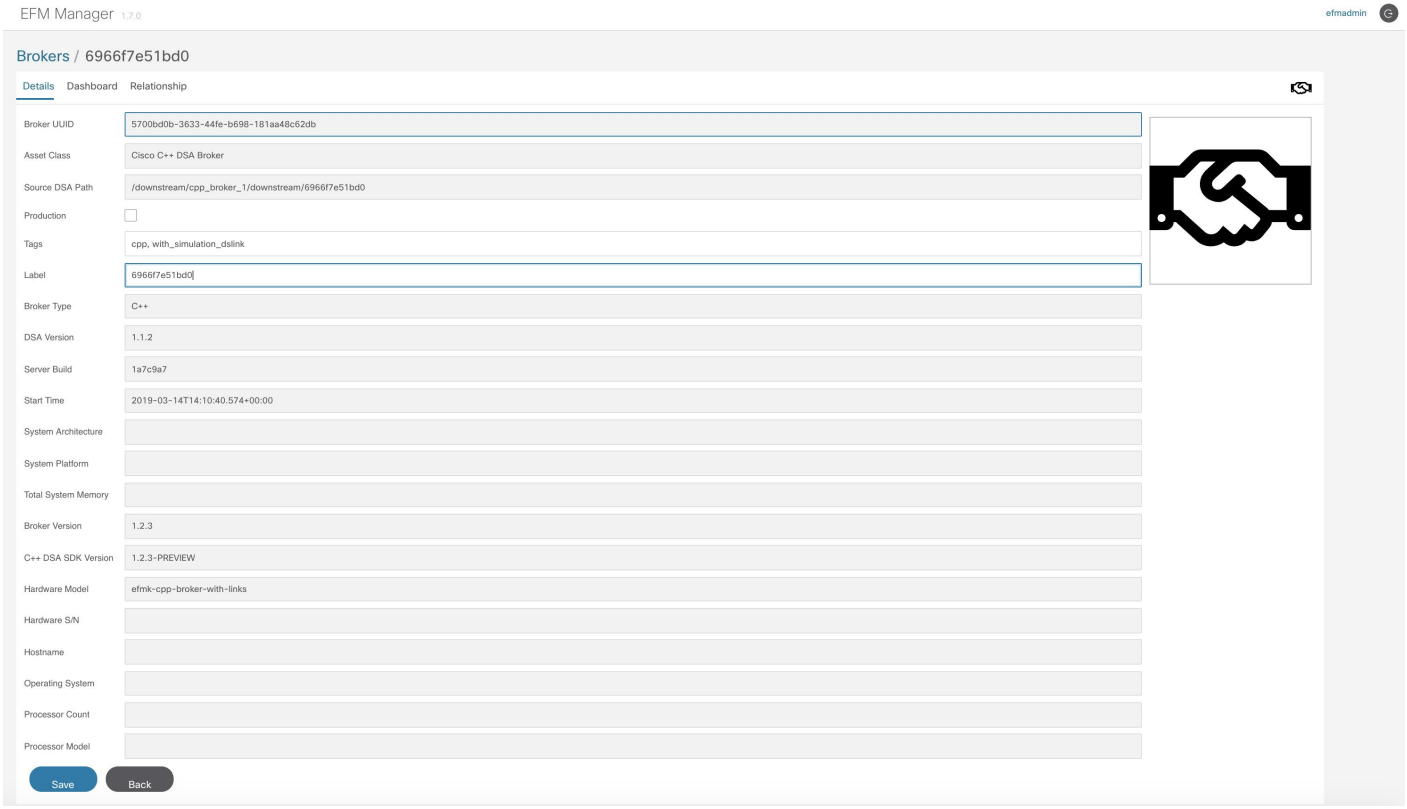


Figure 8. Broker Detail

The broker detail allows for viewing and editing of the broker attributes, some fields may allow for updating. Select Save to store the updated information.

Broker Dashboard

Select the “Dashboard” tab allows the user to create a dashboard with streaming metrics updated data of the select fields or to create charts with the fields available. The dashboard is specific to this broker.

© 2018 Cisco and/or its affiliates. All rights reserved.

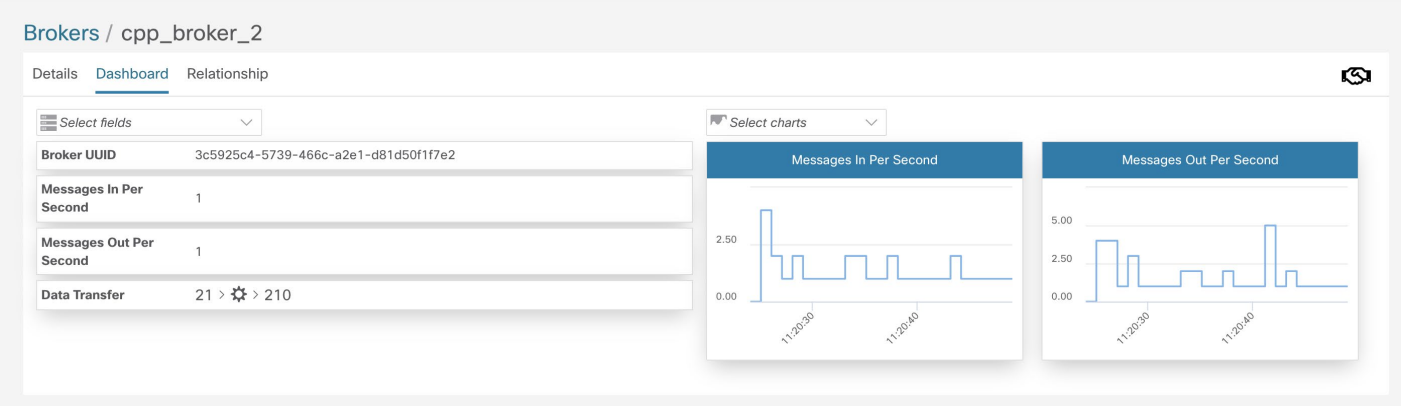


Figure 9. Broker Dashboard

Select the fields using the dropdown selectors on the left for text-based output and on the right.

Broker Relationship

Select the "Relationship" tab allows the user to see the list of upstream and downstream brokers from the select broker. The relationships show is specific to this broker.

1	Broker Detail tabs	**2**	Search -The text box allows for the user to input the search criteria.
3	Refresh – Refresh the list Go to Relationship – when a specific broker and this item is selected, a new page shows the relationship list of the selected broker	**4**	Select the fields to display in the list
5	Ascending Sort criteria: • Broker Type • Broker UUID • Creation Time • Label • Relationship Type	**6**	Brokers List

© 2018 Cisco and/or its affiliates. All rights reserved.

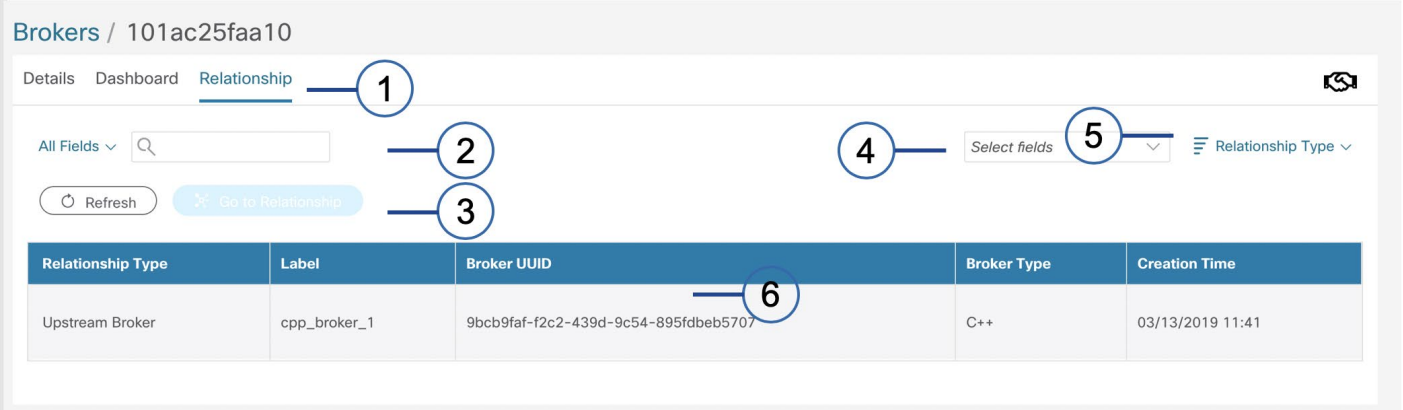


Figure 10. EFM Manager Broker Relationship – see description table above

Select the fields using the dropdown selector.

Removing a Broker

Select the “Remove” or trash can symbol for the specific broker in the list to remove. It will be removed from the Brokers list. A broker may reappear as a rejected discovery entry.

Broker Page and Broker onboarding

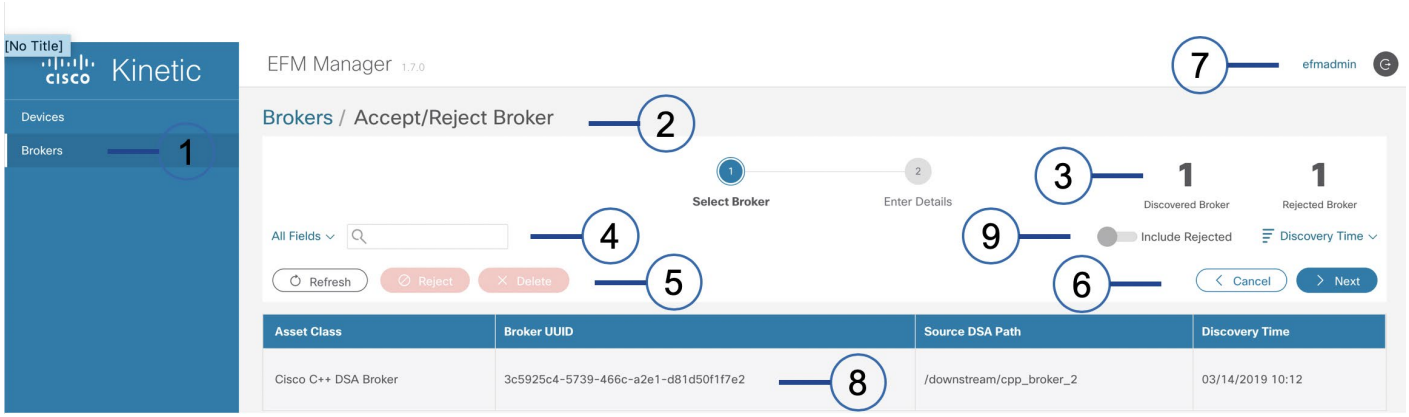


Figure 11. Onboarding Brokers – Discovered Brokers page

© 2018 Cisco and/or its affiliates. All rights reserved.

1	View Selection pane	-	2	Brokers Accept/Reject Broker Title
3	Summary counts of brokers – Discovered and Rejected		4	Search – The text box allows for the user to input the search criteria.
5	Refresh – refresh list Reject – remove selected item from the discovered list. Does not delete entry. Delete – N/A		6	Cancel – Return to accepted brokers list Next – proceed to details page for onboarding device
7	Username (left) and logoff (right)		8	Discovered Brokers List
9	Include Rejected – adds rejected devices to list Descending Sort criteria: • DSA Version • Label • Server Build			

From the Brokers main page, select the “Accept/Reject Brokers” to view the list of brokers pending acceptance or rejection. Brokers that have been deleted from the “Accept/Reject Brokers” list can be viewed by toggling the “Include Rejected” switch. Rejected items appear highlighted in red.

The EFM Manager discovery is a continuous process. The EFM Manager queries the EFM message system for brokers. When a new broker is found and it is not already in the Approved Brokers list nor the Rejected Brokers, it is placed in the Discovered list section.

Accepting a Broker

Select a broker from the list and select the “Next” icon to proceed. A new page is displayed for user input. Some fields may be populated by the system. Other fields may allow for user modification or user input.

Once the broker is saved, it is placed in the broker approved list and can be viewed by selecting the “Brokers” tab on the left pane.

© 2018 Cisco and/or its affiliates. All rights reserved.

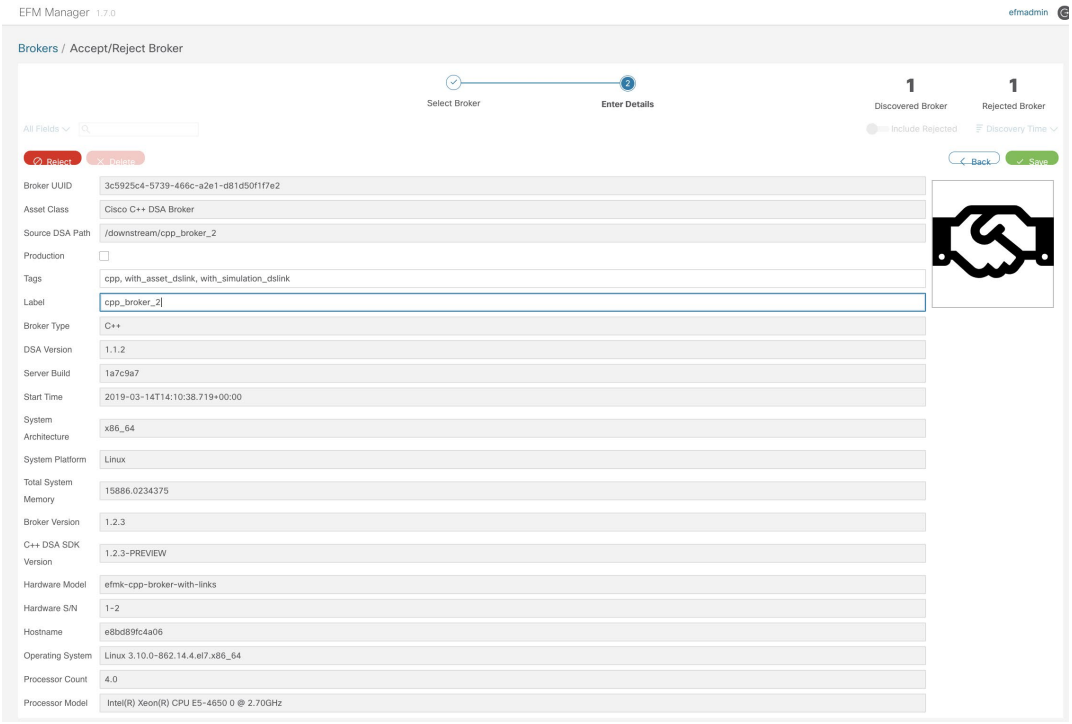


Figure 12. Onboarding a Broker – Accepting a Broker

Rejecting a Broker in the Discovered Broker list

If a broker is rejected, it will stay in the discovered list, but may not appear on the list if the “Rejected Devices” toggle is off. Once a device is rejected, all rejected brokers can be viewed by selecting the “Reject Devices” option on the screen. Even after rejecting a broker, the user can decide to accept a broker or delete.

© 2018 Cisco and/or its affiliates. All rights reserved.

Normalized Asset Synchronization

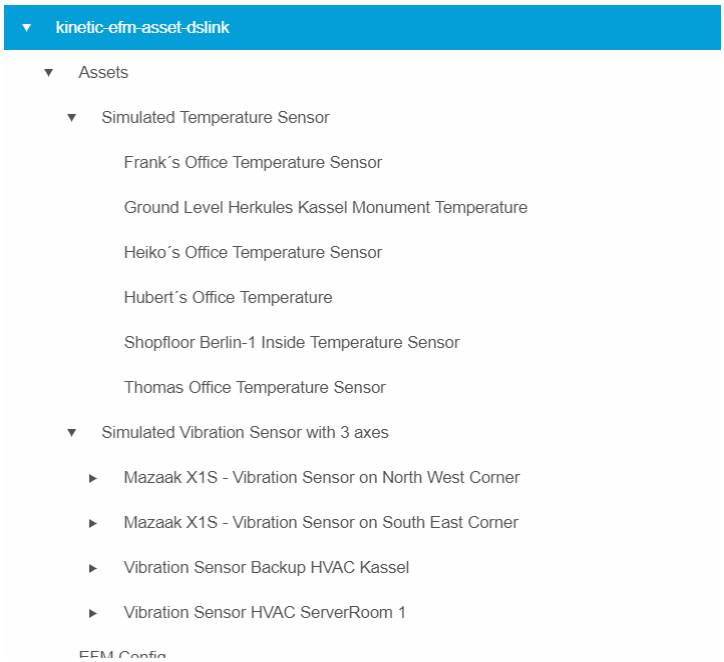
The EFM Manager maintains synchronization between the Asset Registry and the node structure under the Asset DSLink (kinetic-efm-asset-dslink/Assets). The synchronization process provides the following:

- On removing device or broker from EFM Manager UI, the entry is removed from Asset DSLink.
- On removing a device or broker from Asset DSLink, the entry gets recreated.
- The Changes done on the EFM Manager UI are immediately reflected in Asset DSLink.

Normalized Asset Grouping

The kinetic-efm-asset-dslink 1.7 link supports device grouping under the node structure downstream/kinetic-efm-asset-dslink/Assets. The grouping is performed per the label indicated in the configuration schema file(graphql file).

Asset grouping example



© 2018 Cisco and/or its affiliates. All rights reserved.

Backup and restore of EFM Manager

The vital information from EFM Manager can be backed up and restored.

Implications:

- The EFM Manager efm-manager needs to be stopped for backup and restore
- Files are copied into the same directories they have been backed up from
- During restore Ignite data is deleted

For Backup:

1. Stop the EFM Manager efm-manager:
   `service efm-manager stop`
2. Execute
   `java -cp lib/efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher backup --outputDir=/path/to/backup`
3. Default parameters such as `--configDir`, `--workingDir` and `--conf` are supported
   a. `configDir` - the directory of the application where configuration is stored
   b. `workingDir` - the working directory of the application, used for storing ignite data and DSLink data
   c. `conf` - default config parameter defining json configuration file
4. A zip files will be created if backup is successful, during the backup a directory named `_build` will be created in the `outputDir`
5. An Example is given below.
   This command can be run from `/opt/cisco/kinetic/efm_manager/lib`
   ```
   java -cp efm-servicelayer* com.cisco.efm.launcher.ApplicationLauncher backup --
   outputDir=/home/efm/ --configDir=/opt/cisco/kinetic/efm_manager/config/ --
   workingDir=/opt/cisco/kinetic/efm_manager/app-data/ --
   conf=/opt/cisco/kinetic/efm_manager/config/application-conf.json
   ```

   This command can be run from `/opt/cisco/kinetic/efm_manager/`
   ```
   java -cp lib/efm-servicelayer* com.cisco.efm.launcher.ApplicationLauncher backup --
   outputDir=/home/efm/
   ```
6. On executing the above command, the logs of execution are written in to log/server.log relative to the path from where the script is executed.

© 2018 Cisco and/or its affiliates. All rights reserved.

For Restore:

1. A successful installation of the efm-manager is required before restoring
2. Stop efm-manager
   `service efm-manager stop`
3. The folder `config` and `app-data` in `/opt/cisco/kinetic/efm_manager/` will be overwritten. It is necessary to delete the folder before a restore.
4. Start restore
   `java -cp efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/path/to/backup.zip`
5. No other parameters are required to restore.
6. An example is given below:
   This command can be run from `/opt/cisco/kinetic/efm_manager/lib`
   `java -cp efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/home/efm/backup-1540303333.zip`

   This command can be run from `/opt/cisco/kinetic/efm_manager/`
   `java -cp lib/efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/home/efm/backup-1540303333.zip`
7. On executing the above command, logging would be shown. Look for message that says " Restored all service data" .
8. Check whether the `config` folder and the `app-data` folder are restored and run the following command in case `efm` is not the owner of the folder.
   `chown -R efm:efm config/`
   `chown -R efm:efm app-data/`
9. Start efm-manager
   `service efm-manager start`
10. Restart `efm-asset-dslink` from EFM System Administrator or dataflow editor

© 2018 Cisco and/or its affiliates. All rights reserved.

Configuring the EFM Manager via the application-conf.json file

The EFM Manager is configured in the `/opt/cisco/kinetic/efm_manager/config/application-conf.json` file.

Configuration Property Name / Path	Description	Mandatory	Default Value
asset-registry/normalization/assetSynchronizationInitialDelaySeconds	Initial delay for asset DSLink synchronization in seconds. When the value is negative or zero initial synchronization is disabled.	N	30 s
asset-registry/normalization/assetSynchronizationTimeoutMinutes	Total timeout minutes for asset DSLink synchronization.	N	5 min
asset-registry/normalization/assetSynchronizationListIdleTimeoutSeconds	Idle timeout seconds for asset DSLink synchronization list request. After the given number of seconds without message from the broker the list request is closed.	N	15 s
asset-registry/normalization/actionTimeoutSeconds	Timeout seconds for asset DSLink action invocations	N	30 s
asset-registry/normalization/enabled	Boolean that indicates if the asset DSLink integration is enabled.	N	true
asset-registry/normalization/assetSynchronizationIntervalMinutes	Interval minutes for asset DSLink synchronization. Default value is 10. When the value is negative, or zero synchronization is disabled.	N	10 min
asset-registry/discovery/enabled	Boolean that indicates if the device discovery is enabled.	N	true
asset-registry/discovery/fetchDataTimeoutSeconds	Timeout seconds for fetching a single dimension value from a source DSLink node.	N	60 s
asset-registry/discovery/restartIntervalMinutes	Interval minutes between scheduled restarts of the device discovery workflow.	N	10 min
asset-registry/discovery/startDelaySeconds	Delay before initial first start of device discovery.	N	10 s
asset-registry/discovery/queryIdleTimeoutSeconds	Idle DQL / List timeout seconds.	N	15 s
asset-registry/listenerInvocationTimeoutSeconds	Amount of minutes a before asset registry change listener has to respond until a timeout occurs.	N	2 min
asset-registry/listenerMaxRetryWaitSeconds	Max. number of seconds between two retry attempts.	N	30 s
asset-registry/listenerInvocationRetryCount	Max. number of retry attempts	N	5
web-app/jwt/usernameClaimKey	JWT token claim name for user name	N	username
web-app/jwt/permissionsClaimKey	JWT token claim name for user permissions array	N	permissions
web-app/jwt/pubSecKeys	Array of public key information for JWT authentication. The array has to contain objects with keys publicKey and algorithm.	Y	

© 2018 Cisco and/or its affiliates. All rights reserved.

web-app/noAuthX	Disables authentication requirement	N	false
dslink-bridge/broker	Broker connection url. This has to be the url for the root broker.	Y	
dslink-bridge/token	Optional credential token	N	
dslink-bridge/listTimerIntervalMillis	Interval milliseconds in between checks if a list request is considered to be finished	N	500 ms
dslink-bridge/listIdleTimeoutSeconds	Timeout seconds for a list request. If no response is being received for the specified amount of time the list request is considered to be finished.	N	5
dslink-bridge/requestTimeoutSeconds	Total timeout seconds for a DSA request.	N	30 s
dslink-bridge/customDQLQuery	Custom DQL query for resolving DSLinks by sysId	N	option traverseBrokers =true | list * | filter $sysId=" %s" | subscribe :name $base $sysType $sysVersion
dslink-bridge/dqlPath	Optional DQL path used for global queries	N	
config-installer/discoveryConfigFolder	Folder from where asset class definitions and discovery definitions are loaded and installed. The files can be placed in sub-folders. If an absolute path is configured the absolute path is used. Otherwise the path is interpreted relative to the configuration folder.	N	discovery
runtime/systemdNotifierEnabled	Flag if the systemd notifier workflow should be enabled	N	true

© 2018 Cisco and/or its affiliates. All rights reserved.

Onboarding Configuration and Workflow

Sensor Normalization Workflow

The sensor normalization workflow involves device on-boarding and sensor normalization. The on-boarding process is about finding suitable source DSLink node structures that can be mapped to a certain sensor model schema.

The result of the sensor on-boarding is a representation of the source sensor data in the EFM Asset DSLink.

The configuration consists of at least two files that describe the normalized sensor (**Asset Schema Definition**) and a source DSLink mapping (**Device Discovery Definition**).

Asset Schema Definition

Normalized sensors are described by a schema. The schema defines the dimension and metric fields and their value scopes. There are some fields like *id* or *label* that all sensors share and can be generalized to an abstract sensor base type. The sensor schema definition needs a serialized format because adding of new schemas is expected to work at runtime. The serialized sensor schema will be part of a data set that is used by the API for adding and removing sensor schema definitions.

Syntax

The sensor types need to be described in GraphQL schema. It makes sense to use the GraphQL type definition as primary definition format.

Not all semantic requirements are met by the default GraphQL type definition. But, GraphQL has the concept of *directives* which are basically annotations. These annotations are allowed at both type and field level. The concept of *directives* can be used to extend the existing expressiveness of GraphQL type and field definitions to meet our needs.

For that purpose, four *directives* are being defined in our implementation of the GraphQL engine:

- **@asset**: Asset Type Directive
- **@internal**: Internal Field Directive
- **@dimension**: Dimension Field Directive
- **@metric**: Metric Field Directive

For general type definition format the GraphQL documentation can be reviewed. There is an *interface* type for general sensor fields that is called **Device**.

Basic Format Description

This is how an asset type definition will look like:

```
# Simulated Temperature sensor
type SimulatedTemperatureSensor implements Asset & Device @asset(
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
    version: 1
    label: "Simulated Temperature Sensor"
) {
    # Building ID
    building: String @internal(
        label: "ID"
        access: "RW"
    )
    ...
}
```

This example is not complete, some required arguments are missing and would need to be present in an actual type definition.

Device schema definitions will need to implement both **Asset** and **Device** interface. This is a requirement of the GraphQL API that is to be provided by the asset registry.

The asset type directive has to be specified. All dimension and metric fields need to be described. For the first release it is expected that all fields are top level. No nested structures are supported for now.

Asset Interface

```
# Generic interface that is implemented by all asset types
interface Asset {

    # Asset configuration
    assetConfig: AssetConfig! @internal(
        label: "Asset Configuration"
        access: "INTERNAL"
    )

    # Asset ID
    id: ID! @dimension(
        label: "ID"
        access: "INTERNAL"
    )

    # Asset Type Definition ID
    assetTypeDefinitionId: String! @internal(
        label: "Asset Type Definition Id"
        searchable: true
        access: "INTERNAL"
    )

    # Human readable label for asset
    label: String! @dsaMapping(path: ":displayName") @dimension(
        label: "Label"
        searchable: true
    )

    # Tags
    tags: [String!] @dsaMapping(path: "$sysTags") @dimension(
        label: "Tags"
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
        searchable: true
        access: "RW"
    )
}
```

EFMAssetConfig object definition:

```
# EFM Asset Configuration
type EFMAssetConfig implements AssetConfig {

    # Value is static 'ASSET_MANAGER' for the Asset Manager application
    type: String! @internal(
        label: "Asset Configuration Type"
        access: "INTERNAL"
    )

    # DSA Id of Asset DSLink
    assetDSLinkSysId: String! @internal(
        label: "Asset DSLink Id"
        access: "INTERNAL"
    )

    # DSA node path of target normalized asset
    targetAssetPath: String! @internal(
        label: "Target Asset Path"
        access: "INTERNAL"
    )

    # DSA node path of source asset
    sourceAssetPath: String! @internal(
        label: "Source Asset Path"
        access: "INTERNAL"
    )
}
```

The asset interface defines these fields:

- **id**: Unique global identifier of the instance
- **label**: Human readable label for the instance
- **assetConfig**: The asset config contains information about the DSA structure of the given instance
- **assetTypeDefinitionId**: ID of the asset class definition that describes the asset
- **tags**: Tag values can be used to flag instances for searching

Device Interface

```
# Generic interface that is implemented by all device types
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
interface Device {

    # Asset configuration
    assetConfig: AssetConfig! @internal(
        label: "Asset Configuration"
        access: "INTERNAL"
    )

    # Asset ID
    id: ID! @dimension(
        label: "ID"
        access: "INTERNAL"
    )

    # Human readable label for asset
    label: String! @dsaMapping(path: ":displayName") @dimension(
        label: "Label"
        searchable: true
    )

    # Tags
    tags: [String!] @dsaMapping(path: "$sysTags") @dimension(
        label: "Tags"
        searchable: true
        access: "RW"
    )

    # Asset Type Definition ID
    assetTypeDefinitionId: String! @internal(
        label: "Asset Type Definition Id"
        searchable: true
        access: "INTERNAL"
    )

    # Discovery Definition ID
    discoveryDefinitionId: String! @internal(
        label: "Discovery Definition Id"
        searchable: true
        access: "INTERNAL"
    )

    # Device serial number
    serial: String! @dsaMapping(path: "$sysSerial") @dimension(
        label: "S/N"
        unique: true
        searchable: true
        access: "R"
    )

    # Device vendor name
    vendor: String @dsaMapping(path: "$sysVendor") @dimension(
        label: "Vendor"
        searchable: true
        access: "RW"
    )

    # Device product name
    product: String @dsaMapping(path: "$sysProduct") @dimension(
        label: "Product"
        searchable: true
        access: "RW"
    )
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
}
```

The sensor interface redefines all fields of the **Asset** interface and these additional fields:

- **serial**: The serial is a predefined primary key value for sensor instances
- **vendor**: Dimension value for the sensor vendor string
- **product**: Dimension value for the sensor product string
- **discoveryDefinitionId**: ID of the discovery definition by which the device was found

GraphQL Directive Extension

As mentioned above the asset type definition is dependent on domain specific directives. These directives are *@asset* located at the type definition and *@internal*, *@dimension* and *@metric* located at the field definitions.

Asset Directive [@asset]

The *@asset* directive marks a GraphQL type to be an asset type. This directive has two arguments:

version: Int!

The version of the asset definition.

label: String!

Device type label. This argument contains a human readable label for the asset type. The argument is required and there is no default value.

Internal Field Directive [@internal]

The directive *@internal* is one of the three sensor field definition directives that provide additional meta information for fields. This directive is to be used for fields that are neither dimension or metric. Only fields of EFM base domain are neither domain nor metric. User defined fields have to be annotated with *@dimension* or *@metric*. This directive has three arguments:

access: String = `RW`

Access definition for the given field. The purpose of this is to define if the value is mutable. Following values are allowed:

- **`R`**: Field value is readonly=
- **`RW`**: Field value is mutable
- **`INTERNAL`**:

© 2018 Cisco and/or its affiliates. All rights reserved.

- Field value is readable but not mutable by user actions. Actions executed in system context are able to modify the values
- Fields with *access* set to *INTERNAL* will not be published to DSA. There will be no **Asset Link** representation for those values.

- **`EXTERNAL`**: Field value is bound to a specific DSA path. If the value at the path changes and the device discovery realizes the change, then the value is updated in the asset.

The Default value is **`RW`**.

searchable: Boolean = false

Indicator if the field is accessible in search queries. Default value is false.

label: String!

Device field label. This argument contains a human readable label for the field. The argument is required and there is no default value.

Dimension Field Directive [@dimension]

The *@dimension* directive is to be used for dimension asset fields. This directive has six arguments:

access: String = `RW`

Access definition for the given field. The Default value is **`RW`**.

searchable: Boolean = false

Indicator if the field is accessible in search queries. Default value is false.

unique: Boolean = false

Indicator if the field value is unique.

label: String!

Device field label. This argument contains a human readable label for the field. The argument is required and there is no default value.

DSA Mapping Field Directive [@dsaMapping]

The *@dsaMapping* directive is to be used if non default mapping to a DSA path is required.

path: String = `"_DEFAULT_"`

Relative path to respective information in the **Asset DSLink**. Default value is *`"_DEFAULT_"`*. Using the default value will derive a default path for the respective dimension value that looks like this: `$<field name>`. Valid path value possibilities are:

© 2018 Cisco and/or its affiliates. All rights reserved.

- **@<attribute name>**: DSA node attribute
- **$<config name>**: DSA node config item
- **:displayName**: DSA node display name
- **:value**: DSA node name
- **/<sub node name>:**
  - Path to DSA sub node.
  - This can also be combined with one of the paths above for navigation to a sub node:
  - `/<sub node1>/<sub node 2>/@<attribute name>`

Metric Field Directive [@metric]

The *@metric* directive is to be used for metric asset fields. This directive has three arguments:

label: String!

Asset field label. This argument contains a human readable label for the field. The argument is required and there is no default value.

unitName: String

Contains the metric unit name. The nunit name is being used for value conversion. There is a list of supported unit names and their semantics.

unitSymbol: String

Contains the metric unit symbol.

© 2018 Cisco and/or its affiliates. All rights reserved.

Devices Onboarding Definition

For DSA devices to be added to the EFM Device Manager, there needs to be a definition on how the node structure should be transformed to a normalized EFM Manager device. Device onboarding definitions enable exactly this. There can be multiple onboarding definitions for the same target device schema definition.

A device discovery definition contains an `id` and `label` for identifying purpose. The field `assetType` defines the sensor type, which the sensor on-boarding definition finds. As a method of finding several `discoveries` are used. They are defined as an array of *Discovery Configs*, which describe how to find an asset in DSA. The `dimensionMappings` holds the paths in DSA to the unique identifying field values for the defined `assetType`. The config item `dataflow` defines the mapping of metric fields from DSA to EFM. The field `productImageRef` is used to reference a UI icon that will be shown hen displaying discovery entries. Default dimension values for on-boarded sensors can be defined in `defaultValues`. The field `discoveryClassId` has to define a unique value that represents the given source DSLink in its target version. Therefore `discoveryClassId`, `assetType` and `assetTypeVersion` define what DSLink source sensor will be on-boarded into which device schema definition.

Discovery Definition Example:

```
{
   "label": <String>
   "discoveryClassId": <String>
   "assetType": <String>
   "assetTypeVersion": <Integer>
   "productImageRef": <String>
   "dimensionMappings": <JsonArray>
   "defaultValues": <JsonObject>
   "discoveries": <JsonArray>
   "dataflow": <JsonObject>
}
```

Device Discovery Configs

Currently only DQL is supported as a means to find new sensor DSA structures. For that purpose, configuration needs to be placed in the on-boarding definition in the field `discoveries`. Every such query defines a DQL statement that will return the paths of the top-level source sensor node.

Device Discovery Configuration format example:

```
{
   "type": <String>
   ...
}
```

Dimension Mappings

The discovery definition field `dimensionMappings` holds information on how the discovery can resolve values for dimensions of an asset. The field `dimensionMappings` is an array of objects, where `type` is the name of the resolution type and `fieldName` defines the sensor schema definition field name (dimension name). There are more properties in each object which are resolution type specific and described below.

DSA Dimension Mapping

© 2018 Cisco and/or its affiliates. All rights reserved.

For DSA `dimensionMappings` the structure of a dimension mapping object looks like the example below. `targetPath` is specific to the `DSA` type here and contains the DSA path expression which is to be used to resolve the dimension value. `targetPath` is relative to the source sensor discovery path.

DSA Dimension Mappings example:

```
[
    {
        "type": "DSA",
        "targetPath": "/$sysId",
        "fieldName": "serial"
    }
]
```

DQL Dimension Mapping

For DQL `dimensionMappings` the structure of a dimension mapping object looks like the example below. `query` is specific to the `DQL` type here and contains the DQL expression which is to be used to resolve the dimension value(s). Only using the `DQL` dimension mapping type it is possible to resolve a list value, i.e. a list of strings or numbers. However, for this to work the field definition in the asset class definition must be of a list type. In the below example, if the `nodeLabels` is a list ([String] type in the asset class definition), it will store the labels of all found nodes and if not, it will just use the first one found.

DQL Dimension Mappings example:

```
[
    {
     "type": "DQL",
     "filter": "<DQL Query which fetches the device lists>"
    }
]
```

The `query` has some special properties:

There is the variable `${SOURCE_ASSET_PATH}` that will be replaced with the actual source asset path. That means the path under which the sensor / device which this dimension belongs to was discovered, i.e. `/downstream/sensor-simulation/sensors/temperature-sensor-1`.

the subscription / list table must contain a column named `value`. Only the content of this column will be used as value for the dimension.

The query runs and collects rows until there was no new row for 5 seconds. Then it creates the dimension value from the received rows. To change the amount of idle timeout seconds, update the config item `asset-registry/discovery/dimensionResolvingIdleTimeoutSeconds`

Asset Dataflow Config

There are two representations of sensors in EFM. One is the set of all fields after a sensor is on-boarded to DSA. The other (EFM), once an asset has been accepted and added to the list of accepted assets using EFM UI. Dataflows are used to map the fields from DSA to EFM. There are multiple ways to achieve that. The config that is stored in the database is a String > Object map that holds a dataflow configuration for each metric field name. The actual config is a

© 2018 Cisco and/or its affiliates. All rights reserved.

String > Object map with the fields `type` and `config`, where `type` specifies a certain configuration kind and `config` is the actual configuration.

Asset Dataflow Config example:

```
{
   <metric_field_name>: {
      "type": "write-through-dataflow"
      "config": <JsonObject>
   }
}
```

Currently only `write-through-dataflow` configurations are supported. They consist of a single configuration property `path` that defines the relative path to the source sensor metric, relative to the root node of the source sensor DSA tree.

EFM Device Definition files

The EFM Manager is an extensible component and supports many types of device definitions. It ships with the simulator link and the corresponding device onboarding configuration file definitions for vibration and temperature sensors. In addition, there is a set of template files to make the process of onboarding definition easier for the user. These will be explained in the next section.

All EFM Manager configuration files are placed in the `$EFM_ROOT/efm_manager/config/discovery` directory. They can be grouped under folder if desired, such as Panduit-link, simulation-link, template, etc. Each device requires two file definitions:

- Asset Onboarding Definition for the device (source DSLink mapping). This is a json formatted file. This file defines the inputs for the device.
- Asset Class that corresponds to the specific device Definition (describes the normalized device). This is a GraphQL formatted file. This file defines what the outputs will be for the device.

Creating user Onboarding Definition files using the Onboarding Definition Templates

The EFM Manager installation includes a generic set of onboarding definition files that we recommend using as templates. These can be found under the efm_manager/config/discovery/template directory. These files contain several sections, but we describe what fields are user configurable as well as the mandatory input fields.

Note that the files suffix terminate with .template. This termination is not a valid .json or .graphqls suffix and is ignored by the EFM Manager at startup or restart. The user must copy and rename the template files for detection by the EFM Manager.

The EFM Manager template configuration files are placed in the `$EFM_ROOT/efm_manager/config/discovery/template` directory. Copy the files to a new directory in the `$EFM_ROOT/efm_manager/config/discovery` path. For example, power-link.

Each device requires two file definitions:

© 2018 Cisco and/or its affiliates. All rights reserved.

- Asset Onboarding Definition for the device (source DSLink mapping). This is a json formatted file. This file defines the inputs for the device. The template file is device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template. It is suggested that the user replace ((assetType)) with the assetType name and v((assetTypeNumber)) with the version number.[2]
- Asset Class that corresponds to the specific device definition (describe the normalized device). This is a GraphQL formatted file. This file defines what the outputs will be for the device. The template file is device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template. It is suggested that the user replace ((assetType)) with the assetType name and v((assetTypeNumber)) with the version number.

For example, we will rename the json file as `asset_discovery_definitions_temperatureSensor_v2.json` and graphqls file as `asset_class_temperatureSensor_v2.graphqls`.

In the following sections, we will describe the template files and the sections that can be modified. After these descriptions, example to discover message brokers and onboard them as devices are explained.

Device Discovery Definition JSON Template explained

This file is a JSON formatted file that defines how a device class is discovered and where it is placed into the broker data path. The template file name is "`device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template`". An example file name is `device_discovery_definitions_BrokerDevice_v1.json`.

The values surrounded by <> require valid values to be configured. Remove all lines that start with ##, since commenting is not supported in the json file definition.

Note that if a device has been discovered with an existing device onboarding definition, modification of the device definition file is not recommended. Rather it is recommended a new instance be created increasing the version number inside the file, for example from version 1 to 2, 2 to 3, and so on.

This file has the following functional sections:

- General definitions (label, discoveryClassId, assetType, assetTypeVersion and productImageRef)
- Dimension Mapping, an array that defines a **static** field to be read from the discovered node, which will contain the serial field mapping and any other static fields
- Discoveries, the method of finding devices is an array of Discovery Configs or queries
- Dataflow, an optional array that defines a **streaming** field to be read from the discovered node. A Dataflow field can only be output as streaming value in the node path under the "label", the EFM Manager UI does not show this field.

[2] The JSON file format is a standard JSON structure. All comments must be removed from the template for proper parsing to occur.

© 2018 Cisco and/or its affiliates. All rights reserved.

[
  {
  "label": "<Label for Discovery>",
  "discoveryClassId": "<Discovery Class ID>",
  "assetType": "<Name for Asset Type>",
  "assetTypeVersion": <Version Number for Asset Type>,
  "productImageRef": "<static/images/cisco.svg>",
	Section 1. - General definitions:
	
	Label - Discovery name, for example "Simulated Temperature Sensor On-Boarding''
	
	discoveryClassId - unique name definition, for example "SimulationDevice-TemperatureSensor-1.0"
	
	assetType - unique asset type, for example "SimulatedTemperatureSensor"
	
	assetTypeVersion - asset type configuration version
	
	productImageRef - icon image that appears in the EFM Manager UI. Path relative to $EFM_ROOT/efm_manager /web/efm_manager/static/images/cisco.svg

"dimensionMapping": [
  {
  "type": "DSA",
  "targetPath": "<Source DSA Path to read dimension>",
  "fieldName": "<uniqueFieldName>"
  }
	Section 2. - Dimension Mapping
	
	Type - always DSA
	
	tartgetPath - Source DSA Path to read dimension
	
	fieldname - unique field name for the field read dimension
	
	If more than one dimension needs to be read, repeat the red highlighted lines for those many times and separate them by a comma

"discoveries": [
  {
  "type": "DQL",
  "query": "<DQL Query which fetches the device lists>"
  }
	Section 3. - Discoveries
	
	Type - DQL
	query - DQL query statement
	
	If more than one DQL needs to be run, repeat the above red highlighted lines for those many times and separate each of them by a comma

"dataflow": {
  "<uniqueFieldName>": {
  "type": "write-through-dataflow",
  "config": {
  "path": "<Source DSA Path to read metric value>"
  }
  }
	Section 4. Dataflow
	
	uniqueFieldName - unique field name for the field read metric
	type - must be "write-through-dataflow"
	path - source of DSA path to read metric value (Relative path from broker where efm manager is connected to)
	
	If more than one metric needs to be read - repeat the above red highlighted lines for those many times and separate each of them by a comma

© 2018 Cisco and/or its affiliates. All rights reserved.

Device Discovery Definition JSON Template reference

This section details the device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template reference file. This first section is the template file followed by a detailed description of the sections of the file.

```
## <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values. Consider |
rule explained above a
s well. Remove characters < and > as well ##
## Remove lines that start with ## characters as well ##
[
  {
    "label": "<Label for Discovery>",
    "discoveryClassId": "<Discovery Class ID>",
    "assetType": "<Name for Asset Type>",
    "assetTypeVersion": <Version Number for Asset Type>,
    "productImageRef": "<static/images/cisco.svg>",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "<Source DSA Path to read dimension>",
        "fieldName": "<uniqueFieldName>"
      }
      ## If more than one dimension needs to be read - repeat the above 5 lines for those many times and sep-
arate them by comma##
    ],
    "defaultValues": {
      "<uniqueFieldName>": "<Default Value to be entered into this dimension field>"
      ## If more than one dimension needs to be defaulted - repeat the above 1 line for those many times and
separate them by comma#
#
    },
    "discoveries": [
      {
        "type": "DQL",
        "filter": "<DQL Query which fetches the device lists>"
      }
      ## If more than one DQL needs to be run - repeat the above 4 lines for those many times and separate
each of them by comma##
    ],
    "dataflow": {
      "<uniqueFieldName>": {
        "type": "write-through-dataflow",
        "config": {
          "path": "<Source DSA Path to read metric value>"
        }
      }
      ## If more than one metric needs to be read - repeat the above 6 lines for those many times and sepa-
rate each of them by comma
##
    }
  }
]
```

Device Class Definition graphql Template explained

Normalized devices are described by a schema. The schema defines the *dimension* fields (that appear in the UI and also under the EFM Manager data node structure) and *metric* (that are streamed to the EFM Manager data node structure with the label header) fields and their value scopes. There are some fields such as **`id`** or **`label`**, that all devices share, can be generalized to an abstract device base type. The device schema definition needs a serialized format because adding of new schemas are expected to work at start or restart.

© 2018 Cisco and/or its affiliates. All rights reserved.

Note that if a device has been discovered with an existing device onboarding definition, modification of the device class definition file is not recommended. Rather it is recommended a new instance be created increasing the version number inside the file, for example from version 1 to 2, 2 to 3, and so on.

This file has the following functional sections:

- General definitions
- Defining Dimensions, apart from ones defined in Asset or Device class definitions
- Defining Metrics
- Renaming dimensions
- Renaming metrics
- Defining Default Value, needs to be set when changing a nullable to non-nullable dimension
- Defining enum dimensions (a list dimension) required in the asset. The next section defines the enum items that will use a UI drop down list. They must be unique definitions and used once.
- Defining a mandatory set of different enum dimensions required in the asset. The next section defines the enum items which will appear in the UI drop down list. They must correspond to the enum definition in the previous section.

This is the example file "`device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template`".

The definition file has rules that need to be followed:

- Comments are allowed and will be ignored. It is recommended the use of comments for documentation purposes.
- | Rule - Values which have a | in the middle indicate there are choices that user has, by which only one choice can be chosen. Remove all other choices including the | symbol(s)
- <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values. Consider | rule explained above as well. Remove characters < and > as well
- [] Rule - Values surrounded by [] are optional definitions which either needs to be completely removed or to be used exactly as defined. Consider <> rule explained above as well. Remove characters [and] as well
- The optional directive @dsaMapping will be defaulted to $<uniqueFieldName> if the whole @dsaMapping directive is omitted
- The optional directive @rename must be used when intending to rename a field name
- The optional directive @defaultValue for dimension must be used when making a nullable field as a non-nullable field
- The optional argument *unique* will be defaulted to false if whole argument is omitted
- The optional argument *searchable* will be defaulted to false if whole argument is omitted
- The optional argument *access* will be defaulted to RW if whole argument is omitted

© 2018 Cisco and/or its affiliates. All rights reserved.

- The optional argument *unitName* will be defaulted to " " , if whole argument is omitted
- The optional argument *unitSymbol* will be defaulted to " " , if whole argument is omitted
- The type name defined in enum <assetType>_<uniqueEnumFieldDefinition> must be unique across all available asset definitions. Hence <assetType> is also added in the type name
- The type name must match between the field definition and the enum definition, for the enum values to be used

`[#<Comments for type of asset>]`
`type <AssetType> implements Asset & Device @asset(`
`  version: <AssetTypeVersion>`
`  label: "<Label for Asset Class>"`
`) {`
	Section 1. - General definitions:
	
	assetType - unique asset type, for example “SimulatedTemperatureSensor”
	
	assetTypeVersion - asset type configuration version(numeric value)
	
	Label -Asset Class name, for example “`Simulated Temperature Sensor`’’
	
	This section is define once.

© 2018 Cisco and/or its affiliates. All rights reserved.

[#<Comment for Dimension Field>] <uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@rename(oldName: "<uniqueFieldNameOld>")] [@defaultValue(value: "<String representation of default value>")] [@dsaMapping(path: "<DSA path expression>")] @dimension(label: "<Display Label of Field>" [unique: <true | false>] [searchable: <true | false>] [access: "<R | RW>"])	**Section 2. - Defining Dimensions, apart from ones defined in Asset or Device class definitions:** Dimension - This field appears in the UI at acceptance. Field will be copied once from source link and as per access type. It can be modified, or this can be a completely new field which will be made available in the normalized asset DSLink uniqueFieldName - unique name. Optional use of "!" to define a non-null field. label - The Label to be shown in the UI unique - indicates if field values is unique (example, a serial value is unique) searchable - To indicate whether the field is searchable in UI access - read and write capability Repeat this block for each dimension to be defined, apart from ones defined in Asset or Device class definitions. As part of installation the following dimensions are already available. They are Serial, Tag, Label, Product and Vendor. In case of renaming the field after initial version is installed, then the @rename directive must be used. In case of making a nullable field as non-nullable after initial version is installed, @defaultValue directive must be used
[#<Comment for Metric Field>] <uniqueFieldName>: <String | Int | Float | Long | Boolean> [@rename(oldName: "<uniqueFieldNameOld>")] [@dsaMapping(path: "<DSA path expression>")] @metric(label: "<Display Label of Field>" [unitName: "<Metric unit name>"] [unitSymbol: "<Metric unit symbol>"])	**Section 3. - Defining metrics:** Metric - which is updated with the latest streamed value from source node uniqueFieldName - unique name. label - The Label for the metric unitName – indicates the Unit Name for Metric unitSymbol – indicates the Unit Symbol for Metric Repeat this block for each metric to be defined In case of renaming the field after initial version is installed, then the @rename directive must be used.

© 2018 Cisco and/or its affiliates. All rights reserved.

`[#<Comment for Dimension Field>]`
`<uniqueFieldName>:`
`<assetType>_<uniqueEnumFieldTypeName>`
`[@dsaMapping(path: "<DSA path expression>")]`
`@dimension(`
`label: "<Display Label of Field>"`
`[searchable: <true | false>]`
`[access: "<R | RW>"]`
`)`
	Section 4. - Defining a set of enum dimensions required in an asset.
	
	The Type <assetType>_<uniqueEnumFieldTypeName> must match between the enum and this field definition.(Section 5)
	
	Repeat the 6 lines for numbers of enum dimensions required in the asset.

`[enum <assetType>_<uniqueEnumFieldTypeName> {`
`<uniqueEnumFieldItemName> @enumValue(label: "<Display Value for Dropdown Item>")`
`## Repeat above 1 line for number of options which needs to be shown in the Dropdownlist ##`
`}]`
	Section 5. - Defining (mandatory set of different enum dimensions required in the asset
	
	Note: This is inserted into the global section of the template.
	
	`uniqueEnumFieldTypeName` – the unique enum definition that corresponds to a specific enum dimension field name in section 4.
	
	`uniqueEnumFieldItemName` – field item name for the dropdown list
	
	Repeat the Display Value Dropdown Item line for the number of options which need to be shown in the Drop-down list
	
	Repeat the block for each different enum dimensions definition required in the asset.

Device Class Definition graphqls Template reference

This section details the device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template reference file. This first section is the template file followed by a detailed description of the sections of the file.

```
## | Rule - Values which has | in the middle are the choices that user has by which only one choice can be
chosen. Remove all other
 choices including the | symbol(s)##
## <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values. Consider |
rule explained above a
s well. Remove characters < and > as well##
## [] Rule - Values surrounded by [] are optional definitions which either needs to be completely removed or
to be used exactly as d
efined. Consider <> rule explained above as well. Remove characters [ and ] as well ##

[#<Comments for type of asset>]
type <AssetType> implements Asset & Device @asset(
    version: <AssetTypeVersion>
    label: "<Label for Asset Class>"
) {

    ## Repeat below 7 lines for numbers of dimensions to be defined apart from ones defined in Asset or De-
vice class definitions ##
    [#<Comment for Dimension Field>]
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
    <uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@dsaMapping(path: "<DSA path expres-
sion>")] @dimension(
        label: "<Display Label of Field>"
        [unique: <true | false>]
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )

    ## Repeat below 6 lines for numbers of metrics to be defined apart from ones defined in Asset and Device
class definitions ##
    [#<Comment for Metric Field>]
    <uniqueFieldName>: <String | Int | Float | Long | Boolean> [@dsaMapping(path: "<DSA path expression>")]
@metric(
        label: "<Display Label of Field>"
        [unitName: "<Metric unit name>"]
        [unitSymbol: "<Metric unit symbol>"]
    )

    ## Repeat below 7 lines for numbers of dimensions to be renamed##
    [#<Comment for Dimension Field>]
    <uniqueFieldNameNew>: <String | Int | Float | Long | Boolean>[!] @rename(oldName: "<uniqueFieldNameOld>")
[@dsaMapping(path: "<D
SA path expression>")] @dimension(
        label: "<Display Label of Field>"
        [unique: <true | false>]
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )

    ## Repeat below 6 lines for numbers of metrics to be renamed##
    [#<Comment for Metric Field>]
    <uniqueFieldNameNew>: <String | Int | Float | Long | Boolean> @rename(oldName: "<uniqueFieldNameOld>")
[@dsaMapping(path: "<DSA
path expression>")] @metric(
        label: "<Display Label of Field>"
        [unitName: "<Metric unit name>"]
        [unitSymbol: "<Metric unit symbol>"]
    )

    ## Repeat below 7 lines each of dimension where Default Value needs to be set when changing a nullable to
non nullable dimension
##
    [#<Comment for Dimension Field>]
    <uniqueFieldName>: <String | Int | Float | Long | Boolean>! @defaultValue(value: "<String representation
of default value>") [@d
saMapping(path: "<DSA path expression>")] @dimension(
        label: "<Display Label of Field>"
        [unique: <true | false>]
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )

    ## Repeat below 6 lines for numbers of enum dimensions required in the asset. The Type <asset-
Type>_<uniqueEnumFieldTypeName> mus
t match between the enum and this field definition.##
    [#<Comment for Dimension Field>]
    <uniqueFieldName>: <assetType>_<uniqueEnumFieldTypeName> [@dsaMapping(path: "<DSA path expression>")]
@dimension(
        label: "<Display Label of Field>"
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )
}

## Repeat below 4 lines for number of different enum dimensions required in the asset##
[enum <assetType>_<uniqueEnumFieldTypeName> {
    <uniqueEnumFieldItemName> @enumValue(label: "<Display Value for Dropdown Item>")
    ## Repeat above 1 line for number of options which needs to be shown in the Dropdownlist ##
}]
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
## The optional directive @dsaMapping will be defaulted to $<uniqueFieldName> if whole @dsaMapping directive
is omitted ##
## The optional argument unique will be defaulted to false if whole argument is omitted ##
## The optional argument searchable will be defaulted to false if whole argument is omitted ##
## The optional argument access will be defaulted to RW if whole argument is omitted ##
## The optional argument unitName will be defaulted to "", if whole argument is omitted ##
## The optional argument unitSymbol will be defaulted to "", if whole argument is omitted ##
## The type name defined in enum <assetType>_<uniqueEnumFieldDefinition> must be unique across all available
asset definitions. Henc
e <assetType> is also added in the type name ##
```

Note: The following definitions must match the corresponding values from the Device Definition file to the Asset Class.

device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template	**device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template**
"assetType": "**<Name for Asset Type>**", "assetTypeVersion": **<Version Number for Asset Type>**,	type **<Name for Asset Type>** implements Asset & Device @asset(version: **<Version Number for Asset Type>**
"dataflow": { **"<uniqueFieldName>":**	**<uniqueFieldName>:** <String | Int | Float | Long | Boolean> [@dsaMapping(path: "<DSA path expression>")] @metric(label: "<Display Label of Field>" [unitName: "<Metric unit name>"] [unitSymbol: "<Metric unit symbol>"])

"dimensionMapping": [
{
"type": "DSA",
"targetPath": "<Source DSA Path to read dimension>",
"fieldName": "<**uniqueFieldName**>"
}
	<uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@rename(oldName: "<uniqueFieldNameOld>")] [@defaultValue(value: "<String representation of default value>")] [@dsaMapping(path: "<DSA path expression>")] @dimension(
	label: "<Display Label of Field>"
	[unique: <true | false>]
	[searchable: <true | false>]
	[access: "<R | RW>"]
	)

Onboarding a Message Broker as a device example

This example describes the use of a set of defined files that discover message brokers and onboard them as devices. Throughout the example we will make modifications in incremental steps to the files and highlight them we go for clarity. The following steps will be shown:

- Modifying a field
- Adding a field
- Renaming a field
- Removing a field
- Making a field non-nullable

© 2018 Cisco and/or its affiliates. All rights reserved.

Let us create two files under a new directory broker-link in /opt/cisco/kinetic/efm_manager/config/discovery/ and place the files device_discovery_definitions_BrokerDevice_v1.json and device_class_definitions_BrokerDevice_v1.graphqls respectively as follows. See the tables below.

And we will use the following file content as version 1. Note that even if one of the files is not updated, the version numbers must be in sync on both definition files and therefore incremented.

device_discovery_definitions_BrokerDevice_v1.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 1,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter $is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Usage/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

© 2018 Cisco and/or its affiliates. All rights reserved.

device_class_definitions_BrokerDevice_v1.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @asset(
    version: 1
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Distribution/:value") @dimension(
        label: "Distribution"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path: "/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

The files will be show for before and after with the changes highlighted in red.

- **Modifying a field**
    - Changing the display name of a dimension distribution from " Distribution" to " Distribution Name"
    - Changing the unit Symbol of metric data_in from " bytes" to " B"

device_discovery_definitions_BrokerDevice_v1.json	device_discovery_definitions_BrokerDevice_v2.json

© 2018 Cisco and/or its affiliates. All rights reserved.

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 1,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v1.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 2,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v2.graphqls

© 2018 Cisco and/or its affiliates. All rights reserved.

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 1
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 2
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "B"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
Starting Ignite
Acquiring EFM Manager startup lock
Reading configuration
Loaded 14 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v1 to v2'
[SUCCESS - processed 1 object(s) in 316 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

© 2018 Cisco and/or its affiliates. All rights reserved.

- **Adding a field**
    - Adding a new dimension field hostname which will be read from the source link and must be a searchable field.
    - Adding a new dimension field contact_person, which will be given as input by the user and must be searchable as well.
    - Adding a new Metric field data_out which will be read from the source link.

device_discovery_definitions_BrokerDevice_v2.json	device_discovery_definitions_BrokerDevice_v3.json

© 2018 Cisco and/or its affiliates. All rights reserved.

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 2,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }




    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }







    }
  }
]
```

device_class_definitions_BrokerDevice_v2.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 3,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v3.graphqls

© 2018 Cisco and/or its affiliates. All rights reserved.

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 2
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 3
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo systemctl efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
************************************************************************************************
Schema migration command started
************************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
------------------------------------------------------------------------------------------------
Starting Ignite
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
Acquiring EFM Manager startup lock
Reading configuration
Loaded 16 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v2 to v3'
[SUCCESS - processed 1 object(s) in 493 ms]
Done

[root@centos7 efm_manager]# sudo systemctl efm-manager start
```

- **Renaming a field**
    - Renaming the dimension field distribution to distribution_name
    - Renaming the metric filed cpu to be called as cpu_usage

device_discovery_definitions_BrokerDevice_v3.json	device_discovery_definitions_BrokerDevice_v4.json

© 2018 Cisco and/or its affiliates. All rights reserved.

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 3,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v3.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 4,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution_name"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v4.graphqls

© 2018 Cisco and/or its affiliates. All rights reserved.

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 3
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 4
    label: "Broker Device"
) {

    distribution_name: String @rename(oldName:
"distribution") @dsaMapping(path: "/Distribu-
tion/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @rename(oldName: "cpu")
@dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo systemctl efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
Starting Ignite
Acquiring EFM Manager startup lock
Reading configuration
Loaded 18 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v3 to v4'
[SUCCESS - processed 1 object(s) in 558 ms]
Done

[root@centos7 efm_manager]# sudo systemctl efm-manager start
```

- **Removing a field**
    - Remove the dimension field distribution_name
    - Remove the metric field data_in

device_discovery_definitions_BrokerDevice_v4.json	device_discovery_definitions_BrokerDevice_v5.json

© 2018 Cisco and/or its affiliates. All rights reserved.

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 4,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution_name"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v4.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 5,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v5.graphqls

© 2018 Cisco and/or its affiliates. All rights reserved.

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 4
    label: "Broker Device"
) {

    distribution_name: String @rename(oldName:
"distribution") @dsaMapping(path: "/Distribu-
tion/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @rename(oldName: "cpu")
@dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 5
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Note: The @rename directive must be removed on the next version of changes on the graphqls file.

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
Starting Ignite
Acquiring EFM Manager startup lock
Reading configuration
Loaded 20 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v4 to v5'
[SUCCESS - processed 1 object(s) in 532 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

- **Making a field non-nullable**

device_discovery_definitions_BrokerDevice_v5.json	device_discovery_definitions_BrokerDevice_v6.json

© 2018 Cisco and/or its affiliates. All rights reserved.

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 5,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v5.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 6,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "sublist brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v6.graphqls

© 2018 Cisco and/or its affiliates. All rights reserved.

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 5
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )









    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 6
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String! @defaultValue(value:
"Joe") @dsaMapping(path: "$contactPerson") @dimen-
sion(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )


    contact_method: String! @defaultValue(value:
"Telephone") @dsaMapping(path: "$contactMethod")
@dimension(
        label: "Contact Method"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo systemctl stop efm-manager

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
Starting Ignite
Acquiring EFM Manager startup lock
Reading configuration
```

© 2018 Cisco and/or its affiliates. All rights reserved.

```
Loaded 22 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v5 to v6'
[SUCCESS - processed 1 object(s) in 515 ms]
Done

[root@centos7 efm_manager]# sudo systemctl start efm-manager
```

Restarting EFM Manager after adding or modifying Device Definition files

As a reminder, for all changes to onboarding configuration files require restarting the EFM Manager for them to be read as well as running the load-cofngurations.sh script before restarting. This is accomplished with the following steps:

1. Stop EFM Manager:
   `sudo systemctl stop efm-manager`
2. Run load configuration script.
   `sudo $EFM_ROOT/efm_manager/bin/load-configurations.sh`
   Look for the SUCCESS Message
3. Start EFM Manager:
   `sudo systemctl start efm-manager`

© 2018 Cisco and/or its affiliates. All rights reserved.

Onboarding FAQ

Q: Should I stop the efm manager before I start to add new definitions files?

No. It is not required to stop the efm manager to add new configuration files to file system. But for the new files to be read by efm manager, the following three steps are required:

1. Stop efm-manager
2. Run load_configuration.sh script
3. Start efm-manager

Q: Do I need to do the above steps for every new version of the definition file?

Yes. the above mentioned three steps are must to followed for the new definitions to take effect.

Q. What if a dimension is defined in json and not defined in graphqls?

This field will not be shown in UI and when trying to save the device, an error will be generated that the field is missing.

Q. What if a dimension is defined in graphqls and not defined in json?

This field will be shown in UI as new field and this will be a new dimension for the device.

Q. What if a metric is defined in json and not defined in graphqls?

This metric will not be carried over to asset DSLink and there will not be any error specific to that.

Q. What if a metric is defined in graphqls and not defined in json?

This metric will be displayed in asset DSLink but will be empty.

Q: Should a field name be unique in the same asset?

Yes. The field name must be unique within a defined asset.

Q. Should a field name be unique across different version of same asset?

The field name must be unique considering all the versions of same asset.

Q. Should the field name be unique across different types of asset?

No. Each of the assets can have its own dimension and they could be used in whichever way required for the specific asset.

Q. Should the enum fieldtype name be unique across different assets?

YES. This is very important to have different fieldtype names for enums across different assets.

Q. Can a comment be added anywhere in the json file?

© 2018 Cisco and/or its affiliates. All rights reserved.

No. The json file cannot contain any comments.

Q. Can a comment be added to graphql file? Can it be added anywhere?

Yes. The comment is recommended to be added just before the start of a field definition or a type definition. But graphql file will allow comments to be added anywhere. The comment line starts with #.

Q. Can I read the same value from source as dimension and metric, with two different field names and use it in the graphqls file.

Yes. it is possible.

Q. Will it break if any dimension related argument is used in metric and vice versa?

No. It will not break anything. Those additional arguments are ignored.

Q. Where can I place the config files for on-boarding?

{EFM_INSTALLATION_PATH}/efm_manager/config/discovery/

Q. Will it break if I have older versions of the asset definition files in the file system?

No. It will not break anything and in fact its recommended to have the older versions of file as well, so its easily understandable for the user to see changes between versions.

Q. Why two files are required for on-boarding a new device type?

There are two files which drives the on-boarding of a device type.

- A Json file, which has details on the dimensions to read from source dslink, a field name for it, the source DQL Query to discover the device, default values for any fields and path from where the metric needs to be read from the source.
- A corresponding graphqls, file which will have details about the way its stored in the asset DSLink and how it will be shown in EFM manager UI as well.

Q. What do I need the serial field for?

With EFM 1.6, the serial field is the primary key. It is used to uniquely identify the device.

Q. Can I leave the @rename directive in the next version of changes on graphqls file?

No. The @rename directive must be removed on the next version of changes on the graphqls file.

Q. Do I need to have @defaultvalue directive for every non nullable(mandatory) dimension?

Yes, if it is not the first version of the definition file. In case of changed version of file for migration, this directive is mandatory, as this will be used in migrating the already accepted devices from older version to newer version.

Q. What are the specific details of the each of the field, argument, type?

Refer to the documentation help for exact definition, syntax template etc.,

Q: What if the schema definition files are duplicated with same content?

© 2018 Cisco and/or its affiliates. All rights reserved.

Nothing will be affected, as the migration is triggered only based on the assettype and version number changes.

Q: What if in the config files, the contents were changed, but the version number is not increased?

Nothing will be affected, as the migration is triggered only based on the assettype and version number changes.

Q: What if in the json config file, the version is increased and not in graphqls file?

The load configuration script will error out expecting the similar version file of graphqls.

Q: What if in the graphqls config file, the version is increased and not in json file?

The load configuration script will error out expecting the similar version file of json.

Q. Do I have to stick to the file naming convention?

The file naming is not important as long as the extension are json and graphqls for the two files. The recommendation is to have name like below where assetType and version numbers are mentioned.

- device_discovery_definitions_((assetType))_v((assetTypeNumber)).json
- device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls

Q: Can I change the data type of a field in a new version?

No. Changing the data type of a field is not supported.

© 2018 Cisco and/or its affiliates. All rights reserved.

Device Simulator Link

The sensor-simulation DSLink is an optional component of the EFM Manager. It is used for testing and demonstration purposes and generates virtual devices that are published to the message broker. Since the sensor-simulator devices are defined using the Cisco EFM Device Object Model Standard structure, they will be detected properly by the EFM Manager.

The System Administrator can decide to disable or remove the device-simulator DSLink if desired.

By default, the device simulator performs the following:

- Generates 10 virtual devices, 5 temperature and 5 vibration devices
- The device update interval is every 60000 ms (milliseconds)
- A sinus finishing time of 15 minutes

The metrics for the device-simulator DSLink can be modified by the System Administrator tool or in the dataflow Editor under the device-simulator DSLink, but not in the EFM Manager. The following metrics are user definable and can be set to new values:

- Simulated Temperature Device Count
- Simulated Vibration Device Count
- Sinus Finishing Time (in minutes)

DSLink Input Definitions for generating devices

This DSLink uses the following as input in generating the device simulation. They can be modified using the Dataflow Editor or the System Administrator for the device-simulator DSLink.

- Device Update Interval – This defines the interval in which the device reading has to be updated. Default – 60000 msec.
- Simulated Temperature Device Count – This defines the number of temperature devices that needs to be simulated. Default – 5
- Simulated Vibration Device Count – This defines the number of Vibration devices that needs to be simulated. Default – 5
- Sinus Finishing Time: This defines the time required for the sinus curve to complete. Default – 15 minutes.

© 2018 Cisco and/or its affiliates. All rights reserved.

Device Simulation Details

Simulated Temperature Device:

According to the number specified in Simulated Temperature Device Count, the Devices are labelled. By default, the labels are Temperature Device 1, Temperature Device 2 thru Temperature Device 5.

On changing the “Simulated Temperature Device Count” value, those specific number of NEW devices are simulated. For example, if it set to 20, then Temperature Device 1 thru Temperature Device 20 will be generated.

Each Device will have Temperature value simulated from 18 to 22 in a sinus curve. The value is represented as °C.

Simulated Vibration Device

According to the number specified in Simulated Vibration Device Count, the Devices are labelled. By default, the labels are Vibration Device 1, Vibration Device 2 thru Vibration Device 5.

On changing the “Simulated Vibration Device Count” value, those specific number of NEW devices are simulated. For example, if it set to 20, then Vibration Device 1 thru Vibration Device 20 would be generated.

Each device will contain X Axis, Y Axis and Z Axis simulated values for Acceleration, Distance and Frequency.

Acceleration is defined as a value simulated from -1 to +1 in a sinus curve. The value is represented in $\frac{mm}{s^2}$.

Distance is defined as values between 0.7 and 3.7 (with stronger vibration) and between 0 and 0.2. The value is represented in mm.

Frequency – Upper values between 10 and 20 (with stronger vibration) and between 1 and 5. The value is represented in hertz.

© 2018 Cisco and/or its affiliates. All rights reserved.

Obtaining documentation and submitting a service request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

© 2018 Cisco and/or its affiliates. All rights reserved.