Horizon 2020
European Union Funding
for Research & Innovation

In2Rail

In2Rail

Project Title:	**INNOVATIVE INTELLIGENT RAIL**
Starting date:	01/05/2015
Duration in months:	36
Call (part) identifier:	H2020-MG-2014
Grant agreement no:	635900

Deliverable D8.7
Interface Control Document (ICD) for Application-specific Interfaces

Due date of deliverable	Month 27
Actual submission date	30-07-2017
Organization name of lead contractor for this deliverable	CAF
Dissemination level	PU
Revision	FINAL

Authors

		Details of contribution
Author(s)	**CAF Signalling (CAF)** Carlos Sicre Vara de Rey Manuel Castro Viñas	Coordination and Document structure of D8.7 Contributions to Sections 1-8 Review of Appendix 1
	Siemens (SIE) Stefan Wegele	Contributions to Sections 1-8 Appendix 1
Contributor(s)	**Ansaldo STS (ASTS)** Gian Luigi Zanella Matteo Pinasco	Contributions to Sections 1-8 Review of Appendix 1
	AZD Praha s.r.o. (AZD) Martin Bojda Michal Žemlička Martin Růžička	Contributions to Sections 1-8 Review of Appendix 1
	Bombardier Transportation (BT) Roland Kuhn Martin Karlsson Zbiewniew Dyksy	Contributions to Sections 1-8 Review of Appendix 1
	HaCon (HC) Sandra Kempf Rolf Gooßmann	Contributions to Sections 1-8 Review of Appendix 1
	Thales (THA) Jean-Yves Friant Jean-Jacques Rodot	Contributions to Sections 1-8 Review of Appendix 1

Executive Summary

The overall aim of the In2Rail project is to set the foundation for a resilient, cost-efficient, high capacity, and digitalised European Rail Network.

Intelligent Mobility Management (I^2M), a sub-project of I2R, is one of the three technical sub-projects and comprising Work Package 8 (WP8). WP8 addresses and develops a standardised integrated ICT environment capable of supporting diverse TMS dispatching services and operational systems. It also includes standard interfaces to external systems outside TMS/dispatching (for other railway management systems and transport modes) with a plug-and-play framework for TMS/dispatching applications.

WP8 represents the part of I²R lighthouse project to Shift2Rail IP2 and CCA which addresses works which are key inputs to S2R TD2.9 “Evolution of Traffic Management System” and CCA WA4.2 Integrated Mobility. All deliverables from WP8 will form the base for proceeding works in X2RAIL-2. WP6 “Traffic Management System” (IP2) and IMPACT-2 WP7 “Integrated Mobility” (CCA).

The current document corresponds to the seventh deliverable inside WP8, and is focused in the description of the required Data structure and Message Definition Syntax for the applications inside the TMS Application Framework related to their lifecycle.

The research has been conducted by all partners of WP8, and the inputs have been consolidated in this document.

TABLE OF CONTENTS

Abbreviations and Acronyms

Term	Description
AF	**A**pplication **F**ramework
AL	**A**pplication **L**ayer
API	**A**pplication **P**rogramming **I**nterface
CDM	**C**anonical **D**ata **M**odel
CENELEC	European Committee for Electrotechnical Standardization
EU	**E**uropean **U**nion
ICT	**I**nformation and **C**ommunication **T**echnologies
IF	**I**nter**f**ace
IL	**I**ntegration **L**ayer
I²M	**I**ntelligent **M**obility **M**anagement
I2R	**I**n**2R**ail
TMS	**T**raffic **M**anagement **S**ystem
TSR	**T**emporary **S**peed **R**estriction
UI	**U**ser **I**nterface
UML	**U**nified **M**odelling **L**anguage
UUID	**U**niversal **U**nique **ID**entifier
WP7	**W**ork **P**ackage **7**
WP8	**W**ork **P**ackage **8**
XML	e**X**tensible **M**arkup **L**anguage
XSD	**X**ML **S**chema **D**ocument

1 Objectives

WP8 constitutes one of the issues in the framework of the Project titled "Innovative Intelligent Rail" (Project Acronym: In2Rail; Grant Agreement No 635900).

The overall objective of WP8 is to address and develop a standardised integrated ICT environment capable of supporting diverse TMS dispatching services and operational systems. Additionally, WP8 deals with standard interfaces to external systems outside TMS/dispatching and with a plug-and-play framework for TMS/Dispatching applications.

The WP 8 includes two areas; the Integration Layer and the Application Framework for applications. Each area is devoted to specific subtopics, which are shown in Figure 1.1.

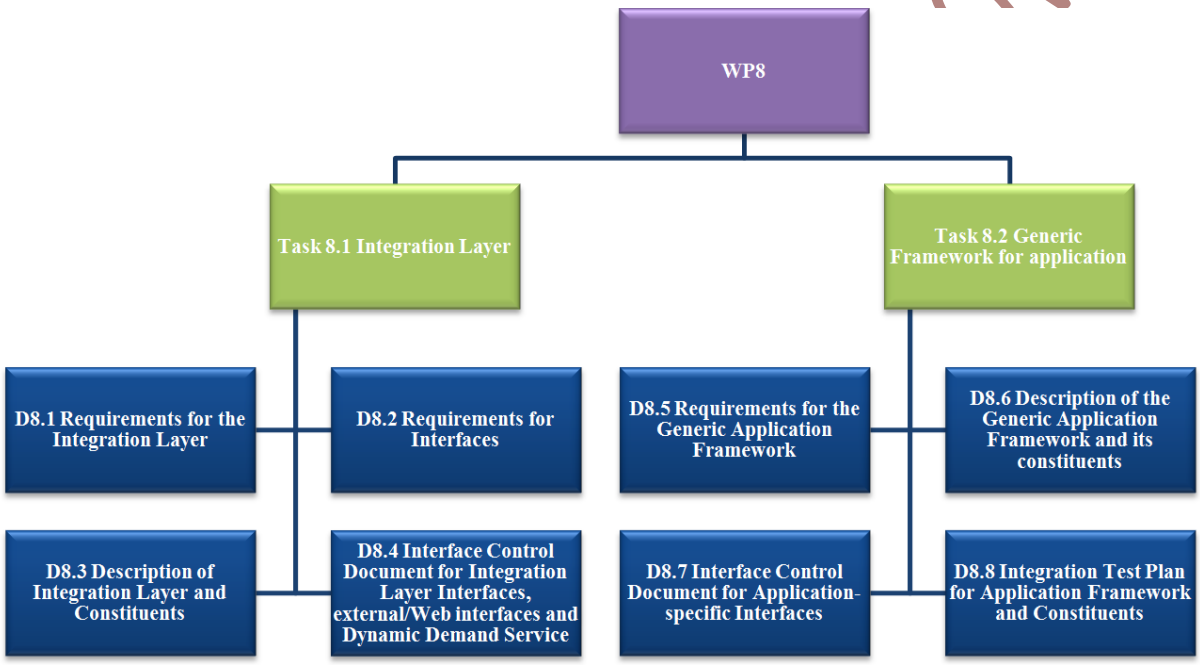

Figure 1.1: Subtopics of WP8

The Application Framework should comprise TMS core applications managing highly dynamic services and enable plug-and-play functionality (Figure 1.2).

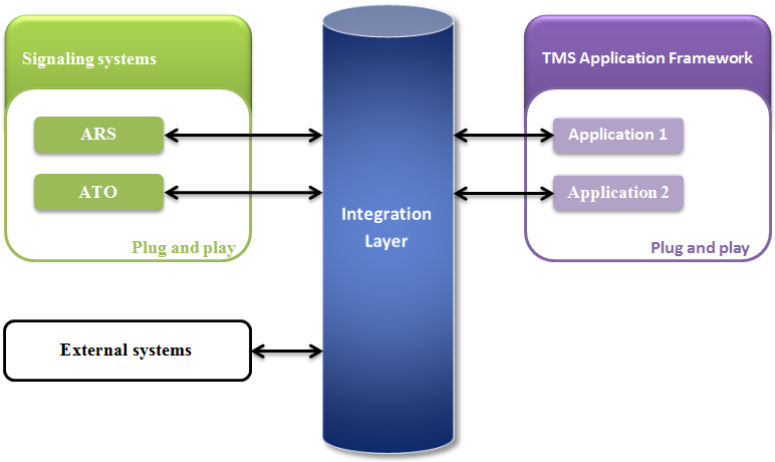

Figure 1.2: Overview of integration of TMS Application Framework

The long term objective for the project is to provide a standardised integrated ICT environment supporting TMS applications connected to other multimodal operational systems.

The objective of this Interface Control Document (ICD) is description how Plug-and-Play installation of the different business service applications in a framework can be ensured and hence avoiding complex and costly function and data mapping processes within the Interface structures.

Deliverable 8.7 is the first step towards standardized Interfaces in the Application Framework and will be followed from proceeding activities in X2RAIL-2, WP6 project of S2R including development of prototypes up to TRL6.

2 Background

The available products and systems for the Traffic Management Application available on the market from the various supply sources do not have standardized data structures and interfaces. This leads to enormous one-time efforts and cost to link sub-systems and products of different suppliers.

Cost savings linked to the reduction of these non-recurrent cost are considered to reach up to 10% of the total project cost if combinations of sub-systems would use a standardized ICT structure are applied within the overall system.

Therefore, the standardisation of Interfaces between different TMS business service applications is a key target for In2Rail and the preceding S2R activities.

In the frame of specifying and developing a new integrated and standardized ICT structure for Rail Operation services the standardization this deliverable is the first step towards the required Data structure and Message Definition Syntax for the Interfaces between applications inside the TMS Application Framework.

3 Purpose and structure of the document

The aim of this document is to provide a formal specification of the Data structures and Message Definition Syntax for the Applications inside TMS Application Framework related to their lifecycle.

This document together with [D8.6] shall enable fulfilment of the requirements specified in [D8.5]. The deliverable D8.6 provides functional description of the AF-modules with reasoning for architectural decisions.

The document is structured as follows:

- Chapter 4 describes the Integration Layer and Application Framework;
- Chapter 5 provides a description of the Application Framework constituents;
- Chapter 6 provides the message definition syntax;
- Chapter 7 focuses on the different Topics used by the Application Framework for communication;
- Chapter 8 consists of the conclusion of the document;
- Appendix 1 contains the definition of the data structures utilized.

As the Application Framework uses Integration Layer as communication platform, the implementation of the Application Framework services is only possible after issuing the D8.4 (ICD for Integration Layer).

4 Description of AF/IL

A general overview on IL/AF is provided in Section 1. In this Section both are described in further detail.

According to the current design activities in In2Rail, the communication platform for applications is provided by the Integration Layer, as it is shown in Figure 4.1.

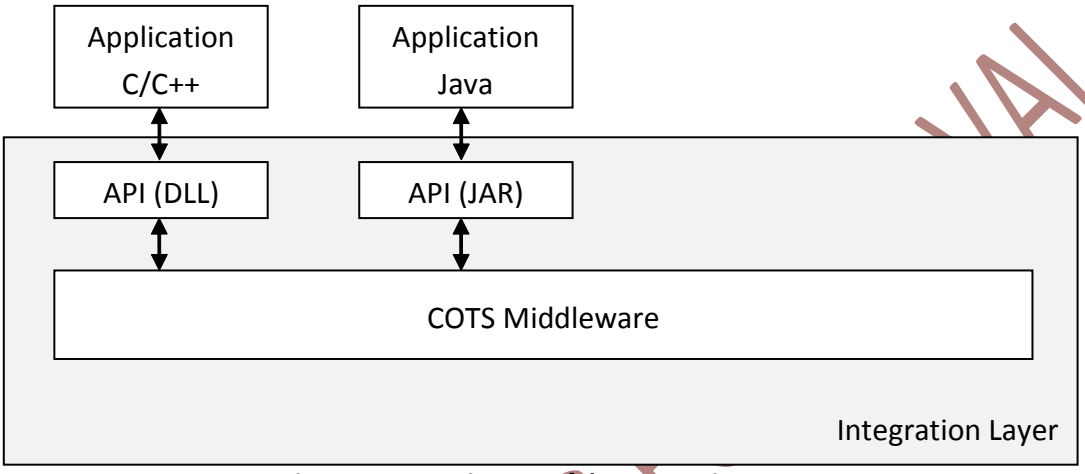

Figure 4.1: Constituents of the Integration Layer

The Integration Layer uses existing COTS middleware and separates it by means of a dynamic library for C/C++ and Java clients. In opposite to the conventional message based middleware products, the IL is responsible not only for the data distribution, but for data management as well. For this purpose, it combines a non-relational database with publish-subscribe mechanism.

To manage the communication processes on the Integration Layer InMemoryGrid-Technology will be applied. [Hazelcast 2017], [Redis 2017] show the general approach for data access and distribution. They support on the one hand the Java.Utils.Map-like API for access of key-value pairs. On the other hand they provide publish-subscribe mechanism to distribute modifications of key-values to arbitrary number of clients.

Another important aspect of the IL is the standardised data structures and serialisation, which is managed by IL. For this purpose, the IL provides a class diagram in an XML-Format, which can be used for generation of client and serialisation code. At the current stage, the Protobuf-Protocol [Protobuf 2017] has been selected for serialisation, as it combines most of the advantages of a binary protocol with build-in versioning and a “one-command-bi-directional-conversion” in JSON.

The Application Framework (AF) is a set of add-on services, allowing plug-and-play functionality for TMS applications. It is responsible for deployment and appropriate execution of the applications, allowing centralised application management and configuration.

At a first glance the reason for this document and Application Framework at all seems to be questionable. If the main responsibility of AF is container management (plug-and-play functionality of TMS applications), it seems to be obvious to select an existing technological stack and prescribe it for TMS applications for the next 25-30 years.

Several technologies are competing to provide the best solution in the context of container management:

- Different Platform offering function as a service (so called Server less computing) like AWS Lambda (Amazon), Azure Functions (Microsoft), Cloud Functions (Google), Open Whisk (IBM/Apache), Manta (Joyent), hook.io, Iron.IO, Webtask.io, Fission, Function (Red Hat), Kubeless (Bitnami) [iX 6/2017];
- Platforms offering container management as a service (so called container as a service CaaS) e.g. EC2 Container Service (Amazon), Google container Engine (GKE) using different software stacks like Google Kubernetes, Docker Machine, Docker Swarm, Apache Mesos, etc;
- Platforms as a service (PaaS) with over 70 vendors today [https://paasfinder.org/vendors].

There are several justifications against this approach:

- The available platforms are incompatible, and provide non standardised API;
- The required granularity for management of stateful microservices (hot standby, warm standby, real-time failover, global singletons, etc) is not supported.

As it was shown in the previous section, the market for PaaS is currently very volatile and the standardised solution available on the market for the next 30 years needs to be yet established.

This is the reason to specify a narrow API hiding different existing implementations and providing enough flexibility for management of TMS applications. A detailed functional description of AF is provided in [D8.6].

5 AF Constituents

The overall architecture of the Application Framework is represented in Figure 5.1, where there are represented the active instances of the Application Framework and the Topics they use for communication (with green background).

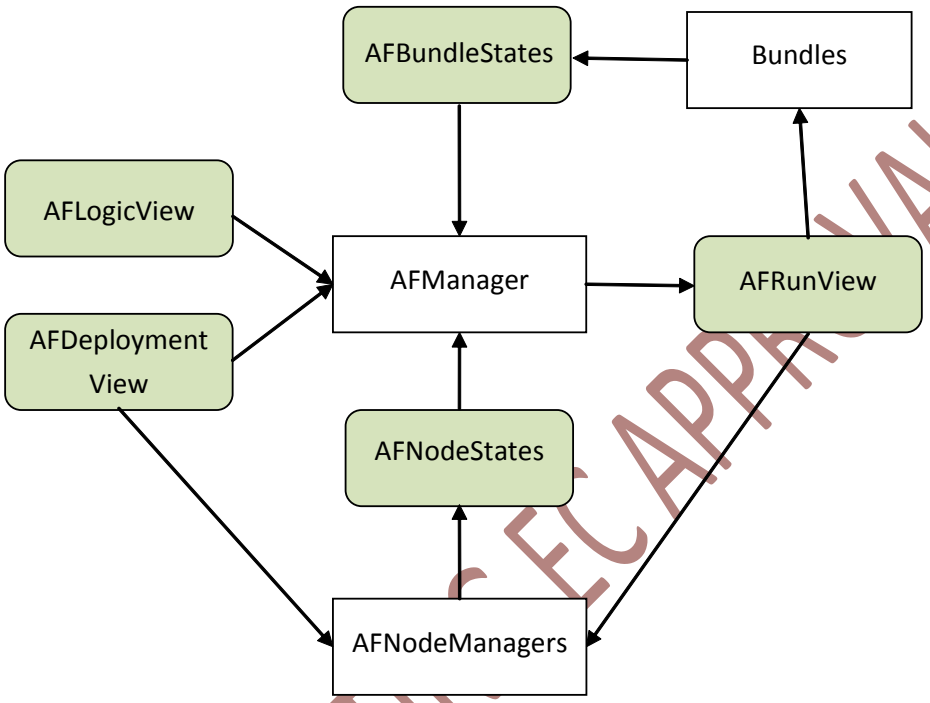

Figure 5.1: Architecture of Application Framework

It consists of the following information Topics (or "Maps", term used in the context of IMDG):

- AFLogicView;
- AFDeploymentView;
- AFRunView;
- AFNodeStates;
- AFBundleStates.

We only specify five message types – one for each Topic in IL (AFLogicView, AFDeploymentView, AFRunView, AFNodeStates, AFBundleStates), so that we can manage applications connected to IL-Topics (start, stop, connect, monitor, deploy, undeploy).

The active instances of the Application Framework are the following:

- Bundles;
- AFManager;
- AFNodeManagers.

These instances will communicate with each other with the five Topics mentioned above. Although we will not specify neither AFManager nor AFNodeManager, they are exposed as a proposal in this document, but the vendor is free to apply his own ideas.

5.1. Bundles

Bundles represent executable entities like exe-files, JAR/DLL libraries, Virtual Machines or Containers (e.g. Docker). They can be started either by user specific mechanisms or by Application Framework constituents named **AFNodeManagers**. Bundles publish their running state on the Topic named **AFBundleStates** using appropriate data structure (Appendix 1). Bundles are also subscribed to **AFRunView** Topic and start and stop services contained in the bundle. They also finish themselves according to the "commands" published on **AFRunView** topic (Section 7.1.2). Bundles are provided by several software vendors and cover some TMS-specific functionality.

Every bundle shall include a MANIFEST.MF file that will be deployed with the rest of the artefacts. This file will include the following fields:

5.2 AF Manager

AF Manager is a constituent of the Application Framework and represents a central logic, which decides on which Node which Bundle and Service shall run. At any point in time only one AFManager instance is active in one Application Framework. It publishes its decisions on the Topic named **AFRunView**.

5.3 AF Node Managers

AF Node Managers are constituents of the Application Framework and represent a logic unit running on each Node (executing entity like Server, Virtual Machine or a Docker cluster). It is responsible for the following tasks on the Node it manages:

- Deployment of the specific bundle;
- Publication of the Node state on the Topic AFNodeStates to allow dynamic failover, monitoring and load balancing;
- Start of the single bundles according to the decisions published on Topic AFRunView.

5.4 AF Implementation

To configure the Application Framework, the system administrator shall provide two configurations:

- **AFLogicView,** where it is specified which services shall read and write to which Topics on the IL. Besides that, it specifies the start/stop logic and failover options (see Section 6);

- **AFDeploymentView**, where it is specified which bundles (containing services) shall be deployed on which Node.

The initial intention for the possible implementation was to follow the microscopic architectural approach and allow a specific Application Framework-Service for each function:

- **Node monitoring (publishing Node state);**
- **Bundle deployments;**
- **Bundle starting on Nodes.**

A prototype implementation showed that the code wrapping some Off-the-shelf-functionality involved in each function is relatively small, so tripling the administration complexity in comparison to the monolithic approach is not acceptable. The decision about the number of services influences the number of used Topics and Data structures strongly as well; each service must have at least one specific Topic with special data structures for publishing its state. With the decision of having only one AF-executable on each Node, the following points are achieved:

- Reduction of the complexity of setup of a new Node managed by Application Framework. Only one executable must be installed on a Node and started in service-mode with configured ID. The deployment and management of the functional software in plug-and-play-manner is responsibility of the Application Framework;
- Combining several services in one executable allows a reduction of the number of required messages and Topics, what means a strong reduction of the required network bandwidth – instead of two independent messages with Node state, bundle deployment state we have only one.

Every Topic will be described in section 7, and they will be structured in the following way:

- Configuration Topics:
  - AFLogicView,
  - AFDeploymentView;
- Start/Stop command Topic:
  - AFRunView;
- Status Topics:
  - AFNodeStates;
  - AFBundleStates.

6 Message Definition Syntax

To specify the class diagram for automatic code generation for serialisation and to use in vendor specific applications, a simple and specific description mean is required. Existing tools and standards for UML produce very flexible but complex object serialisation. Such UML models also require a common meta-model in order to enable interoperability between applications using the generated code. Therefore, it was decided to use a particular XML-based format for specification of data structures. The following tables provide a description of its elements:

- Element **module**

Attribute name	string
Children	enum, struct, union
Description	It is a container for structure and has only one attribute – name. It is used to separated namespaces between different domains and prevent name collisions. For example, a "signal" element in the context of TMS can be a different structure than in the Maintenance management system.

Table 6.1: Description of element module

- Element **enum**

Attribute name	string
Children	enumerator
Parent	module
Description	Provides a container for enumeration. The child objects define single values

Example	`<enum name="ChangeType">`
	`<enumerator name="SET" value="0"/>`
	`<enumerator name="DELETE" value="1"/>`
	`<enumerator name="INSERT" value="2"/>`
	`</enum>`

Table 6.2: Description of element enum

- Element **enumerator**

Attribute name	String, contains a name to be used for enumeration. Typically in capital letters
Attribute value	Integer
Children	No
Parent	enum
Description	Provides a container for enumeration. The child objects define single values
Example	`See "enum"`

Table 6.3: Description of element enumerator

- Element **struct**

Attribute name	String, id of the structure
Attribute extends	String, references some other defined structure if the current structure is inherited from it. The reference shall be of type moduleName.structName. If module name is the same or “common”, it can be skipped.
Attribute abstract	Boolean, if the structure will not be used for serialisation itself, but planned to be extended by others, the attribute shall be set to “true”. Default value: “false”
Children	attr
Description	The element specifies a structure with a name. Typically it represents a serialised message.
Example	`<struct name=”valueChange” extends=”Change” abstract=”true”/>`

Table 6.4: Description of element struct

- Element **attr**

Children	no
Parent	**struct**
Description	It is a container for structure and has only one attribute – name. It is used to separate namespaces between different domains and to prevent name collisions. For example, a “signal” element in the context of TMS can be a different structure than in the Maintenance management system.
Attribute name	String, it will be used in generated code for access the value of the attribute.
Attribute id	Boolean, defines if the attribute represents an id. Typically it would be of type “string”, but it could be an integer or reference to some other struct as well.
Attribute type	string, as type the id of any defined struct can be used. Some of primitive structs like integer, double are specified in common.cm. To reference structs defined in different modules, a point-separated notation shall be used. For example “sd.Track” with moduleName.structName. If module name is not specified, then it is the current module or the common-module.
Attribute attrId	Integer, mandatory. To enable binary encoding and ensure long term backwards compatibility, an integer id shall be provided for each attribute. This id shall be unique inside of the struct-element. If shall start with 1, but in case the struct extends some other it shall start with 2.
Attribute defaultValue	String, it is not used for serialisation, but can be used for client code generation.
Attribute maxOccurs	Positive integer, if the attribute represents a sequence, the value of maxOccurs shall be either “unbounded” or contain a number > 1.
Attribute minOccurs	Integer, optional. The default value is 0 – all attributes of a structure are optional by default. For required attributes minOccurs shall have a value > 0.
Attribute containment	Boolean (“true” or “false”). Default = “true”. Annotates the composition relation between the current structure and the attribute type structure.

	If true, in the serialisation the attribute will contain the structure by value. If false, the serialisation will contain a string representing an address of the referenced object.
Attribute sorted	Boolean, default = “false”. If the attribute maxOccurs is not 0, this attribute annotates that the elements inside are sorted according to their id-attribute. This allows appropriate selection of the container (e.g. map) during code generation. If true, the elements inside of this sequence can be referenced by their id in other structures. Otherwise, an index has to be used.
Attribute typicallySigned	Boolean, for the purpose of efficient serialisation it is important to know if the integer (int32 or int64) represented by the current attribute will be signed (positive and negative) or unsigned in most of the cases.
Attribute typicallyVarLength	Boolean, default “true”. For the purpose of efficient serialisation it is important to know if the declared type will require all the declared bit – length in most of the cases, e.g. a timestamp uint32 in seconds will typically cover up to 24*3600 seconds and require 17 bits. In this case, it is reasonable to say “true” to enable encoding of the value in 1-3 bytes depending on value.

Table 6.5: Description of element attr

```
<struct name=”ValueChange” extends=”Change” abstract=”true”>
      <attr name=”objectRef” type=”string” attrId=”2”/>
<attr name=”attributeId” type=”uint32” typicallyVarLength=”true”
attrId=”3”/>
<attr name=”type” type=”ChangeType” defaultValue=”SET” attrId=”4”/>
<attr name=”index” type=”uint32” typicallyVarLength=”true” attrId=”5”/>
</struct>
```

Table 6.6: Example of a struct-element

- Element **union**

This element is almost the same as the element struct with the exception that only one attribute of the listed is present in the message. It can be used as a type in other unions and structures. The main purpose of this structure is to provide dynamic polymorphism – if used in some attribute, the message type and the message will be sent, so that the receiver can reconstruct it.

```
<union name=”ChangeUnion”>
<attr name=”longValue” type=”ChangeLong” attrId=”1”/>
      <attr name=”doubleValue” type=”ChangeDouble” attrId=”2”/>
      <attr name=”stringValue” type=”ChangeString” attrId=”3”/>
      <attr name=”objectValue” type=”ChangeObject” attrId=”4”/>
      <attr name=”objectRefValue” type=”ChangeObjectRef” attrId=”5”/>
      <attr name=”group” type=”ChangeGroup” attrId=”6”/>
</union>
```

Table 6.7: Example of element union

7 Topics

This section describes the different Topics used in the Application Framework.

7.1 Configuration Topics

7.1.1 AFLogicView

It represents a logical view on the Application Framework. This view is built by a list of the structure **ServiceConfig** representing a logical entity, which is connected to specific Topics (see Figure 7.1).

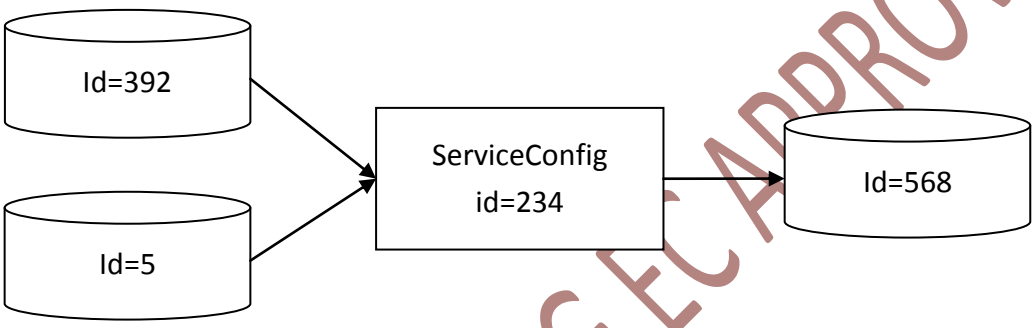

Figure 7.1: Logical view on AF as a list of ServiceConfigs – here service id234 connected to topics with ids 392, 5 and 568

The primary information pieces of the **ServiceConfig** are:

- Id of the **ServiceConfig** to be referenced by **AFRunView**;
- Ids of the connected Topics (incl. of their specification) to allow Bundles to connect to the required topics after bundle start.

The following assumptions are made:

- One service can be delivered in different bundles; e. g. a timetable editor can be delivered as an executable and as a Docker image;
- Several instances of the same **ServiceConfig** can be started concurrently. They all would read and write to the same topics, but probably to different keys. Again the same example: several timetable editors can be started concurrently, but they can modify concurrently distinct trips to prevent inconsistencies.

In the following we specify the **ServiceConfig** in more detail according to the class diagram in Figure 7.2.

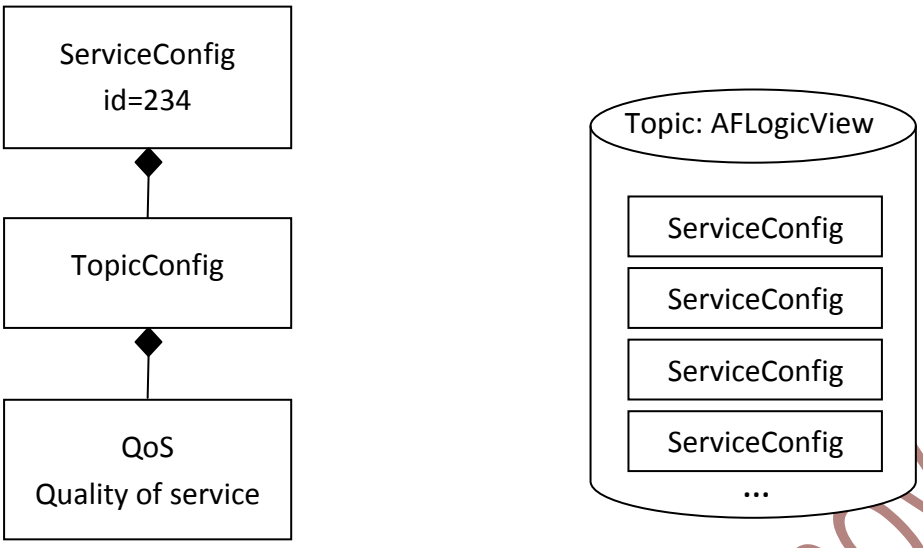

Figure 7.2: Class diagram for ServiceConfig, which is managed in AFLogicView-Topic

It seems illogical to specify **TopicConfig** for the same topic in each **ServiceConfig**: at least once for the Topic-writer and once for the Topic-reader. The reason for that is that different participants on a Topic could have different requirements on data delivery:

- the Topic-writer could provide persistent data management and a data history for the last 1000 states of an object;
- the Topic-reader wants to represent only the current state, therefore it is not interested in persistency and require only the one last value;
- another Topic-reader requires the history of last 100 states and is interested in at least transient durability of the data (several copies of the data in RAM).

All these participants would specify their requirements on Topic with different Quality of Services, allowing the Middleware to establish the most effective communication.

7.1.1.1 Data structure QoS (Quality of Service)

Quality of Service specifies different aspect for delivery of messages. We are starting according to bottom-up principle. The term Quality of Service in the context of IMDG is strongly influenced by the Data Distribution Service – Standard of OMG [OMG DDS 2015]. Their specification provides more data structures than we used in Application Framework. We had to limit QoS functionality to be able to integrate other existing IMDG-technologies. In the future, QoS could be extended together with improvements on IMDG market.
One of the attributes of the Quality of Services is Durability, which will be specified as an enumerator (Appendix 1).

7.1.1.2 Data structure TopicConfig

The TopicConfig data structure describes the Topic configuration needed by the Service for exchange information with other applications (Appendix 1).

7.1.1.3 Data structure ServiceConfig

One of the interesting aspects of a service is when should it be activated. There are several policies that can apply:

- Always active one instance;
- Batch mode activation on timer;
- Load-balanced activation, e.g. start a new instance if overall load of other instances is higher than 70%;
- Start on occurrence of a special key-value in some topic;
- Start on opening of a new Topic.

The data structure "**KeyOnTopicTrigger**" allows to configure triggering the application start by appearance or removal of specail key-value on some topic (Appendix 1).
The data structure "**ServiceActivationConfig**" allows the AFManager to identify when to start a service as described above (Appendix 1).
The data structure "**ServiceConfig**" is a container of the previously specified data structures and provide additional attributes for administration (Appendix 1).

7.1.2 AFDeploymentView

In the previous section we looked at the TMS as a network of connected logical services. In this section we specify how the Application Framework shall be configured to allow automatic start of the logical services.
The services are logical instances located in bundles – which are the executable entities in Application Framework. Typical examples are executables, Docker-container and JAR-libraries managed by a "jar-box".
The data structure living in **AFDeploymentView** Topic is "**BundleDeployment**", and answers the following questions:

- on which Node which bundle is deployed;
- which logical Services listed in AFLogicView can be started inside of which Bundle.

The Application Framework deduces where to start a logical service from the list of deployed Bundles.

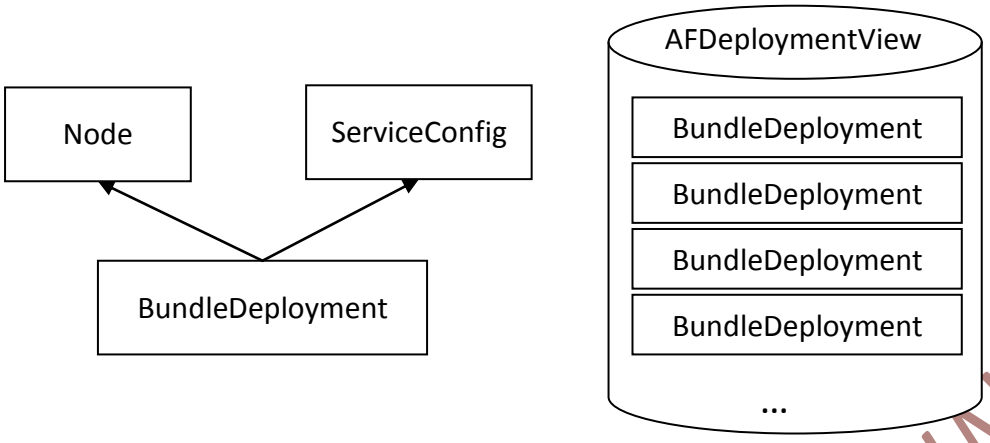

Figure 7.3: Class diagram of BundleDeployment and its location in Topic AFDeploymentView

The Application Framework shall support different types of bundles. On the other hand the bundle type prescribes the runtime environment. E. g. a docker image-bundle requires a Docker infrastructure of a specific type and version installed on the Node.

As a first approach we specify the type of the bundle in an enumeration. We assume that the same information about runtime can be deduced from the bundle-archive-ending (e.g. exe for executable on Windows, jar for a Java archive, vdi, ova, ovf for virtual machine images). Attributes with runtime vendor and version are optional and simplify checks by the system administrator.

The "**BundleDeployment**" data structure allows the **NodeManager** to deploy and to start the deployed bundle (Appendix 1). One of its attributes is the type of the bundle (**BundleType**).

7.2 Topics for AF-Commands

The Application Framework has two kinds of information flow which look like a command:

- Deploy bundle XY on Node N2;
- Start bundle XY on Node N2;
- Start service ZX inside of the bundle instance XY running on Node N2.

In the context of Integration Layer, the commands are represented by the "desired" state, which is published to all subscribers and continues to be active, until the "desired" state changes or the desired state is achieved. To identify that the desired state is achieved, the requesting service observes some topic representing current state or observes the "reply topic", where some other service claims to achieve the desired state.

In this section we model data structure required to represent the commands above. The "replies" are published on **AFBundleStates** and **AFNodeStates** – topics and will be specified partly in this section and partly in Section 7.3.

7.2.1 Deploy bundle command

In the deployment workflow there are two Topics involved:

- the desired deployment state is represented in the Topic **AFDeploymentView**, which is already covered in section 7.1.2;
- the status of the deployment the NodeManagers publish as part of the NodeState-Message on **AFNodeStates**-Topic.

In the following we cover only the "reply" channel. To reduce the network bandwidth we assume that most of the time the current deployment state is equal to the "desired" deployment state. Therefore it is not reasonable to publish mostly the same information twice: once as a desired and once as a current state. We decided to integrate the transition from the current state to the desired state as set of steps (automat) into the **NodeState**-Message. As soon as the transition is completed the only deployment information to be send is the timestamp as the version of the implemented desired deployment state. The deployment status on some node is integrated into **NodeState**-Message (Appendix 1).

If some deployment failed, the current deployment state will be continuously sent until the system administrator or some deployment logic identifies the failure by timeout and creates a new desired deployment state or corrects errors in the Node-Installation.

7.2.2 Start bundle command

The logic view represented in the **AFLogicView**-Topic contains the source information for the Application Framework to decide which services shall be run on which node, depending on their availability. To implement the start/stop functionality it is followed the same pattern as for deployment:

- There is a topic representing desired/current running state which is **AFRunView** (see Figure 7.4);
- There is a topic representing deviation from the desired state as part of the **BundleState**-Message in **AFBundleStates**.

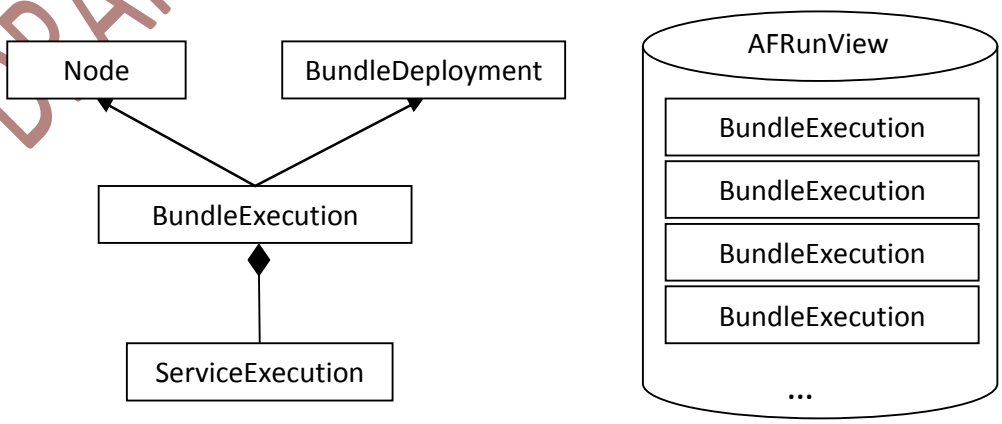

Figure 7.4: Class diagram for BundleExecution structure and the Topic managing it

A service inside of bundle can have different states (see Enum **ServiceActivityState**, Appendix 1).
The Bundle-Implementation uses the data structure "**ServiceExecution**" to identify how to configure the relevant service (why it was started, to which topics it shall connect) (Appendix 1).

The **NodeManager** uses data structure "**BundleExecution**" to identify, which bundle to start on the managed Node. The Bundle-Implementation uses this data structure to identify which services shall be activated and how often the state of the bundle shall be published (Appendix 1).

The **NodeManager** gives feedback about starting activity of a bundle in the context of its state in the data structure "**BundleStartStatus**" (Appendix 1).

7.3 Topics for AF-Status

In the Application Framework, two instances are publishing its current state:

- **NodeManagers** publish information about Node load, deployment and bundle-starting activities. The target of these messages is the Topic **AFNodeStates**, and the data structure used is "**NodeState**" (Appendix 1);
- Bundle-Implementations publish information about its running state. Target of these messages is Topic **AFBundleStates**, and the data struacture is "**BundleState**".

Bundle Implementation shall publish its state using "BundleState" data structure (Appendix 1).
It is assumed that all messages provide a timestamp of the sender as a part of meta information in Integration Layer (s. D8.3).

8 Conclusion

The Application Framework provides optional services building a plug-and-play infrastructure for light weight Apps. The main building blocks are the container management.

This document has described the Data structure, Message Definition Syntax and Communication Paths of the data interchanged by the AF.

This specification is the first draft, and will be evaluated in WP7 and in Shift2Rail before getting a real standard. After that, it will be used as a basis for development of new innovative functions. It is expected that the AF will be extensively used to reduce development efforts and provide integrated system management.

9 Glossary

Term	Definition
Application Layer	Common infrastructure that handles the system management of the applications; a communication framework that enables dynamic and flexible interaction between the TMS applications, an interface framework for implementing the application interfaces for interaction between the applications, a broker framework that allows for interaction of the TMS application with other systems via the Integration Layer.
Application Group	A set of applications that provides a Business capability
Broker	Software that manages Information distribution amongst the connected services between Application Layer and Integration Layer.
Bundle	Set of executables, libraries and configuration files in the same manner as containers. Term used as OSGi-Standard.
Container	See Bundle.
Data Mart	Section of the Data Warehouse.
Data Warehouse	Central repository of integrated data from one or more disparate sources (services one of them being the TMS). The Date warehouse contains for each service specific data in different sections (data marts) presenting History, Now-cast and Forecast information.
Decentralised	Responsibility distributed, not one single entity has control over all the processing (See https://www.quora.com/Whats-the-difference-between-distributed-and-decentralized-in-Bitcoin-land).
Distributed	Physically distributed, not all the processing of the transactions is done in the same place (See https://www.quora.com/Whats-the-difference-between-distributed-and-decentralized-in-Bitcoin-land).
Forecast	Estimate information in the future, deviations from the plan estimated.
Integration Layer	Communication link between the different Business Services.
Node	Typically it is a computer with some operating system running or a virtual machine where bundles/containers are deployed. With the existence of cloud based Container-Services, the term Node can mean a managed cluster as well. In this sense the Node is one unit of the execution platform.
Nowcast	Instant information in the present.
Publisher	Entity that publics messages to be consumed by one or more subscribers. Actor of a business process that publishes Topics (i.e. make available and updates).
Subscriber	Entity that consumes messages sent by the Publisher. Actor of a business process that receives updates about one or more different topics it has subscribed.
Topic	Information specific to a business process. A topic is made up of a structured collection of operational data.

10References

[Hazelcast 2017]	http://docs.hazelcast.org/docs/3.8.1/manual/html-single
[Redis 2017]	https://redis.io/documentation
[Protobuf 2017]	https://developers.google.com/protocol-buffers/
[D8.3]	Description of Integration layer and Constituents
[D8.5]	Requirements for the Generic Application Framework
[D8.6]	Description of the Generic Application Framework and its constituents, In2Rail, 2017
[OMG DDS 2015]	http://www.omg.org/spec/DDS/1.4
[CRON 2017]	https://en.wikipedia.org/wiki/Cron
[iX 06/2017]	Serverless computing, iX Magazin für professionelle Informationstechnik, 06/2017

Appendix 1: Data structures

In this appendix there are defined the different data structures and attributes utilised in the Topics involved in the Application Framework:

Quality of Service / QoS

```
<struct name="QoS">
  <attr name="reliableTransport" type="boolean" default="true"
attrId="1"/>
  <attr name="durability" type="Durability" default="VOLATILE"
attrId="2"/>
  <attr name="historyDepth" type="uint32" typicallySigned="true"
attrId="3" default="1"/>
  <attr name="latencyBudgetMs" type="uint32" attrId="4"
default="1000"/>
  <attr name="transportPriority" type="int32" typicallySigned="true"
attrId="5" default="10"/>
  <attr name="lifespanMs" type="uint64" default="0" attrId="6"/>
</struct>
```

Name	Type	Description
reliableTransport	Boolean	If true, the reliable communication protocol with acknowledges from receiver is applied, similar to TCP. If false best effort algorithm is applied, similar to UDP.
durability	Durability	See above.
historyDepth	uint32	Specifies, how many historical values of an object will be provided to a late joining client. It makes sense only in case of not VOLATILE-duration. Typical value is 1. The historyDepth represents the upper. If quality of services is specified at the subscriber side, the value specifies the number of historical values to be provided to late joining subscriber.
latencyBudgetMs	uint32	Specifies the number of milliseconds the middleware can use to group several messages into one batch for more efficient transmission.
transportPriority	Int32	Values sent on the topic with the lower priority will be sent on a client after higher priority topic has sent all its messages.
lifespanMs	uint64	Specifies, how many milliseconds a value is interesting for the subscribers after publication. After this amount of time, the value can be removed from IMDG. The value 0 means – forever.

Table A.1: Attribute description of the QoS (Quality of Services) – Data structure

```
<enum name="Durability">
```

```
      <enumerator name="VOLATILE" value="0"/>
      <enumerator name="TRANSIENT_LOCAL" value="1"/>
      <enumerator name="TRANSIENT" value="2"/>
      <enumerator name="PERSISTENT" value="3"/>
</enum>
```

Durability defines how long the data should survive in the context of IMDG (s. Table 11.1.2).

Enumerator value	Description
VOLATILE	The value can be removed as soon as all currently subscribing clients received it.
TRANSIENT_LOCAL	The value will be kept for late joining subscribers as long as the service published them is running.
TRANSIENT	The value will be kept for late joining subscribers as long as Integration Layer - preconfigured Nodes - are running.
PERSISTENT	The value will be kept even in case of shut down of the entire Integration Layer.

Table A.2: Description of Durability enumerators

TopicConfig

```
<struct name="TopicConfig">
  <attr name="topicId" type="string" id="true" attrId="1"/>
  <attr name=”portId” type=”uint32” attrId=”2”/>
  <attr name="dataType" type="string" attrId="3"/>
  <attr name="qos" type="QoS" containment="true" attrId="4"/>
  <attr name="modelAddressExpression" type="string" attrId="5"/>
  <attr name="readAccess" type="boolean" attrId="6"/>
  <attr name="writeAccess" type="boolean" attrId="7”/>
</struct>
```

Name	Type	Description
topicId	string	A unique ID of the topic inside of one Integration Layer. Only alpha-numeric characters are allowed [a-zA-Z0-9].
portId	uint32	A service has its distinguished input and output topics as sources of information – these topics the service references internally by portId (which is unique per service). E.g. Port=1 for input timetable Port=2 for output of validation results. The pair topicId & portId connects the internal service logic with a specific Topic in IMDG.
dataType	string	Represents the data type of the values “living” on this topic. Data type is specified as

Name	Type	Description
		ModuleName.MessageName. By convention the module name shall be identical with a protobuf-file-name.
qos	QoS	See above.
modelAddressExpression	string	The modelAddressExpression represents a prefix which shall be applied to the keys managed in this topic to obtain the absolute address of the element in canonical model. E.g. modelAddressExpression=/tms/af key=node[12]/bundle[62] provides the absolute address in canonical model /tms/af/node[12]/bundle[62].
readAccess	boolean	Specifies if the service is allowed to subscribe to the topic.
writeAccess	boolean	Specifies if the service is allowed to publish to the topic.

Table A.3: Attribute description of the TopicConfig-Data structure

KeyOnTopicTrigger

```
<struct name="KeyOnTopicTrigger">
  <attr name="topicId" type="string" attrId="1"/>
  <attr name="keyExpression" type="string" attrId="2"/>
  <attr name="triggerOnAppearance" type="boolean" attrId="3"/>
</struct>
```

Name	Type	Description
topicId	string	A unique ID of the topic inside of one Integration Layer. Only alpha-numeric characters are allowed [a-zA-Z0-9].
keyExpression	string	As soon as a new key patching to the regular expression occurs on the specified topic, the service will be started by Application Framework and notified about trigger element.
triggerOnAppearance	boolean	If true, the service will be activated on occurance of the new value, if false – the service will be activated on removal of the value from Integration Layer.

Table A. 4: Attribute description of the data structure KeyOnTopicTrigger

ServiceActivationConfig

```
<struct name="ServiceActivationConfig">
```

```
  <attr name="timerConfigs" type="string" maxOccurs="unbounded" attrId="1"/>
   <attr name="averageLoadPercent" type="uint32" default="0" attrId="2"/>
   <attr name="keyTriggers" type="KeyOnTopicTrigger" containment="true"
maxOccurs="unbounded" attrId="3"/>
   <attr name="topicTriggers" type="string" maxOccurs="unbounded"
attrId="4"/>
   <attr name="spareInstances" type="uint32" default="0" attrId="5"/>
   <attr name="minActiveInstances" type="uint32" default="0"
attrId="6"/>
</struct>
```

Name	Type	Description
timerConfigs	String [0..*]	Configuration of activation times in cron-format [CRON 2017]
averageLoadPercent	uint32	If not 0, the Application Framework would start additional instances and stop running instances to ensure the specified averageLoad in % as reported by the service in its state-message.
keyTriggers	KeyOnTopicTrigger [0..*]	See above.
topicTriggers	string [0..*]	As soon as a new Topic with an ID matching to one of the topicTriggers is created (a service publishes its state) the service will be activated together with the trigger-topic.
spareInstances	uint32	For services with long activation time, the application Framework can pre-start "spareInstances" bundles with inactive services.
hotStandbyInstances	uint32	For essential services the Application Framework can pre-start a set of bundles with services in "Passive"-State (s. below). If an active service instance misses its heart-beats the Application Framework deactivates the active Service instance and activates the passive one.
minActiveInstances	uint32	Application Framework ensures this amount of instances to be started and in state Active.

Table A.5: Attribute of the data structure ServiceActivationConfig

ServiceConfig

```
<struct name="ServiceConfig">
```

```
  <attr name="id" type="string" id="true" attrId="1"/>
  <attr name="typeId" type="string" attrId="2"/>
  <attr name="readableName" type="string" attrId="3"/>
  <attr name="description" type="string" attrId="4"/>
  <attr name="heartbeatMS" type="uint32" default="0" attrId="5"/>
  <attr name="singleton" type="boolean" default="false" attrId="6"/>
  <attr name="activationConfig" type="ServiceActivationConfig"
containment="true" attrId="7"/>
  <attr name="topics" type="TopicConfig" containment="true"
maxOccurs="unbounded" attrId="8"/>
  <attr name="privateConfigFileURI" type="string" attrId="9"/>
</struct>
```

Name	Type	Description
id	string	A unique ID of the logical Service in the context of one Integration Layer. It could be an UUID.
typeId	string	An id of a service type known to the bundle, so the bundle is able to activate the write service. The difference to the id-attribute is, that typeId is independent from the connected Topics – several services can share the same typeId.
readableName	string	Human understandable representation of the service function.
description	string	Supporting description for the system integrator and system maintenance.
heartbeatMS	uint32	If value > 0 specified, the bundle containing the service should publish its state every heartbeatMS millisecond. Missing heartbeats would be used for activation of a replacement service instance.
singleton	boolean	Specifies, if only one instance of this service is allowed to be in state Active (typically means to allow writing to the output-Topics). The Application Framework is responsible to ensure this functionality.
activationConfig	ServiceActivationConfig	s. above. If activationConfig is not provided, the AF will ensure the service to run always as one instance.
topics	TopicConfig [0..inf]	s. above.
privateTopicConfigURI	string	Represents a link to vendor specific configuration file installed locally on the Node. It could contain some private information and protected by the OS-means (read/write access control).

Table A.6: Attribute description of the data structure ServiceConfig

BundleDeployment

```
<struct name="BundleDeployment">
```

```
  <attr name="id" type="string" key="true" attrId="1"/>
  <attr name="type" type="BundleType" attrId="2"/>
  <attr name="nodeIds" type="string" attrId="3"/>
  <attr name="serviceIds" type="string" maxOccurs="unbounded"
attrId="4"/>
  <attr name="imageId" type="string" attrId="5"/>
  <attr name="executableId" type="string" attrId="6"/>
  <attr name="workingDirPath" type="string" attrId="7"/>
  <attr name="entryPoint" type="string" attrId="8"/>
  <attr name="cpuQuotaPercent" type="uint32" attrId="9"/>
  <attr name="ramQuotaMB" type="uint32" attrId="10"/>
  <attr name="runtimeProduct" type="string" attrId="11"/>
  <attr name="runtimeVersion" type="string" attrId="12"/>
</struct>
```

Attribute name	Type	Description
nodeIds	string [0..*]	Identifiers of the nodes, where the bundle is deployed.
serviceIds	string [0..*]	Ids of the services listed on AFLogicalView incorporated into the bundle.
imageId	string	Depending on Bundle type the value represents: - URL to the archive (JAR, DLL, EXE) - ID of the Docker Image Application Framework would use this value to download the Bundle-Archive and install on the local drive assigned for the Node-Manager. Docker-Images will be handled by the Docker-Infrastructure.
executableId	string	Depending on Bundle type the value represents: - Executable file *.exe/*.cmd/*.sh/* - ID of the Docker Container - Jar file - DLL file
workingDirPath	string	Specifies relative path of the working directory. The basis path will be the deployment directory. This attribute is relevant for executable bundles only.
Type	BundleType	See below.
entryPoint	String	Entry point specifies the class or function that shall be used for starting in case if the bundle type is DLL or JAR.
cpuQuotaPercent	uint32	If > 0, the Application Framework would start the bundle with the limitation of CPU usage.
ramQuotaMB	uint32	If > 0, the Application Framework would start the bundle with the limitation of RAM usage.
runtimeProduct	string	Optional name of the run time product like VirtualMachine, Vmware, Docker
runtimeVersion	string	Optional field which allows the system administrator to decide, if the used bundle is compatible with the

Attribute name	Type	Description
		system installed on the Node.

Table A.7: Attribute of the BundleDeployment data structure

```
<enum name="BundleType">
  <enumerator name="BUNDLE_EXE" value="0"/>
  <enumerator name="BUNDLE_DLL" value="1"/>
  <enumerator name="BUNDLE_JAR" value="2"/>
  <enumerator name="BUNDLE_SHELL_CMD" value="3"/>
  <enumerator name="BUNDLE_DOCKER_IMAGE" value="4"/>
  <enumerator name="BUNDLE_VIRTUALBOX_IMAGE" value="5"/>
  <enumerator name="BUNDLE_WMWARE_IMAGE" value="6"/>
</enum>
```

DeploymentStatus

```
<struct name="DeploymentStatus">
  <attr name="bundleId" type="string" attrId="1"/>
  <attr name="bundleTimestamp" type="timestamp" attrId="2"/>
  <attr name="step" type="DeploymentStep" attrId="3"/>
  <attr name="stepPercent" type="int32" default="0" attrId="4"/>
</struct>
```

Each deployment step can take several seconds, so it is introduced the step completion attribute.

Attribute name	Type	Description
bundleId	string	Identifiers of the bundle listed on AFDeploymentView.
bundleTimestamp	string	Timestamp of the key-value containing the bundleId. It is used by the Application Framework to assign the current bundle deployment state to the "command" – after modification of the BundleDeployment-Structure it takes several milliseconds, until the NodeManager publishes the state of implementation of this "command".
step	DeploymentStep	For bundles EXE, JAR, DLL we have three deployment steps. Docker infrastructure combines them together. At the end of the deployment process the state shall be DEPLOYMENT_FINISHED for each bundle type.
stepPercent	int32	0 .. 100% represents the normal behaviour. -1 represents failed step.

Table A.8: Attributes of the data structure DeploymentStatus

```
<enum name="DeploymentStep">
  <enumerator name="DEPLOYMENT_DOWNLOAD" value="0"/>
  <enumerator name="DEPLOYMENT_UNZIP" value="1"/>
  <enumerator name="DEPLOYMENT_POSTPROCESSING" value="2"/>
  <enumerator name="DEPLOYMENT_FINISHED" value="3"/>
```

```
  <enumerator name="DEPLOYMENT_REMOVED" value="4"/>
</enum>
```

NodeState

```
<struct name="NodeState">
  ...
  <attr name="deployments" type=" DeploymentStatus"
containment="true" maxOccurs="unbounded"/>
  <attr name="deployedTimestamp" type="timestamp"/>
  ...
</struct>
```

In the NodeState-Structure the deployment part is represented by two attributes:

- deployments – representing status of currently deployed bundles during the deployment process;
- deployedTimestamp – represents a time stamp before which all required deployments from AFDeploymentView are successfully implemented.

BundleExecution

```
<struct name="BundleExecution">
  <attr name="bundleId" type="string" attrId="1"/>
  <attr name="nodeId" type="string" attrId="2"/>
  <attr name="heartbeatMS" type="uint32" default="0" attrId="3"/>
  <attr name="services" type="ServiceExecution" containment="true"
maxOccurs="unbounded" attrId="4"/>
</struct>
```

Attribute name	Type	Description
bundleId	string	Ids of the bundle listed on AFDeploymentView.
nodeId	string	Id of the current Node
heartbeatMS	uint32	The Bundle shall publish its state with the heartbeatMS [in milliseconds], which will be calculated from the required heartbeats of the containing active services by the application framework. 0 – no heartbeats are required, publish state on modification only.
services	ServiceExecution [0..*]	s. above

Table A.9: Attributes of the data structure BundleExecution

```
<enum name="ServiceActivityState">
  <enumerator name="SERVICE_NOT_STARTED" value="0"/>
  <enumerator name="SERVICE_RUNNING" value="1"/>
  <enumerator name="SERVICE_PASSIVE" value="2"/>
</enum>
```

Enumerator name	Description
SERVICE_NOT_STARTED	The service is not activated.
SERVICE_RUNNING	The service reads and writes according to its logic.
SERVICE_PASSIVE	The service subscribed and received all required input and configuration and is ready to start writing. The state represents hot standby of the service.

Table A.10: States of activity of a service in a bundle

BundleStartStatus

```
<struct name="BundleStartStatus">
  <attr name="bundleId" type="string" attrId="1"/>
  <attr name="bundleStartTimestamp" type="timestamp" attrId="2"/>
  <attr name="startState" type="BundleStartState" attrId="3"/>
  <attr name="returnCode" type="int32" default="0" attrId="4"/>
</struct>

<struct name="NodeState">
  ...
  <attr name="bundleStarts" type="BundleStartStatus"
containment="true" maxOccurs="unbounded"/>
  <attr name="bundleStartTimestamp" type="timestamp"/>
  ...
</struct>
```

Attribute name	Type	Description
bundleId	string	Ids of the bundle listed on AFDeploymentView.
bundleStartTimestamp	timestamp	The timestamp of the BundleExecution-key,value which is taken into account.
startState	BundleStartState	Function is obvious from the enumerator names.
returnCode	int32	Represents the return code after the startState reached BUNDLE_FINISHED. Value = 0 means successful finish.

Table A.11: Attributes of the BundleStartStatus


```
<enum name="BundleStartState">
  <enumerator name="BUNDLE_NOT_STARTED" value="0"/>
  <enumerator name="BUNDLE_STARTING" value="1"/>
  <enumerator name="BUNDLE_START_FAILED" value="2"/>
  <enumerator name="BUNDLE_STARTED" value="3"/>
  <enumerator name="BUNDLE_FINISHED" value="4"/>
```

```
</enum>
```

NodeState

```
<struct name="NodeState">
  <attr name="nodeId" type="string" key="yes" attrId="1"/>
 <attr name="deployments" type="DeploymentStatus" containment="true"
maxOccurs="unbounded" attrId="2"/>
  <attr name="deployedTimestamp" type="timestamp" attrId="3"/>
  <attr name="bundleStarts" type="BundleStartStatus"
containment="true" maxOccurs="unbounded" attrId="4"/>
  <attr name="bundleStartTimestamp" type="timestamp" attrId="5"/>
  <attr name="cpuLoadPercent" type="uint32" default="0" attrId="6"/>
  <attr name="freeRamMB" type="uint32" default="0" attrId="7"/>
  <attr name="sendRateKBpSec" type="uint32" default="0" attrId="8"/
  <attr name="receiveRateKBpSec" type="uint32" default="0"
attrId="9"/>
</struct>
```

Attribute name	Type	Description
nodeId	string	Node id.
deployments	DeploymentStatus	See section 7.1
deploymentTimestamp	Timestamp	"Version"-id of the AFDeploymentView the Node has successfully implemented (see Section 7.1)
bundleStarts	BundleStartStatus	See section 7.2
bundleStartTimestamp	Timestamp	"Version"-id of the AFRunView the Node has successfully implemented (see section 7.2).
cpuLoadPercent	uint32	CPU load over all cores and CPU-Sockets.
freeRamMB	uint32	Free RAM in MB.
sendRateKBpSec	uint32	Send statistics over all network interfaces.
receiveRateKBpSec	uint32	Receive statistics over all network interfaces.

Table A.12: Attributes of the data structure NodeState

BundleState

```
<struct name="BundleState">
  <attr name="id" type="string" attrId="1"/>
  <attr name="services" type="ServiceState" containment="true"
maxOccurs="unbounded" attrId="2"/>
```

```
</struct>
```

Attribute name	Type	Description
serviced	string	Identifier of the service as listed in the Topic AFRunView.
status	String	current service status (s. above).
Id	String	Bundle id as listed in AFRunView.
services	ServiceState	Current state of the offered services.

Table A.13: Attributes of the data structure BundleState

ServiceState

```
<struct name="ServiceState">
  <attr name="serviceId" type="string" key="true" attrId="1"/>
  <attr name="status" type="ServiceRunStatus" attrId="2"/>
</struct>

<enum name="ServiceRunStatus">
  <enumerator name="SERVICE_NOT_STARTED" value="0"/>
  <enumerator name="SERVICE_RUNNING" value="1"/>
  <enumerator name="SERVICE_PASSIVE" value="2"/>
  <enumerator name="SERVICE_FINISHED" value="3"/>
</enum>
```

Name	Description
SERVICE_NOT_STARTED	List of all not started services which were required by AFRunView. Internal not started services will not be announced.
SERVICE_RUNNING	Service is in running state
SERVICE_PASSIVE	Service is in hot-standby state.
SERVICE_FINISHED	Service is intended for batch-processing and it fulfilled its task. The AFManager uses this state to decide to stop the service or the entire bundle. The service is not allowed to stop itself without a "command" on AFRunView.

Table A.14: Enumerators of ServiceRunStatus