LNI 4.0 Testbed Edge Configuration – Implementation View

[LNI 4.0 Testbed Edge Configuration – Implementation View 1](#_Toc118536031)

[1 Overall Objectives of Testbed Edge Configuration 2](#_Toc118536032)

[2 Purpose of this Document 2](#_Toc118536033)

[2.1 Target audience of this document 3](#_Toc118536034)

[3 Implementation View on Edge Configuration 3](#_Toc118536035)

[3.1 Structure of the document 3](#_Toc118536036)

[3.2 Methodological Approach 3](#_Toc118536037)

[3.3 Versioning of the API 3](#_Toc118536038)

[3.4 API addressing 4](#_Toc118536039)

[3.5 Endpoint Primitives 4](#_Toc118536040)

[3.6 Security, Identity and Rights Management 5](#_Toc118536041)

[3.7 Asset Adressing (B.Fiebiger, M.Rentschler, M.Schmierer, K.Walther) 5](#_Toc118536042)

[3.8 Asset Datamodel (M.Rentschler, M.Schmierer, B.Fiebiger, K.Walther) 6](#_Toc118536043)

[3.8.1 DeviceClass 6](#_Toc118536044)

[3.8.2 Manufacturer 6](#_Toc118536045)

[3.8.3 ManufacturerUri 6](#_Toc118536046)

[3.8.4 Model 6](#_Toc118536047)

[3.8.5 ProductCode 6](#_Toc118536048)

[3.8.6 SerialNumber 6](#_Toc118536049)

[3.8.7 ProductInstanceUri 7](#_Toc118536050)

[3.8.8 HardwareRevision 7](#_Toc118536051)

[3.8.9 SoftwareRevision 7](#_Toc118536052)

[3.8.10 DeviceRevision 7](#_Toc118536053)

[3.8.11 RevisionCounter 7](#_Toc118536054)

[3.8.12 DeviceManual 7](#_Toc118536055)

[3.8.13 Description 7](#_Toc118536056)

[3.8.14 AssetInterfaces 7](#_Toc118536057)

[3.9 Discovery and Directory Service (S.Höckner, B.Vojanec, M.Rentschler) 8](#_Toc118536058)

[3.9.1 Edge Asset Management API 8](#_Toc118536059)

[3.10 Parameter Handling (B.Vojanec, B.Fiebinger, Perinet) 9](#_Toc118536060)

[3.10.1 Capabilities Discovery 9](#_Toc118536061)

[3.10.2 Parameter Discovery 10](#_Toc118536062)

[3.10.3 Provisioning of Asset Parameters 10](#_Toc118536063)

[3.10.4 Propagation of Data Endpoints 12](#_Toc118536064)

[3.11 Application Handling (Hilscher ?) 12](#_Toc118536065)

[3.11.1 Provision of an Application 12](#_Toc118536066)

[3.11.2 Propagation of a provisioned or updated Application 12](#_Toc118536067)

[3.11.3 Deletion of a discontinued Application 13](#_Toc118536068)

[3.11.4 Deployment of an Application 13](#_Toc118536069)

[3.11.5 Obtaining an Application or an Update of an Application 14](#_Toc118536070)

[3.11.6 Selection of an Edge Runtime 14](#_Toc118536071)

[3.11.7 Provision of an Edge Runtime 14](#_Toc118536072)

[3.11.8 Deploying an Edge Runtime 14](#_Toc118536073)

[3.12 Application Runtime Functions (M.Schmierer, V.Volevach) 15](#_Toc118536074)

[3.12.1 Precondition Checks 15](#_Toc118536075)

[3.12.2 Service Registration 15](#_Toc118536076)

[3.13 Monitoring Functions 15](#_Toc118536077)

[3.13.1 Asset Management Functions 15](#_Toc118536078)

[3.13.2 Integrity Monitoring Functions 15](#_Toc118536079)

[3.14 Non-Functional Requirements 15](#_Toc118536080)

[4 Testcase Definition 16](#_Toc118536081)

[5 References 16](#_Toc118536082)

[6 Authors and Contributors 16](#_Toc118536083)

[6.1 Authors 16](#_Toc118536084)

[6.2 Contributors 16](#_Toc118536085)

**Contact:**

Labs Network Industrie 4.0 e.V.

Ernst-Reuter-Platz 7

10587 Berlin, Germany

Phone: +49 (30) 36702177

E-Mail: [info@lni40.de](mailto:info@lni40.de)

# Overall Objectives of Testbed Edge Configuration

The testbed Edge Configuration was established to prepare the standardization in the context of the manufacturing industry with respect to the emerging edge computing technology. The testbed does not address edge computing technology in itself but focuses on *edge configuration*. For this purpose, concepts will be developed, practically implemented and validated. The results and experiences will be made available to the standardization activities to feed them into the further or new development of standards.

From an architectural point of view, the testbed Edge Configuration is based on a layered architecture as shown in Figure 1:

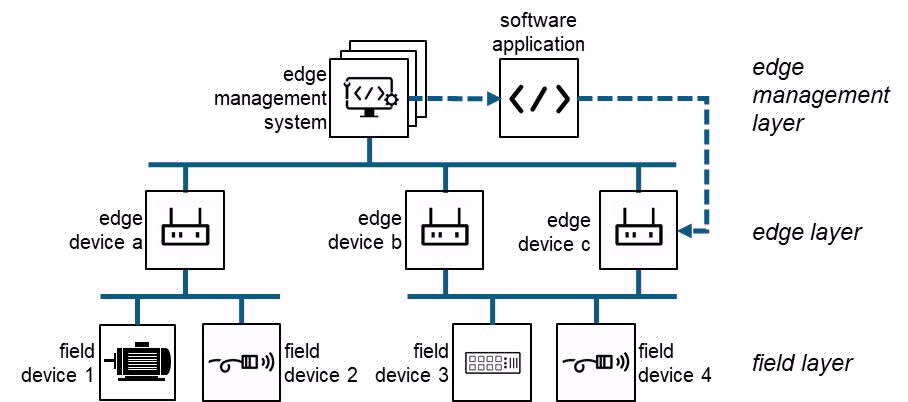


Figure 1. Layered architecture of the proposed testbed Edge Configuration

* The focus of the testbed Edge Configuration is on the configuration of the interaction between the field, edge and edge management layer. Currently there does not exist a suitable standard for this focus and the testbed will develop proposals for this aspect. Based on the functional primitives including parameter sets in the functional view, this document (implementation view) will provide a framework for formal REST-API interface specifications that can be directly implemented on IT relevant protocols like HTTP, MQTT or OPC UA. This formal specification shall be provided in form of JSON-based data models, objects and semantics.

# Purpose of this Document

The purpose of this document is to describe in accordance with the Industrial Internet Reference Architecture [1] a so-called *implementation view* of the testbed Edge Configuration (see Figure 2) and possibly derived standardization activities.

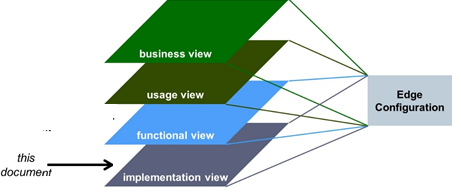


Figure 2. Classification of the document following the Industrial Internet Reference Architecture, see [1]

Regardless of the “stable” application context (see *usage view* document), various concepts (in the terminology in Figure 2 called a functional view) and solution approaches (in the terminology in Figure 2 called an implementation view) will be developed and discussed..

## Target audience of this document

Since the focus of this document is to provide actual implementation guidelines for an edge configuration system, its main target audience is system architects and developers trying to specify a concrete edge configuration system.

# Implementation View on Edge Configuration

## Structure of the document

The following chapters provide categories for implementation specifications to cover the functional requirements identified in given in the Functional View document [4]. Based on the functional primitives in the functional view document [4], the following chapters will provide formal interface specifications for procedure- and event-based interactions between the architectural components that can be directly implemented on suitable IT relevant protocols like REST-API, MQTT or OPC UA. The formal specification shall be provided in form of JSON-based data models, objects and semantics.

## Methodological Approach

In order to derive implementation specifications, the functional requirements identified in the Functional View document [4] are systematically analyzed and formal interface specifications for procedure- and event-based interactions between the architectural components provided that can be directly implemented on suitable IT relevant protocols like HTTP, MQTT or OPC UA.

The formal specification shall be provided in form of JSON-based REST-API data models, objects and semantics. The definitions shall provide a standardized outer frame, thus only the URI addressing scheme, the basic access endpoints and some top-level structure definitions for the response data objects shall be defined and recommended. Detailed variable definitions in the provided examples are just for demonstration purposes and not intended to be binding for implementation.

To reduce the number of REST-API definitions, Field and Edge Devices are considered as a single class of devices whenever possible. For example, in the case of device parameter retrieval, the interface for a Field Device is identical to that of an Edge Device.

To cross-check against the functional view definitions, the sequence diagrams created there are very helpful. For every activity that is started for an object instance in the sequence diagrams, a corresponding interface description must be available in this implementation view.

## Versioning of the API

In order to avoid compatibility issues in the future, encoding version information into the URLs addressing the API endpoints shall be avoided. Instead a specific base endpoint shall return the version information of the current implementation.

## API addressing

A REST API defines interaction endpoints that form URIs which act as the primary addresses for the interactions. These Endpoint-URIs are unique link adresses to access certain functionalities in the target system. The URIs consist in fact of an **address part** and a **functionality related part**. To design a longterm stable and maintainable API for a heterogeneous system landscape with different access paths (hierarchy depths), it is highly recommended to conceptually separate these aspects as good as possible.

A REST-API URI can be conceptually divided into the following elements:

**{scheme}://{gatewayaddress}/{substructure}/{endpoint}**

**{scheme}** determines the access protocol (http or https)

**{gatewayaddress**} determines the ip address or dns name of the access gateway that hosts the REST-API.

**{substructure}** is optional and only used when a hierarchical substructure below the backend (access gateway) needs to be addressed, e.g. internal submodules, ports or connected devices.

**{endpoint}** determines the actual function call on the addressed element and may contain additional query parameters.

To achieve the above mentioned properties like longterm stability and maintainability, it is recommended to strictly separate the **{substructure}** definitions from the **{endpoint}** definitions, keeping the overall number of endpoint definitions low, also the generic **{endpoint}** definitions should be kept as generic as possible and to a minimum.

The following principles are recommended for construction of the **{substructure}** part:

|  |  |
| --- | --- |
| **Addressable substructure element** | **{substructure} representation** |
| Internal subelement with a unique alias. This can be used to address e.g. ports, software modules etc. | **sub/{alias}** |
| External subelement that is connected via one intermediate subelement. | **sub/{alias}/sub/{alias}** |
| External subelement that is connected via two intermediate subelements. | **sub/{alias}/sub/{alias}/sub/{alias}** |
| External connected device with a unique deviceAlias, independent from the hierarchical position of the device. | **devices/{deviceAlias}** |

Typical content for an {alias} migh be a port number (e.g. “1”), a port descriptor (e.g. “XF1”) or the Id of a software module running within the backend or accessible by the backend.

## Endpoint Primitives

The following basic endpoint primitives should be sufficient to handle all kind of addressable system parts:

|  |  |
| --- | --- |
| REST API endpoint | Function |
| **/capabilities** | GET available subelement and capability information of the adressed system. If present, the response object contains a list of **{substructure}** elements or complete URIs, identifying the actual addresses of the endpoint’s subelements, asset interfaces and capability indicators. |
| /**configuration** | GET/POST configuration data of the of the adressed system. The JSON objects are divided into functional blocks that can be selectively accessed. |
| /**diagnostics** | GET diagnostic information (e.g. system states, logs) of the adressed system part. |
| /**processdata** | GET/POST process data values of the adressed system part. |

Note that not all of these endpoint primitives must be supported by all adressable system parts.

## Security, Identity and Rights Management

Role based (Administrator, Operator, Auditor, derived from IEC 62443?) access management supporting User/Password and Certificates (no rights via Anonymus access)

Rights Management is necessary to allow Configuration functions that access data or parameters considered worthy of protection from unauthorized access. These functions provide a basis for assigning and checking access rights

## Asset Adressing (B.Fiebiger, M.Rentschler, M.Schmierer, K.Walther)

Asset addressing is depending on the Discovery and Directory Service (3.7) and should be mapped to standardized identification properties.

Asset adressing for hardware assets as defined in AAS.

* Manufacturer (URI)
* DeviceClass [EdgeApp, EdgeDevice, FieldDevice] – maybe to add at higher level)
* Model/Type
* Serialnumber/InstanceId

Needed information to work and configure (e.g. request of AssetAdressing information as Anonymus as well? SecurityAspects?):

* IP Address
* Name (DNS, Profinet-Stationname,

AssetAdressing for software assets -> OI4.0 concepts? See 3.9 ApplicationHandling?

Identity and access management ensures appropriate access to resources across the system under consideration, utilizing authentication, authorization and role-based access rights.

TODO: Define user management and role based access right scheme. Define an API.

## Asset Datamodel (M.Rentschler, M.Schmierer, B.Fiebiger, K.Walther)

An Edge System can consist of a range of hard- and software entities (further called “assets”). To handle these different assets in a standardized way on the higher layers of the system, an Asset Datamodel according to the properties defined for the **VendorNameplate** Interface in OPC UA [Part 100-4.5.2](https://reference.opcfoundation.org/DI/docs/) is recommended to be used. It shall consist of the elements listed below to provide a human-readable and machine-interpretable interface description of an asset. Additionally it is recommended to adopt he concepts defined in the W3C recommendation “[WoT Thing Description](https://www.w3.org/TR/wot-thing-description/)”, which provides a unified way to describe the capabilities of an IoT device or service with its offered data model and functions, protocol usage, and further metadata.

### DeviceClass

The mandatory property **DeviceClass** of type string stores semantic references regarding the application domain roles of the asset. A semantic referencing framework should be employed to specify the purpose of the asset.

Recommended suitable default classes of assets in the context of Edge Management are [**EdgeApp, EdgeRuntime,** **EdgeDevice**, **FieldDevice**].

If a device does not supply a content for **DeviceClass** in this defined semantic namespace, the content should be configurable and presetted with default values by the EMS or Edge Device.

### Manufacturer

The mandatory property of type string provides the name of the manufacturer of an asset. Uniformity in the name representation should be a goal, as typically there are different ways of writing a company name regarding all legal appendices and so forth. In order to avoid shortcomings in identification due to this, the ManufacturerUri is also given.

### ManufacturerUri

The mandatory property **ManufacturerUri** shall provide a valid URI to uniquely identify the Manufacturer of the asset. It shall comply to RFC 3986 and must contain a domain name and may contain subdomains.

The **ManufacturerUri** given for this property shall also be used for the **ProductInstanceUri** of the asset.

### Model

The mandatory property **Model** of type string shall provide the name of the product that the asset information belongs to. Since model names often are not sufficient to unambiguously identify a product, the **ProductCode** shall also be given.

The given **ProductCode** may also be used for the **ProductInstanceUri** of the asset.

### ProductCode

The mandatory property **ProductCode** of type string has the purpose to unambiguously identify the product **Model** of an asset within the manufacturer’s schemes for the generation of product codes.

The given **Model** may also be used for the **ProductInstanceUri** of the asset.

### SerialNumber

The mandatory property **SerialNumber** of type string shall provide a unique identifier for a hardware instance of an asset. It is the manufacturer's responsibility to provide globally-unique serial numbers for their hardware product samples.

In case of a software-only asset, the **SerialNumber** should reflect the active instance of the software, i.e. in form of a GUID.

The given **SerialNumber** given shall also be used for the **ProductInstanceUri** of the asset.

### ProductInstanceUri

The mandatory property **ProductInstanceUri** shall contain an IEC 61406 conform URI. According to the OPC UA specification the ProductInstanceUri must not exceed 255 characters. It is recommended to built the URI according to the following scheme:

**<ManufacturerUri/Model/ProductCode/SerialNumber>**

In case of multiple active instances of software applications during system operation, the **ProductInstanceUri** shall reflect this via the **SerialNumber**.

### HardwareRevision

The mandatory property **HardwareRevision** of type string shall be given according to the manufacturer-internal revision number of the hardware.

A recommended best practice is to use a dot-separated four numbers schema. For any subsequent hardware revisions, this value shall be incremented in an appropriate way. A physical asset with undefined or unknown hardware revision shall set **HardwareRevision** to “undefined”.

​In case of a software-only asset, the **HardwareRevision** shall be set to an empty string to detect the differences between hardware and software only assets.

### SoftwareRevision

The mandatory property **SoftwareRevision** of type string shall provide the manufacturer-internal revision number of the software of the asset.

A recommended best practice is to use a dot-separated four numbers schema. For any subsequent software revisions, this value shall be incremented in an appropriate way. An asset with undefined or unknown software revision shall set **SoftwareRevision** to “undefined”.

In case of a pure mechanical without any software component, the **SoftwareRevision** should be set to an empty string.

### DeviceRevision

The mandatory property DeviceRevision is a conglomerate revision that manufacturers may use to distinguish certain shipment configurations of their devices. Typically, a device revision will contain a specific hardware revision and a specific software revision, where the latter may change during the lifecycle through updates.

### RevisionCounter

The mandatory property **RevisionCounter** of type Integer reflects changes to the configuration of the asset information by an increment of the revision counter by one.

### DeviceManual

The optional property **DeviceManual** shall contain an URL that points to a human-readable manual document for the asset..

### Description

The optional property **Description** of type localized text provides a human-readable description of the asset. This object is not defined in OPC UA Part 100, but in the W3C recommendation and might be useful for a user-friendly overview of the available assets.

### AssetInterfaces

The W3C recommendation “[WoT Thing Description](https://www.w3.org/TR/wot-thing-description/)” provides a unified way to describe the capabilities of an IoT device or service with its offered data model and functions, protocol usage and further metadata. See also <https://en.wikipedia.org/wiki/Thing_Description>. It defines 3 kinds of interaction affordances, named **Property**, **Action** and **Event**:

**Property** exposes the state of an asset. This state can then be retrieved (read) and optionally updated (write). Devices can also choose to make Properties observable by pushing the new state after a change.

**Action** allows to invoke a function of an asset, which manipulates state (e.g., toggling a lamp on or off) or triggers a process on the device (e.g., dim a lamp over time).

**Event** describes an event source, which asynchronously pushes event data to the subscribers of the event (e.g., overheating alerts).

The W3C recommendation “[WoT Thing Description](https://www.w3.org/TR/wot-thing-description/)” defines also a **security** scheme vocabulary that can be used to describe the security capabilities of the asset.

To expose the interfaces offered by an asset it is recommended to adopt these definitions of the W3C as optional properties to be used within the edge management asset datamodel

## Discovery and Directory Service (S.Höckner, B.Vojanec, M.Rentschler)

Industrie 4.0 environments require the ability to discover available devices and services. The directory or registry service stores IDs, network addresses and protocol types of each participant. Entities in an Industry 4.0 environment that provide Asset Administration Shells may register them including their submodels and enable other participants to find them.

Entity Discovery (see FuncView Chapter 4.1.1) deals with the identification of hitherto unknown or unacknowledged entities inside the system under consideration. The process for discovery is expected to be either automatic or semi-automatic as to ensure ease of use. Information gained via discovery functions is critical for any form of automated onboarding process to make detected entities addressable inside the system. Entities can be devices, applications or services provided by them.

**Object**: Discovery Service

**Subject**: Field Device, Edge Device, Application, Edge Management System

The system under consideration shall provide a Discovery Service that registers entities in a network and provides access to that information via a standardized interface. The protocol mechanisms this service uses towards the entities is scenario related and left to the implementor. Most likely a range of legacy mechanisms like ARP, DHCP, ProfinetDCP, LLDP etc. needs to be supported for device detection. It is however recommended to implementors of Field Devices and discovery mechanisms to utilize the Zeroconf-Protocolsuite, consisting of AutoIP, mDNS and DNS-SD as standard mechanisms.

See also OPC 10000-21: Device Provisioning.

The Discovery Service can reside in any Edge Device or Edge Management System and provides access to any Edge Device or Edge Management System via the following functionalities:

1. Get list of all discovered assets of type [device/service/application]
2. Get list of newly discovered assets of type [device/service/application] since [absolut time]
3. Register for notification when a new asset of type [device/service/application] is discovered

### Edge Asset Management API

Any Edge Device or Edge Management System shall have access to the Discovery and Directory Service within an Edge Device via the following API commands:

|  |  |
| --- | --- |
| REST API Endpoint | Function |
| GET /assets | retrieve current asset directory content |
| POST /asset/notification | register a notification |
| DELETE /asset/{assetID} | unregister an asset |
| … | … |

This covers also FuncView Ch 4.2.3.1 “Propagation of a Detected Field Device”,

## Parameter Handling (B.Vojanec, B.Fiebinger, Perinet)

The Discovery of Parameters (see FuncView Chapter 4.1.2) encompasses those functions that deal with the retrieval of available and currently set parameters for the entities in the system under consideration. These functions are supportive of the configuration functions discussed below insofar as they provide starting information for any kind of configuration process that would edit the parameters of an entity.

It is recommended to transfer configuration parameters of a device via a standardized configuration file format, utilizing AutomationML.

### Capabilities Discovery

This function recovers structural and functional capability information (i.e. asset interfaces) from the target device (edge or field) needed for performing its expected duties in the automation process.

**Object**: Field Device, Edge Device

**Subject**: App on Edge Device, EMS

Context: Especially when a field device is being newly commissioned (see 4.1.1.1) to the overall automation system, the parameters it needs have to be retrieved in order to allow system integrators to set them (see 4.2.1.1).

Result: A set of capability information for an asset is available at the requesting instance.

Different types of target devices or assets often contain different functionalities and substructures, such as modules on a backplane or ports and device connected to them. To achieve a generic approach for deliveruing this structural information, the following concept shall be implemented:

The **GET/capabilities** endpoint is responsible for reading the necessary structural asset information from the backend. The returned JSON data object shall be structured in functional blocks, depending on the requirements of the backend implementation. The functional blocks shall be identifiable by locally unique URIs or names. The functional blocks can contain substructures (i.e.  port lists). If a substructure shall be directly addressable (i.e. a connected device), it shall also be identified by a URI according to the API addressing concept.

The **GET/capabilities** endpoint implementation shall allow to retrieve all existing functional blocks at once or only selected types of functional blocks (i.e. ports) by the use of a query parameter:

|  |  |
| --- | --- |
| REST API Endpoint | Function |
| GET capabilities | Get a detailed list of all available capabilities |
| GET capabilities?topicId=overview | Get an overview of the available capability types |
| GET capabilities?topicId={type} | Get detailed info on a single capability type |
| … | … |

Parameter attributes: name, datatype, size, volatility, accessRights (readOnly, readWrite, roleAccess)

…

### Parameter Discovery

This function recovers the set of parameters that the target device (edge or field) needs for performing its expected duties in the automation process.

**Object**: Field Device, Edge Device

**Subject**: App on Edge Device, EMS

Context: Especially when a field device is being newly commissioned (see 4.1.1.1) to the overall automation system, the parameters it needs have to be retrieved in order to allow system integrators to set them (see 4.2.1.1).

Result: A set of parameters for a field device is available at the edge device.

🡪 **already defined in AAS or JSON-IO-Link that can be adopted?:**

**<https://industrialdigitaltwin.org/wp-content/uploads/2021/09/09_details_of_the_asset_administration_shell_part2_V1_en_2020.pdf>**

|  |  |
| --- | --- |
| REST API Endpoint | Function |
| GET configuration | Get a detailed list of the available functional blocks and their parameters |
| GET configuration?topicId=overview | Get an overview of the available functional blocks |
| GET configuration?topicId={name} | Get detailed info of a single functional block |
| … | … |

Parameter attributes: name, datatype, size, volatility, accessRights (readOnly, readWrite, roleAccess)

…

### Provisioning of Asset Parameters

Either after a new asset (device or application) is connected or when the system integrator deems changes necessary for system-external reasons, the parameters that an asset receives for its operation can be set by the system integrator in the edge management system.

**Object**: Edge Management System, Edge Device, Field Device

**Subject**: System Integrator, Edge Management System

Context: This function is performed either when a new asset is connected or when the working parameters of an asset need to be changed during operation or maintenance. If an edge device receives parameters to be set on a field device, it has to propagate these parameter values to the device in question and make sure that they are incorporated (4.2.3.2).

Results:

1. The Object receives a new set of target parameters for an asset which it can propagate down to the asset through suitable communication functions.
2. The asset has changed its parameters to the transported values.

|  |  |
| --- | --- |
| REST API Endpoint | Function |
| POST parameter | Modify a set of or a single parameter |
| POST parameter/xyz | Modify a single parameter |
| POST parameter/activate | Set the modified parameters into active state |
| … | … |

### Propagation of Data Endpoints

This function allows a field device to propagate its contained data endpoints to a connected edge device for further processing.

TODO: Specify typical types of data endpoints: Process data variables? Parameter variables?

**Object**: Edge Device

**Subject**: Field Device

Context: Data endpoints are not defined in edge devices in an explicit manner. However, edge devices have to obtain these data endpoint so that a communication connection is established.

Result: The edge device receives the data to be exchanged.

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

## Application Handling (Hilscher ?)

Edge Management System to Application Store Communication handles all those communication relationships that exist between entities in the application store and entities on the Edge management layer. This kind of communication will usually encompass acyclic and event-based communication.

### Provision of an Application

The function provision of an application sets up the working parameters and the application environment.

**Object**: Application

**Subject**: Application developer

Context: Application must be tested before being provided into an application store of an edge management system.

Result: The application can be offered in the application store and can be bought, deployed, and executed on an edge device or IT infrastructure.

* An application certification scheme may be necessary.

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Propagation of a provisioned or updated Application

This function propagates information about a new application towards the edge management system.

Object: Edge Management System

Subject: Application Store

Context: A developer has provisioned/updated an application to the application store and would like to make them available for use in the edge management system.

Result: The edge management system lists the new/updated application.

* Push notification towards the EMS ?

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Deletion of a discontinued Application

This function propagates information about an application that has been discontinued towards the edge management system.

Object: Edge Management System

Subject: Application Store

Context: A developer has discontinued an application to the application store. The corresponding application is no longer available in the edge management system.

Result: The edge management system lists the updated application

* Push notification towards the EMS ?

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Deployment of an Application

This function provides the capability of deploying a selected application in an edge device using the edge management system. The deployment of an application also includes the configuration of data exchange and initial parameters.

Object: Application

Subject: Plant Operator

Context: This function can only be triggered if the target edge device supports the deployment of an application.

Result: The application can be executed with initial parameter on the edge runtime. The edge management system is notified after the selected applications has been successfully deployed and started.

* Installation routine on Edge Device ?

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Obtaining an Application or an Update of an Application

This function covers the act of obtaining an application’s update from the application store.

Object: Application itself and its update.

Subject: Edge Management System, Application Store

Context: A system integrator has selected an application for deployment or to be updated in an edge runtime and the edge management system has been supplied with the appropriate access right, for example after successful transactions, to obtain the application’s update from the application store.

Result: The edge management system has the installation binaries/files of the application or its update

* Installation routine on Edge Device ?

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Selection of an Edge Runtime

For an edge runtime to be installed on an edge device, it has to be selected from an application store.

Object: Edge Runtime

Subject: System Integrator

Context: The system integrator wants to commission an edge runtime to an edge device.

Result: An edge runtime is made available for download to an edge device.

### Provision of an Edge Runtime

When an edge runtime is selected to be downloaded to an edge device, it first needs to be made available to the edge management system that is to perform the deployment.

Object: Edge Management System

Subject: Application Store

Context: A system integrator has selected an edge runtime from an application store and now wants to deploy it to an edge device.

Result: The edge management system is set up for deploying the edge runtime to an edge device.

### Deploying an Edge Runtime

This function covers the deployment of an edge runtime to an edge device after it has been selected.

Object: Edge Device

Subject: Edge Management System

Context: A system integrator has selected an edge runtime for deployment to an edge device and the edge management system has been supplied with the appropriate access rights.

Result: The edge device runs the edge runtime.

## Application Runtime Functions (M.Schmierer, V.Volevach)

While some applications can be run almost independently of their surrounding infrastructure, many need accesses to specific resources provided by their host system. The provision of such access is the task of Application Runtime functions.

* Investigate Functional View

|  |  |
| --- | --- |
| REST API Endpoint | Function |
|  |  |
|  |  |
|  |  |
|  |  |

### Precondition Checks

For each function that is being executed, the preconditions according to the expected context given and according to any security policy have to checked. If any of the preconditions for executing the function are not met, a warning should be propagated through the system-inherent mechanisms. Execution should be prevented in order to avoid destabilizing conditions in the overall system.

### Service Registration

Upon startup, an application must register itself and the services it offers to the Discovery Service.

## Monitoring Functions

While configuration tasks aim at changing certain elements of a system under consideration, outside roles need means to identify the demand for a change of configuration in the first place. Monitoring functions allow the supervision of the workings of a system under consideration and provide alarms to any changes in the setup that require interaction by an affected outside role.

### Asset Management Functions

Asset management deals with the economic handling of systems with regards to the status of their entities. In the context of a system under consideration, Asset Management functions can be utilized to handle changes to entities that are covered by the use cases given in the Usage view document.

### Integrity Monitoring Functions

Not all changes to a system under consideration will always be planned events. Sometimes, malfunction, or unforeseen outside influence, can have an impact on the system. Integrity Monitoring functions allow outside roles who hold stakes in the system to detect any such unplanned events and to trigger proper event handling procedures.

## Non-Functional Requirements

The following requirements are mandatory attributes of each of the functions discussed in the other subsections of this chapter. Since they hold true for all functions, they are not explicitly mentioned in each function’s description.

# Testcase Definition

# References

[1] The Industrial Internet Reference Architecture Technical Report, [Link](https://www.iiconsortium.org/IIRA.htm)

[2] LNI Testbed Edge Configuration – Business View

[3] LNI Testbed Edge Configuration – Usage View

[4] LNI Testbed Edge Configuration – Functional View

[5] [IEC 61406](https://webstore.iec.ch/publication/67673) specifies “minimum requirements for a globally unique identification of physical objects which also constitutes a link to its related digital information."

[6] …

# Authors and Contributors

## Authors

Markus Rentschler, Murrelektronik GmbH

## Contributors