Abstract
This deliverable describes the vf-OS Devices, Drivers and APIs Toolkit Specifications and contains a report of the activities and progress of all development tasks in WP4. Thus, this document explains the development and availability of the different Device Drivers and API Connectors of the virtual factory Input / Output (vf-IO).
Document Status

<table>
<thead>
<tr>
<th>Deliverable Lead</th>
<th>Raúl Poler, UPV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Reviewer 1</td>
<td>João Sarraipa, UNINOVA</td>
</tr>
<tr>
<td>Internal Reviewer 2</td>
<td>Raquel Melo de Almeida, KBZ</td>
</tr>
<tr>
<td>Internal Reviewer 3</td>
<td>Stuart Campbell, ICE</td>
</tr>
<tr>
<td>Type</td>
<td>Deliverable</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP4: Virtual Factory I/O</td>
</tr>
<tr>
<td>ID</td>
<td>D4.1.2a: WP4 Virtual Factory I/O Umbrella Deliverable</td>
</tr>
<tr>
<td>Due Date</td>
<td>2018-03</td>
</tr>
<tr>
<td>Delivery Date</td>
<td>2018-03</td>
</tr>
<tr>
<td>Status</td>
<td>Consortium Approved</td>
</tr>
</tbody>
</table>

History

See Annex A

Status

This deliverable is subject to final acceptance by the European Commission.

Further Information

www.vf-OS.eu

Disclaimer

The views represented in this document only reflect the views of the authors and not the views of the European Union. The European Union is not liable for any use that may be made of the information contained in this document.

Furthermore, the information is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user of the information uses it at its sole risk and liability.
Project Partners:
Executive Summary

This deliverable provides a description of the means to integrate vApps with real factories. This integration is enabled through Device Drivers and API Connectors: vf-OS components that serve as interoperability mechanisms with factory assets. Device Drivers are designed to exchange data with physical manufacturing devices, whereas API Connectors are designed to exchange data with software services or applications. These components have many commonalities, since their functionality is fundamentally the same regardless of the specific factory asset they interconnect with.

The vf-OS IO Toolkit provides common functionalities for seamless/open access and smart virtualisation of the factory resources. At design time, the vf-OS IO Toolkit provides a set of tools for the development of Device Drivers and API Connectors that virtualise the factories’ assets in the vf-OS platform. At runtime, the functionalities implemented in Device Drivers and API Connectors enable the connection of sources of information to any vApp through interactions with other vf-OS components, such as Middleware or Data Storage. Security is also an aspect of this WP to ensure the protection of the factory assets and adherence with the vf-OS Security requirements. This document first elaborates these concepts to provide some context and presents the development roadmap and the implementation strategy to be applied in the development tasks. It then reports the status and availability of the different WP4 Software Deliverables. Finally, the document annexes the specifications of the vf-OS IO Toolkit, describing the key features that enable the fast and efficient development of Device Drivers and API Connectors.

As all other deliverables in this WP are of type “OTHER”, this document also serves as an Umbrella document to report the progress and results of the different software deliverables in WP4.

Software deliverables (type “OTHER”) are available for download from the vf-OS repository. The access details are included in Section 0. The installation and usage instructions are available at the corresponding annexes of this document. This iteration covers the following components: IO Toolkit and Security Command Centre.

This document is structured according to the different software deliverable versions at M18, M24, M30, and M33. Each section describes the planned activities, progress and next activities for the different WP4 tasks in the corresponding reporting period. In this period, M12-M18, this document reports about the following tasks:

- T4.1 – Device Drivers/API Toolkit
- T4.2 – Device Drivers
- T4.3 – API Connectors
- T4.4 – Security & Data Access
# Table of Contents

0. Introduction ................................................................................................. 1

1. Context ........................................................................................................... 5
   1.1 Positioning in Overall vf-OS Architecture ............................................... 5
   1.2 Key Concepts ............................................................................................ 7
   1.3 Implementation Strategy ........................................................................... 10
   1.4 Roadmap ................................................................................................... 11
      1.4.1 IO Toolkit Development Roadmap ....................................................... 13
      1.4.2 Device Drivers Development Roadmap ............................................... 14
      1.4.3 API Connectors .................................................................................. 16
      1.4.4 Security ............................................................................................... 18

2. IO Toolkit (T4.1) .......................................................................................... 21
   2.1 Scope ........................................................................................................... 21
   2.2 M12 Report ................................................................................................. 21
      2.2.1 Planned Activities .............................................................................. 21
      2.2.2 Progress ............................................................................................... 21
      2.2.3 Next Activities ................................................................................... 22
   2.3 M18 Report ................................................................................................. 22
      2.3.1 Planned Activities .............................................................................. 22
      2.3.2 Progress ............................................................................................... 22
      2.3.3 Next Activities ................................................................................... 22
   2.4 M24 Report ................................................................................................. 22

3. Devices Drivers (T4.2) ............................................................................... 23
   3.1 Scope ........................................................................................................... 23
   3.2 M18 Report ................................................................................................. 23
      3.2.1 Planned Activities .............................................................................. 23
      3.2.2 Progress ............................................................................................... 23
      3.2.3 Next Activities ................................................................................... 23
   3.3 M24 Report ................................................................................................. 23
      3.3.1 Planned Activities .............................................................................. 23
      3.3.2 Progress ............................................................................................... 24
      3.3.3 Next Activities ................................................................................... 24
   3.4 M30 Report ................................................................................................. 24
   3.5 M33 Report ................................................................................................. 24

4. API Connectors (T4.3) ................................................................................ 25
   4.1 Scope ........................................................................................................... 25
   4.2 M18 Report ................................................................................................. 25
      4.2.1 Progress ............................................................................................... 25
      4.2.2 Next Activities ................................................................................... 25
   4.3 M24 Report ................................................................................................. 25
      4.3.1 Planned activities ............................................................................... 25
      4.3.2 Progress ............................................................................................... 25
      4.3.3 Next Activities ................................................................................... 26
   4.4 M30 Report ................................................................................................. 26
   4.5 M33 Report ................................................................................................. 26

5. Security (T4.4) ............................................................................................ 27
   5.1 Scope ........................................................................................................... 27
   5.2 M18 Report ................................................................................................. 27
      5.2.1 Planned Activities .............................................................................. 27
Conclusions ................................................................. 29

Annex A: History .......................................................... 30
Annex B: References ....................................................... 31
Annex C: IO Toolkit Specifications ...................................... 32
  C.1. IO Toolkit Features Specifications ............................. 32
     C.1.1. Top Interfaces ............................................... 32
     C.1.2. Logic .......................................................... 33
     C.1.3. Bottom Interfaces ........................................... 34
     C.1.4. Containerisation ............................................. 36
     C.1.5. Development tools ......................................... 37
     C.1.6. IO Toolkit Security .......................................... 37
Annex D: IO Toolkit R1 Software Packages .......................... 40
  D.1. IO Toolkit Generator Software Package .................. 40
     D.1.1. IO Toolkit Generator Requirements .................... 42
     D.1.2. IO Toolkit Generator Installation ...................... 42
     D.1.3. IO Toolkit Generator Execution ......................... 43
  D.2. IO Toolkit Templates Software Package ................ 44
     D.2.1. IO Toolkit Templates Requirements ................... 44
     D.2.2. IO Toolkit Templates Installation ..................... 46
  D.3. IO Toolkit Device Drivers SDKs Software Package .... 46
     D.3.1. IO Toolkit Device Drivers SDKs Requirements ...... 46
     D.3.2. IO Toolkit Device Drivers SDKs Usage ................. 47
     D.3.3. Empty-Driver SDK ......................................... 47
     D.3.4. OPC UA SDK ............................................... 48
  D.4. IO Toolkit API Connectors SDK Software Package ...... 49
     D.4.1. IO Toolkit API Connectors Requirements ............ 49
     D.4.2. IO Toolkit API Connectors SDK Usage ................. 49
Annex E: Security Command Center R1 Software Packages ..... 51
  E.1. Identity Management Service Software Package ......... 51
     E.1.1. Identity Management Service Requirements ........ 51
     E.1.2. Authentication Service Installation ................... 51
     E.1.3. Authentication Service Execution ..................... 53
  E.2. Authorisation Policy Decision Point Generic Enabler ...... 53
     E.2.1. Authorisation Policy Decision Point Generic Enabler Requirements .... 53
     E.2.2. Authorisation Policy Decision Point Generic Enabler Installation .... 53
     E.2.3. Authorisation Policy Decision Point Generic Enabler Execution .... 54
0 Introduction

0.1 vf-OS Project Overview

vf-OS – virtual factory Open Operating System – is a project funded by the H2020 Framework Programme of the European Commission under Grant Agreement 723710 and conducted in the period October 2016 until August 2019. It engages 14 partners (Users, Technology Providers, Consultants and Research Institutes) from 7 countries with a total budget of circa 7.5M€. Further information can be found at www.vf-OS.eu.

The World is facing the fourth industrial revolution based on ICT, specifically architectures and services, as key innovation drivers for manufacturing companies. Traditional factories will increasingly be transformed into smart digital manufacturing environments, but currently the full potential for ICT in manufacturing is far from being fully exploited. Factories are complex systems of systems and there is a need to develop a platform on which future manufacturing applications can be built. Examples of platforms exist in some industrial sectors but there is a lack of cross cutting platforms based on open standards for creating an ecosystem for cooperative innovation. Innovative open platforms to attract talent from solution developers and to provide accessible manufacturing smart applications to European SMEs are examples of the kind of solutions being sought.

The goal of vf-OS is to develop an Open Operating System for Virtual Factories composed of a kernel, application programming interface, and middleware specifically designed for the factory of the future. An Open Applications Development Kit (OAK) will be provided to software developers for deploying Manufacturing Smart Applications for industrial users, using the vf-OS Manufacturing Applications Store all operated through a Virtual Factory Platform.

The Virtual Factory Platform is an economical multi-sided market platform with the aim of creating value by enabling interactions between four customer groups:

- **Software Developers** (independent or within individual manufacturers) which will build Manufacturing Apps either through innovation or from manufacturing user demand
- **Manufacturing and Logistic Users** which will explore the marketplace for already created solutions, ready to be run on the vf-OS
- **Manufacturing and Logistics Solutions Providers** which will provide ICT interfaces and manufacturing connections
- **Service Providers** (vf-OS innovators and third parties) will make available services (hosting, storage, connected cloud services, etc.) including those based on developed solutions
The Virtual Factory Platform will provide a range of services to the connected factory of the future to integrate better manufacturing and logistics processes. Manufacturing Applications Store will be open to software developers who, using the free Open Applications Development Kit provided, will be able to quickly develop and deploy smart applications to enable and optimise communication and collaboration among supply networks of all manufacturing sectors in all the manufacturing stages and logistic processes.

vf-OS aims to become the reference system software for managing factory related computer hardware and software resources and providing common services for factory computational programs. This operating system will be the component of the system software in a real factory system where all factory application programs will run.

0.2 Deliverable Purpose and Scope

The main purpose of this document is to provide a centralised report of all the activities in WP4, together with the IO Toolkit Development Specifications. The scope of the deliverable can be defined as:

- Describe and motivate the functionalities of the software deliverables in WP4 and their corresponding development roadmap / implementation strategy
- Document the status and progress of the software deliverables in WP4
- Provide the development specifications for the IO Toolkit

0.3 Target Audience

This deliverable is primarily aimed at WP3 and other partners, but it can be useful for any audience interested in interoperability of open platforms with manufacturing devices, services, and applications. This includes mainly software developers, members of the scientific and industrial communities, as well as other publicly funded projects, which may also be interested in collaborating with vf-OS.

0.4 Deliverable Context

This document is focused on the reports of the development activities of WP4. The activities of WP4 are dependent upon deliverables of other WPs as follows:

- **Vision Consensus (D1.1):** The common understanding of the project vision and the overall aims and objectives of the project. Available now
- **Global Architecture Definition (D2.1):** High-level global architecture of vf-OS with all the first level modules and different interactions, both at service calls and data exchange. Available now
- **Functional Specification & Mockups (D2.2):** Definition of the functionalities and behaviours of all vf-OS components. Available now
- **Requirements Specifications (D1.5):** The requirements specifications for the vf-OS Platform and the pilot vApps. Available now
- **Validation Scenarios (D8.1ab):** The methodology that establishes a reference and technical specification for the interactions between different vf-OS customer groups developing and using new vApps. Available now
- **Validation Scenarios (D8.1cd):** A series of reports providing the results of the evaluation of pilot applications at different stages of the project, following the
functional specifications defined in this document. D8.1c available in M24, D8.1d available in M36

- **Pilot 1: Manufacturing & Logistic – Automation (D8.2abc):** A series of demonstrators of the applications developed for pilot domain 1. D8.2a available in M24, D8.2b available in M30, D8.2c available in M36
- **Pilot 2: Construction – Industrialisation (D8.3abc):** A series of demonstrators of the applications developed for pilot domain 2. D8.3a available in M24, D8.3b available in M30, D8.3c available in M36
- **Pilot 3: Manufacturing Assembly – Collaboration (D8.4abc):** A series of demonstrators of the applications developed for pilot domain 3. D8.4a available in M24, D8.4b available in M30, D8.4c available in M36

### 0.5 Document Structure

This deliverable is broken down into the following sections:

- **Section 1: Context:** An introduction to this deliverable in the context of the vf-OS vision and overall architecture. This section also presents the key concepts, the implementation strategy, and the development roadmap
- **Section 2: IO Toolkit (T4.1):** A report of task T4.1 for its reporting periods M12, M18, and M24
- **Section 3: Devices Drivers (T4.2):** A report of task T4.2 for its reporting periods M24, M30, and M33
- **Section 4: API Connectors (T4.3):** A report of task T4.3 for its reporting periods M24, M30, and M33
- **Section 5: Security (T4.4):** A report of task T4.4 for its reporting periods M18 and M24
- **Section 6: Conclusions:** Final conclusions and remarks

- **Annexes:**
  - Annex A: Document History
  - Annex B: References
  - Annex C: IO Toolkit Specifications
  - Annex D: IO Toolkit R1 Software Packages
  - Annex E: Security Command Center R1 Software Packages

### 0.6 Document Status

This document is listed in the Description of Action as ‘PU’ (“Public”).

### 0.7 Document Dependencies

This document is the second part of a series of deliverables describing the IO Toolkit specifications and the availability and status of software deliverables in WP4. The previous iteration at M12 covered the Specifications. This iteration covers the status of software deliverables at M18 and the next planned activities. Other iterations will be delivered in M24, M30, and M33.
0.8 Glossary and Abbreviations

A definition of common terms related to vf-OS, as well as a list of abbreviations, is available in the supplementary and separate document “vf-OS Glossary and Abbreviations”.

Further information can be found at [http://www.vf-OS.eu/glossary](http://www.vf-OS.eu/glossary)

0.9 External Annexes and Supporting Documents

All software deliverables (type “OTHER”) are available from the vf-OS repository with read only access details as follows:

- URL: [https://owncloud.ascora.de/index.php/s/smJN0KDHyewJp9a](https://owncloud.ascora.de/index.php/s/smJN0KDHyewJp9a)
- Password: 123vfOS

0.10 Reading Notes

None
1 Context

1.1 Positioning in Overall vf-OS Architecture

The vf-OS IO Toolkit consists of a series of development resources implementing functionalities common to all vf-OS Components exchanging data with factory assets (Device Drivers and API Connectors) and other services and systems (External Services and Enabler Framework). These components are referred to as IO Components.

Fundamentally, IO Components connect new assets (devices, services, applications, etc.) to the vf-OS Platform and the IO Toolkit provides functionalities to guarantee the interoperability of such assets. Thus, the IO Toolkit ensures that IO Components are compatible with vf-OS and can be used in vApps through known interfaces.

As shown in Figure 1, at runtime, the IO Toolkit implements core functionalities to integrate components into the vf-OS Platform and enable the exchange of data with other components (Security, Registration, middleware endpoints, etc.).

At design time, the IO Toolkit provides additional development resources to Manufacturing and Logistic Providers wishing to develop new IO Components in order to connect new manufacturing resources and services to vf-OS.

For both scenarios, it is important to understand the relationship between the OAK Toolkit and the IO Toolkit. The OAK Toolkit is for the development of vApps by Software...
Developers, whereas the IO Toolkit is for the development of new IO Components. In the context of connecting vf-OS to manufacturing assets:

- The OAK SDK provides programming interfaces to access data and services from the IO Components available in the vf-OS Platform, so that they can be used within vApps.
- The IO Toolkit provides development resources to facilitate the development of new interoperability mechanisms with devices and on-premise software applications and services by third parties. The IO Toolkit development resources consist of:
  - The IO Toolkit SDK: Implementation of the IO Toolkit functionalities in a given technology. The IO Toolkit functionalities enable interaction with other vf-OS components and are further described in Sections 1.2 and Annex C of this document.
  - Documentation: Document describing the functionalities of the IO Toolkit and containing examples, best practices, and guidelines to facilitate the development of new components.

The IO Toolkit is used to develop a new Device Driver to interact with Siemens automation equipment. The OAK SDK allows Software Developers to use data from this and other installed Device Drivers in their applications.

Post project there can be various IO Toolkit SDK implementations for prominent technologies to facilitate the development of new components. For instance:

- The NodeJS IO Toolkit SDK is the NodeJS¹ implementation of the IO Toolkit and integrates different modules implementing all the IO Toolkit functionalities.
- The Java IO Toolkit is the Java implementation of the IO Toolkit and consists of different libraries implementing the various IO Toolkit functionalities.

Within the project, the development platform selected to implement the IO Toolkit is NodeJS. This was selected based on performance, the technical requirements of the pilots, and the available open source resources.

The long-term vision is to provide, within the IO Toolkit, development resources to facilitate the development of virtually any device or manufacturing asset. During the project, although this vision is taken into account, the IO Toolkit is primarily focused on the development of the IO Components needed in the pilots. For this reason, in the current architecture, the IO Toolkit is not regarded as part of, or an extension of, the OAK Toolkit.

There is an analogy between the use of the OAK Toolkit for the development of vApps and the use of the separate IO Toolkit for the development of IO Components such as Device Drivers. In the IO Toolkit, Manufacturing and Logistic Service Providers developing a new IO Component will find development resources (eg NodeJS modules) implementing the core functionalities that need to be implemented in any IO Component. These functionalities are listed and described in Annex C.: Manufacturing and Logistic Service Providers can extend these core functionalities with software developed specifically to connect to a set of manufacturing assets with a given technology.

¹ https://nodejs.org
New IO Components can be published in the Marketplace just as any other vf-OS Asset (see vApp Development in Figure 2). The IO Toolkit provides development resources (documentation, examples, etc.) to facilitate the editing of a Manifest file used to describe the IO Component in the Marketplace and allow Software Developers to use it within a vApp. The OAK SDK provides developers programming interfaces to access data and services of IO Components. The OAK SDK obtains the Manifest files of IO Components from the Marketplace and use this information to provide programming interfaces that allow Software Developers to use specific IO Components within vApps.

Software Developers can use any available IO Component in the development of vApps with the vf-OS OAK SDK (See IO Component in Figure 2). The vApp can implement mechanisms to discover installed IO Components and automatically configure connections to specific devices or software services in runtime (similar to Plug-and-Play mechanisms in traditional operating systems). If a vApp uses a specific IO Component, then the Manifest file of the vApp describes this dependency, so that users can be informed before installing the vApp. In addition to this mechanism, the OAK SDK provides programming interfaces to call services of any IO Component instance available in a particular vf-OS installation, using the Device Driver and API Connector APIs defined in the architecture. Software Developers can thus develop functionalities that interact with any manufacturing asset that is connected to vf-OS via an IO Component. In this case, the OAK SDK provides programming interfaces to call the Configuration APIs of Device Drivers and API Connectors. Using these programming interfaces Software Developers can develop UIs for users to configure within the vApp the connection to a specific IO Component.

1.2 Key Concepts

There are many different manufacturing assets (software systems and devices) involved in the creation of products, from the R&D phase, through the manufacturing and assembling processes, until shipping of the final product to the customer. The IO Toolkit, Device Drivers, and API Connectors offer resources for fast data integration from various factory
assets. These components enable the development of vApps aiming to support data-driven decision-making in supply networks and to interact with factory software and devices. Figure 3 shows an overview of how a Device Driver and/or an API Connector, would be organised internally. The various parts are described after Figure 3.

Figure 3: I/O components functionalities

In brief, these concepts are defined as follows:

- **IO Toolkit**: Development resources (source code and documentation) for Software Developers aiming to develop Device Drivers and API Connectors. These components have the same basic functionalities, for which the IO Toolkit provides software libraries and implementation documentation (examples, best practices and guidelines and library descriptions). As highlighted before, the functionalities of the IO Toolkit libraries can be classified into four different groups:
  - **Top interfaces**: Core functionalities to interact with other vf-OS components. These include client implementations of the services provided by the middleware and the vf-OS kernel and implementation of the endpoints of the REST APIs defined for Device Drivers and API Connectors. Top interface functionalities are common to all IO Components.
  - **Logic**: Functionalities of the internal logic of the component. Different IO Components will have different internal Logic implementations, but they all can benefit from a set of common functionalities (eg internal storage) useful to implement the internal logic of the component.
  - **Bottom interfaces**: Functionalities to integrate with factory assets. Bottom interfaces are used to interact with the factory asset and are specific of every IO Component.
  - **Security**: Functionalities to comply with the vf-OS Security and Privacy Concepts. Security functionalities are common to all IO Components.

- **Device Drivers**: Device Drivers are IO components specifically implemented to interact with physical industrial devices. Besides the Top Interfaces and Security
functionalities needed to integrate with the vf-OS Platform, Device Drivers implement inner functionalities to integrate device sensor data and to execute actuation commands. Furthermore, any Device Driver implements a specific Bottom Interface implementation with a specific technology to integrate with a set of physical devices.

For instance, a Device Driver to integrate devices using the OPC UA protocol will implement Bottom Interfaces based on this technology to interact with automation devices. Device Drivers can be deployed on a runtime platform with a hardware connection to the device sensors and actuators. In this case, the specific Bottom Interfaces implementation is based on system calls to the driver of the sensor/actuator connection hardware.

Alternatively, Device Drivers can be deployed on a runtime platform with a network connection to devices or sensors that implement a specific industrial communications technology. Here, the specific Bottom Interfaces implementation is based on the specific industrial communications technology used. Nevertheless, in both cases, the Device Driver component functionalities are primarily the same and most of the functionalities that need to be implemented are provided by the IO Toolkit.

- **API Connectors:** API Connectors act as interfaces to (legacy) software in factories (eg corporate applications such as ERPs), so that vf-OS Components can exchange data with them. An API is a method provided by the legacy software vendor to allow third-party developers to write programs that interface easily with their software. Thus, an API defines exactly the methods for a third-party software program to interact with the legacy software.

  The APIs can be exposed in several ways which include: COM objects, DLL and .H files in C/C++ programming language, JAR files or RMI in Java, XML over HTTP, JSON over HTTP, etc. As with Device Drivers, some of these methods require that an API Connector is deployed in the same runtime as the legacy software. For instance, Desktop applications such as spreadsheets and word documents use VBA and COM-based APIs that need to be called locally in the desktop application runtime. In other cases, interactions involve sending data over a network using web services (eg based on REST or SOAP).

- **IO Components:** The concept of IO Components is introduced for the sake of clarity to refer to both Device Drivers and API Connectors. Thus, IO Components offer a simple way of connecting to, integrating with, and extending physical devices and software systems. To enable this connectivity, IO Components should respect the following principles:
  
  - **Platform-Centric:** The IO Component need to mediate between the clean REST API format presented to the vf-OS Platform and the complex formats used in the Bottom Interface. Thus, the value of the IO Component is hiding the complexity for the vf-OS Platform component consumers, but still to be valuable and relevant to connect with manufacturing assets. In summary, The IO Components are made for vf-OS, in order to provide interoperability with factory assets, not to provide additional functionality to the specific device or software system.
  
  - **Documented:** The IO Components need to be documented to facilitate its installation, configuration, and usage. This makes it necessary to provide knowledge of the underlying technology used to interconnect to the manufacturing assets, eg for configuration purposes.
• **Secure and Compliant:** The IO Component needs to ensure that it can only be accessed by authenticated and authorised consumers and that it is compliant with the best practices and with security regulations ruling vf-OS.

• **Performant:** For Manufacturing and Logistic Users, the performance of the IO Component is an important requirement. For Software Developers, a highly performant IO Component allows them to build responsive vApps with a good end user experience.

• **Interoperable and Standard-Conform:** IO Components should apply relevant standards and follow industry conventions to foster the interoperability of vf-OS.

• **Reusable:** The design of an IO Component should take into consideration reusability, so that it can easily adapt to different use case scenarios. For instance, a Device Driver designed to interact with devices using a specific communication protocol can be reused to develop a more specialised Device Driver that interacts with a specific device model using the same communication protocol. In this sense, the IO Component itself should also be built from reusable components.

• **vf-OS Versioning:** IO Components need to be compatible with the vf-OS versioning system. Even though IO Components can be developed in an agile way, once they are published and used, all the agility is left behind, and the given version of the API becomes immutable. If new features do not allow for backward compatibility, a new IO Component or a new IO Component version needs to be created.

### 1.3 Implementation Strategy

The Implementation strategy for the IO Toolkit and the different IO Components is based on the principles of:

• **User-centred design:** Building on the use cases to be implemented in the 3 project pilots

• **Software reusability:** Implementing common functionalities for all IO Components in the most effective way.

IO Components that need to be developed for the pilots have been defined in D8.1b Validation Scenarios. The development of the IO Toolkit is guided by the requirements of this IO Components. Annex C identifies the core functionalities common to all IO Components. These functionalities are implemented in T4.1 as part of the IO Toolkit SDK used to develop the pilot IO Components, so that the generated code is as easy to produce as possible. In the development of IO Components, existing open source solutions to connect to manufacturing assets may be used to further improve productivity. The IO Toolkit SDK will be further extended to achieve the target performance indicators defined for the assessment of the implementation of the vf-OS IO. Specifically, it will support the implementation of at least 20 of the most popular interconnection protocols to exchange data with manufacturing assets. Post project, other IO Components can be added, and other IO Toolkit SDKs can be developed to foster the development of new IO Components.

Moreover, IO Components are distributable, installable components and need to be encapsulated into executable assets within the vf-OS Platform. To enable this, the Execution Services of the Platform provides a standardised Docker environment. Hence, the IO Toolkit must contain tooling aimed in assisting developers encapsulate the IO
Components. Through this Docker containerisation the Drivers and APIs become assets which can be:

- Placed in the Marketplace
- Integrated into vApps
- Downloaded to and run in the local execution environment
- Signed and version tracked, for security and trust purposes.

Finally, it is important to note that the development of the IO Toolkit and the IO Components is subject to the quality requirements and standards defined in T11.5 Technical Set Up and Quality Toolset for vf-OS components.

1.4 Roadmap

This section analyses the different vApps in the project pilots and identifies the necessary IO Components and their implementation roadmap.

WP4 tasks focus on the development of software to integrate with manufacturing assets thus providing interoperability mechanism between the factory and vf-OS. The focus is to provide support to the pilot use cases, but also to develop a comprehensive, well-documented IO Toolkit that can be used to effectively develop other IO Components. Figure 4 shows the roadmap for the development activities within WP4.

D2.2 Functional Specification and D2.3 Technical Specification define the functional and technical specifications for Device Drivers and API Connectors and specify how they interact with other vf-OS Components and Logic functionalities needed to implement IO Components. Following these specifications, the IO Toolkit needs to provide IO Components with the Top Interface functionalities to interconnect with other vf-OS components and Logic functionalities needed to implement IO Components. These functionalities will be implemented in the first release of the IO Toolkit. The second release will fix bugs and implement missing features that were not needed to be implemented in the first release of Device Drivers and API Connectors.

Section 1.4.1 contains further information about the development roadmap of the IO Toolkit.
D8.1b Validation Scenarios defines the functional specifications for the different pilot use cases. From these, it is possible to check the story maps and identify the IO Components that need to be developed to support the interoperability of the different vApps with physical devices and business software in the pilots. Story Maps organise user stories in a way that makes this identification process straightforward. Since every column in the Story Map represents a different interaction between two actors, the IO Components (and their corresponding user stories) needed for connecting with devices and applications are clearly depicted in the Story Maps. Subsequently, the user stories for the development of the identified IO Components can be matched against the user stories in the functional specifications of Device Drivers and API Connectors. These user stories are found in D2.2 and determine the entire product backlog, functionalities, and roadmap for the implementation of the functionalities of these components.

Figure 5 shows the different IO Components identified in the story maps of the pilot vApps. These IO Components implement the connections to manufacturing devices and legacy services depicted in the corresponding story maps. The releases are defined according to the pilot vApp scheduling described in D8.1a.

<table>
<thead>
<tr>
<th>IO Component</th>
<th>Releases</th>
<th>Pilot</th>
<th>vApps</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS OPC UA Driver</td>
<td>M24, M33</td>
<td>1</td>
<td>vfFailurePrevention, vfFailureManager, vfProductionFollowUp</td>
</tr>
<tr>
<td>Excel File Connector</td>
<td>M33</td>
<td>1</td>
<td>vfStockPolicies</td>
</tr>
<tr>
<td>ODBC Connector</td>
<td>M24, M30</td>
<td>2</td>
<td>vfOnSiteManager, vfSteelValidation, vfConcreteFeedback</td>
</tr>
</tbody>
</table>
The main objectives of the three project pilots regarding interaction with manufacturing assets can be summarised as follows:

- **Pilot 1**: Focuses on value added services offered by a manufacturing machine provider to its customers. Most of the applications in this use case use the TS OPC UA Driver specifically developed to connect to their Tabber Stringer Machine using one of the available communication protocols (OPC UA) in the PLC embedded in the machine. As presented in D8.1b, users wish to use Excel Files to integrate other data into the vApps.

- **Pilot 2**: Many functionalities are covered by FI-Ware Enablers, although an API Connector to integrate data from the user database has been identified in the pilot Story Maps.

- **Pilot 3**: Focuses on the collaboration between two different factories and requires several API Connectors and Device Drivers to interconnect different manufacturing assets involved (i.e., the ERP systems of both companies, CPS systems, and ad-hoc classification and manufacturing order sequence services).

### 1.4.1 IO Toolkit Development Roadmap

The IO Toolkit implementation provides software libraries to be (re)used in Device Drivers and API Connectors. As indicated above, the IO Toolkit has two major software releases R1 (M18) and R2 (M24). The IO Toolkit features and specifications are further described in Annex C; which provides a definition and motivation of the different development resources that are included in the IO Toolkit. The main objective of R1 is to provide the core functionalities needed to develop the first release of Drivers and API connectors. R2 on the other hand fixes bugs and provides additional functionalities for the second and third releases of IO Components, as well as functionalities to implement additional IO Components for leading manufacturing solutions. The IO Toolkit development resources which will be implemented in R1 are:

- REST API Server Composition (Libraries to compose the different REST APIs)
- Messaging client (Messaging Component services client implementation)
- Pub/Sub client (Pub/Sub Component services client implementation)
- Registration service client (Registration Service client)
- Logging service client (Logging Service client)
- Metadata composition (resources to compose the component metadata)
- Internal storage (development resources to build internal storage databases)
- Key-value store (development resources to build key-value stores)
- Edge computing (development resources for edge computing)
- API Access Control (API Access Control implementation)
- API Lifecycle management (API Lifecycle management implementation)
- Simulation (Simulation features)
• OPC UA libraries (development resources for OPC UA based Device Drivers)
• MS libraries (development resources for MS Office files API Connectors)
• LxP ERP (development resources to connect to the LxP ERP system used in Pilot 3)
• STEP file libraries (development resources for STEP file API Connectors)
• ODBC libraries (development resources for ODBC API Connectors)
• Sensor data read skeleton (Device Driver skeleton)

The features to be implemented in R2 are:
• MQTT libraries (development resources for MQTT based Device Drivers)
• Profinet libraries (development resources for Profinet based Device Drivers)
• ADS libraries (development resources for ADS based Device Drivers)
• Modbus libraries (development resources for Modbus based Device Drivers)
• Web Thing API libraries (development resources for Web Thing API Device Drivers)
• GPIO libraries (development resources for GPIO Device Drivers)
• MICA libraries (development resources for MICA Device Drivers)
• CoAP libraries (development resources for MICA Device Drivers)
• OData libraries (development resources for OData API Connectors)
• NAV ERP libraries (development resources for MS Navision API Connectors)
• SAP ERP libraries (development resources for SAP API Connectors)
• Liferay libraries (development resources for Liferay API Connectors)

1.4.2 Device Drivers Development Roadmap

This section describes the development roadmap for the core functionalities implemented in all Device Drivers (Device Drivers Core Functionalities) and the different Device Drivers that need to be implemented for the pilots listed in Figure 5.

1.4.2.1 Device Drivers Core Functionalities

Along with the specific drivers, the vf-OS Driver component packages a set of services and service endpoints that are common to every Device Driver implementation. Those are services such as driver management or instantiation of devices corresponding to specific drivers, together with the specific communication of Device Drivers with other vf-OS components. The user stories defined in D2.2. (and identified as DRUS001-25 in the bullets below) are classified according to the three releases R1(M24), R2 (M30), and R3 (M33) for Device Drivers highlighted in Figure 4. The list indicates which functionalities are already implemented in the IO Toolkit. For the remainder of the user stories, the IO Toolkit provides development resources. The user stories for the development of the different releases are as follows:

R1:
• DRUS001 Receive asynchronous data from device
• DRUS002 Push data on pub/sub
• DRUS006 Query devices / sensors
• DRUS007 Offers API for reading low latency stream data
• DRUS016 List existing devices already configured (Provided by IO Toolkit)
• DRUS010 Register device on vf-OS (Provided by IO Toolkit)
• DRUS011 Register asynchronous sensors of device (Provided by IO Toolkit)
• DRUS012 Register synchronous sensors of device (Provided by IO Toolkit)
virtual factory Operating System – www.vf-OS.eu

- DRUS019 Check device status
- DRUS020 Check sensor status
- DRUS021 List existing drivers of a specific vf-OS instance installation (Provided by IO Toolkit)
- DRUS025 Manifest reading (Provided by IO Toolkit)

R2:
- DRUS003 Store data in short-term historic database
- DRUS004 Polling process for reading sensors
- DRUS005 Edge computing (Provided by IO Toolkit)
- DRUS014 Configure computation of sensor data
- DRUS022 Filter list of drivers by Name
- DRUS023 Filter sensor readings according to their value
- DRUS024 Log messages from Device Driver components

R3:
- DRUS013 Configure short term historic data storage
- DRUS008 Read short-term data filtering by range of dates
- DRUS015 Configure device as command receiver
- DRUS017 Filter list of devices according to name and type
- DRUS018 Sort list of devices by columns
- DRUS009 Send command to device

1.4.2.2 TS (Tabber Stringer) OPC UA Driver

The TS (Tabber Stringer) OPC UA Driver is the Device Driver that connects vf-OS to the MASS Tabber Stringer machine, using the OPC UA\textsuperscript{2} communication standard. OPC UA is becoming the de-facto M2M standard for the exchange of data with industrial control systems in the shop-floor and it allows the porting of the driver to other physical devices with few changes. The TS Machine embeds a Beckhoff PLC that implements an OPC UA Server that can be used to collect data from the machine. The implementation roadmap of the TS OPC UA Driver is planned in two different releases R1 (M24) and R2 (M33). The user stories defined in D8.1b Annex C are identified as (USP11.14-29) in the bullets below. The user stories implemented in:

R1:
- USP11.23 USP12.20 Check TS machine connection status
- USP11.24 USP12.21 USP14.14 Connect to TS machine
- USP12.25 Subscribe to failure data
- USP12.26 Receive failure data

R2:
- USP11.28 Subscribe to anomaly data
- USP11.29 Receive anomaly data
- USP12.22 USP11.25 Subscribe to status data
- USP11.26 Receive status data

\textsuperscript{2} Link to wiki
• USP14.15 Collect production data

1.4.2.3 CPS Driver

The CPS Driver is a Device Driver to exchange data with the CPS’ used in APR and Tardy to control the manufacturing process quality.

The implementation of a CPS network in Pilot 3 (with APR and Tardy) aims to setup new automatic and adaptive manufacturing quality control processes. The CPS system implements several sensors:

• Position sensors detect product position in the output of a manufacturing machine
• High resolution cameras capture several pictures covering the 3D geometry of a product

The distributed decision system identifies the product to control from its geometry, quality tolerance expected, applies validation mechanisms, and decides the convergence of the entire manufacturing batch. The architecture is implemented using service technology to increase the reusability of the CPS network components.

The development roadmap of the CPS Driver is one release R1 (M33). Regarding the user stories defined in D8.1b Annex E, the CPS Driver will implement in:

R1:
• USP34.01 Subscribe to anomalies detected by CPS

1.4.3 API Connectors

This section describes the development roadmap for the core functionalities implemented in all API Connectors (API Connectors Core Functionalities) and the different API Connectors that need to be implemented for the pilots listed in Figure 5.

1.4.3.1 API Connectors Core Functionalities

API core functionalities provides components for API definition, API lifecycle as well as API user management. The user stories identified in D2.2 (APUS001-13) are classified according to three releases R1 (M24), R2 (M30), and R3 (M33).

The user stories implemented in:

R1:
• APUS001: Create API
• APUS002: Update API Specifications
• APUS006: Register Application
• APUS007: Subscribe to API
• APUS008: Generate APIs access tokens
• APUS010: Publish API

R2:
• APUS004: List APIs
• APUS005: Check API Documentation
• APUS009: Consult feedbacks and rating
• APUS0012: Create user
• APUS013: Assign role

R3:
• APUS003: Create new API version
• APUS011: API Monitoring

1.4.3.2 Excel File Connector
The Excel File Connector is an API Connector to import data with other software systems through Excel files. The Excel File Connector parses spreadsheets in various Excel Workbook formats (eg XLS, XLSX) as well as other related formats (eg CSV). The development of the Excel File Connector is planned in one release R1 (M33) and will implement the following user stories defined in D8.1b Annex C:
• USP13.09 Import TS machine spare part information from uploaded file
• UPS13.18 Import customer spare part information from uploaded file

1.4.3.3 ODBC Connector
The ODBC Connector allows the exchange of data with databases running on premise using a standard ODBC API. The ODBC standard provides database independence allowing the re-use of this API Connector in many different use case scenarios. The SQL Connector is primarily used in the scope of Pilot 2 and is planned in two releases: R1 (M24) and R2 (M30). The user stories for Pilot 2 are defined in D8.1b Annex D. The user stories for the development of R1 of the ODBC Connector to be used in this pilot are:
• USP24.03 Collect slump test data

The user stories for the development of R2 are:
• USP23.05 Collect schedule data
• USP22.07 Collect validation data

1.4.3.4 STEP File Connector
The STEP File Connector will be specifically developed to import data from STEP files describing 3D models, allowing the use of this format in vApps for collaborative engineering. The STEP File Connector provides the means for vApps to extract information from the STEP file about the different features (parts of the 3D model) therein. D8.1b Annex E provides additional information about the expected functionality of the STEP File Connector and defines the user stories for development. The Development of the STEP File Connector is planned in one release R1 (M24) and will implement the following user stories for the pilot:
• USP31.12 Import STEP File

1.4.3.5 Classification Service Connector
The Classification Service Connector fetches the production plan from the ERP systems of both APR and TARDY. For common products, this connector allows to upload generated Manufacturing Orders at non-confirmed status. Then, a common and optimised manufacturing order sequence is generated to guarantee the coherence of production activities between both companies. This plan will be confirmed later by both production
managers. The development of the Classification Service Connector is planned in one release R1 (M24) and will implement the following user stories defined in D8.1b Annex E:

- USP31.14 Call classification service

1.4.3.6 LxP ERP Connector

The LxP ERP Connector is designed to exchange data with the LXP³ ERP software used by pilot 2 owners’ APR and TARDY. The development is planned in two releases: R1 (M30) and R2 (M33). The LxP Connector will implement some of the user stories defined for pilot 3 in D8.1b Annex E.

The user stories for the development of R1 are:

- USP32.15 Collect production details from ERP systems

The user stories for the development of R2 are:

- USP35.01 Collect manufacturing orders from LXP ERP

1.4.3.7 Scheduler Service Connector

The Scheduler Service Connector imports data from the ad-hoc scheduler service which determines manufacturing order sequences for production in APR and TARDY. This service generates manufacturing order sequences according to the production plans of both companies the requirements of collaborative projects, to guarantee the coherence of production activities between both companies. The Development of the Scheduler Service Connector is planned in one release R1 (M30) with the following user stories defined in D8.1b Annex E:

- USP33.01 Call external scheduler service

1.4.4 Security

The following sections describe in more detail the progressive implementation plan for the different technologies included in the security concept. D2.4 Holistic Security and Privacy Concept describes in detail the technologies used in the vf-OS Security Command Centre which implements the security processes used by vf-OS Assets in runtime and represents the largest attack surface in the vf-OS Platform. The different user stories for the development of the Security Services that support the Security and Privacy Concept are defined in D2.2 Functional Specifications and identified as (SCUS013-23) in the bullets below. The development roadmap is divided into two releases. The first release comprises the core security technologies:

- Identity Management, provided by the Identity Service
- Security Policy Management, provided by the Authentication Service
- Continuous Security Policy Monitoring

The second release provides upgraded versions of these core security technologies. As mentioned, the vf-OS Security concept will provide development resources as part of the IO Toolkit to ensure that IO Components comply with the vf-OS security concept. These development resources (guidelines and software resources) are described in Annex C:

---

1.4.4.1 Identity Service

The Identity Service is the cornerstone of the security in vf-OS and is aimed to provide capabilities for managing the vf-OS Role Based Access Control (RBAC) Identity Management System through a REST API. The Identity Service supports create, request, update, delete (CRUD) operations in roles, user groups, and users through a REST API. The Identity Service is planned in two releases R1 (M18), providing an initial implementation, and R2 (M24), fixing bugs of the first release. The Identity Management System implements the following functionalities:

- Role Management (List, Create, Request, Update, Delete, Assign to Group, Remove from Group)
- Group Management (List, Create, Request, Update, Delete, List Users, Add User, Remove User, Check whether user belongs to group)
- User Management (List, Create, Request, Update, Delete, List Groups, Change Password)

The user stories for the development of these operations in R1 are:

- SCUS013 User Operations
- SCUS014 Group Operations
- SCUS015 Role CRUD operations

1.4.4.2 Policy Administration

The Policy Administration API is aimed at providing services for creating new security policies, modifying them, or removing existing security policies. Security policies specify who is the actor, what is the subject, which is the operation to be accomplished, and what is the security decision (Accept/Deny). The Policy Enforcer Proxy subsystem ensures that all defined security policies are met. The implementation of the Policy Administration REST API is planned in two different releases R1 (M18), which is the first fully functional release, and R2 (M24) which fixes bugs encountered in the R1. The following bullets summarise the main functionalities of the Policy Administration API:

- Add Policy
- Get Policy and versions
- Remove Policy and Versions
- Set Policy Decision

The user stories for the development of these operations in R1 are:

- SCUS016 Create Asset Security Policy
- SCUS017 Read Asset Security Policy Read
- SCUS018 Update Asset Security Policy
- SCUS019 Delete Asset Security Policy

1.4.4.3 Continuous Security Policy Monitoring

As stated in D2.4, the vf-OS Security Concept supports the definition of security profiles based on the Security Content Automation Protocol System (SCAP) specification. SCAP uses XML files to define security profiles and sandboxes with security policies for both

4 https://scap.nist.gov/index.html
assets as well as the native OS. The SL1 Profile of the IEC 62443 standard, which represents the most up-to-date security profile in the public repositories, are the basis of the vf-OS system. In addition to this, vf-OS will add a new vf-OS specific security profile extending the SL1 Profile. As a result, the first release R1 (M18) of the SCAP Profiles will comprise three different security profiles:

- OS native security profile
- IEC 62443 SL1 Security Profile
- vf-OS Security Profile to guarantee the recommendations for Secure Data in Transit

Based on these profiles the vf-OS Security Command Centre will support the definition of continuous monitoring policies based on these profiles. The user stories for the development of R1 are:

- SCUS020 Create Security Profile
- SCUS021 Read Security Profile
- SCUS022 Update Security Profile
- SCUS023 Delete Security Profile
2 IO Toolkit (T4.1)

The remainder of the document reviews the activities planned, achieved, and forecast for the rest of the WP4 deliverables (one section per task).

2.1 Scope

The IO Toolkit is the development framework available to Manufacturing and Service Providers wishing to generate new Device Drivers and API Connectors for industrial devices and business software packages. Device Drivers and API Connectors components will be accessible through the vf-marketplace and the OAK Studio, empowering vApps with new capabilities to consume or interact with a wider range of shop floor devices and line of business systems.

2.2 M12 Report

2.2.1 Planned Activities

The main activities planned for this period were:

- Definition of the integration of the IO Toolkit within the vf-OS OAK and the vf-OS Studio
- Define the key building blocks that are meant to compose the IO Toolkit
- Analyse the validation scenarios and identify the requirements for the Device Drivers and API Connectors that will allow pilot vApps to interact with factory assets
- Define the functional specifications for the first release of the IO Toolkit planned for M18
- Provide guidelines for the development of the Device Drivers and API Connectors that need to be implemented from M18 using the IO Toolkit to provide support for the first release of pilot vApps (M18)

2.2.2 Progress

In the current reporting month (M12), the progress status of the task is:

- The analysis of the positioning of the IO Toolkit in the vf-OS Platform has been completed. A detailed description of the IO Toolkit, Device Drivers and API Connectors with related components is provided in Section 1.1
- Section 1.2 contains detailed definitions of the key concepts and building blocks supporting the IO Toolkit. These definitions provide the basis for the definition of the functionalities that need to be implemented at every release of the IO Toolkit and the different Device Driver and API Connector components to be released within the project
- Based on these key concepts, the analysis of the validation scenarios has been conducted and the roadmap for the different releases of the IO Toolkit, and the pilot Device Drivers and API Connectors have been completed. As a result, the implementation roadmap is described in Section 1.4. The roadmap establishes the prioritised device protocols to be developed on M24, M30 and M33 according with pilots’ needs
- The functional specifications for the first release of the IO Toolkit have been completed and are presented in Annex C: IO Toolkit Specifications. These functional
pecifications cover all the functionalities that need to be implemented in the IO Toolkit released in M18 to develop the Device Drivers and API Connectors to be released in M24.

- In addition to this, Annex C: IO Toolkit Specifications. Contains detailed development guidelines and test cases for the use of the IO Toolkit.

### 2.2.3 Next Activities

The next activities planned for the period until M18 are:

- Complete the development of the IO Toolkit covering all the functional specifications described in Annex C: IO Toolkit Specifications
- Functional specifications for the next release at M24

### 2.3 M18 Report

#### 2.3.1 Planned Activities

The planned activities planned for the period until M18 are:

- Complete the development of the IO Toolkit covering all the functional specifications described in Annex C: IO Toolkit Specifications
- Functional specifications for the next release at M24

#### 2.3.2 Progress

In the current reporting month (M18), the progress status of the task is:

- The IO Toolkit Generator R1, used to scaffold IO Components, which implements the functionalities described in Annex C. Annex D:.1 provides a description of this generator, how it covers the functional specification and the functional specifications for the next release at M24, together with installation and execution instructions.
- The templating framework for Device Drivers and APIs. As described in Annex D:.1, the IO Toolkit Generator creates a working copy of the binaries and configuration files of an IO Component. To accomplish this, the generator uses a component templating framework. The structure and notations of templates and the templates provided in R1 are described in Annex D:.2.

#### 2.3.3 Next Activities

The current version of the IO Toolkit generators creates a working copy of a Device Driver or API Connector. The planned activities for the next period are the validation and test of the functionalities of the generator in the pilot use cases. The tests will be useful to detect software bugs and to identify new requirements to improve and refine the IO Toolkit generator.

### 2.4 M24 Report

Content to be added at M24.
3 Devices Drivers (T4.2)

3.1 Scope

Device Drivers are vf-OS components capable of operating and controlling industrial devices. Device Drivers provide interfaces used by software applications to access hardware functions or API interfaces offered by industrial devices.

Device Drivers in vf-OS follow a standardised top-level interface and data-exchanged format that make them pluggable into the vf-OS architecture, while obfuscating the complex internal logic of industrial communication protocols and message formats.

3.2 M18 Report

3.2.1 Planned Activities

The main activities planned for this period are:

- Development of the Device Drivers Core Functionalities
- Development of the TS (Tabber Stringer) Driver

3.2.2 Progress

In the current reporting month (M18), the core functionalities to implement drivers and the functionalities used to connect to the Tabber Stringer are ready. Moreover, the TS (Tabber Stringer) Driver uses the vf-OS SDKs for OPC UA Driver development R1, which are used to connect to automation devices and systems using the OPC UA protocol. Annex D: includes details about the implementation and installation of this SDK within Device Driver templates, using the IO Toolkit generator. For the sake of clarity, these software deliverables are described in the same Annex D as the IO Toolkit Generator. Using this framework, developers need only to use the IO Toolkit Generator to create the source code of the Device Drivers to be used in the project pilot 1.

3.2.3 Next Activities

The main activities planned for the next period are:

- Development of the CPS Driver: The driver to be used in pilot 2 use cases
- Conduct Validation tests: Validate the Device Drivers in the pilot scenarios

It is expected that during the validation tests, the OPC UA Driver SDK R1 will need several adjustments to correct software issues and also to implement new functionalities or improvements that were not identified in the definition of the functional specifications. These fixes will be delivered in the OPC UA Driver SDK R2 release at M24.

3.3 M24 Report

3.3.1 Planned Activities

The main activities planned for this period are:

- Development of the CPS Driver
• Conduct Validation tests

3.3.2 Progress
Contents to be added at M24.

3.3.3 Next Activities
Content to be added at M24.

3.4 M30 Report
Content to be added at M30.

3.5 M33 Report
Content to be added at M33.
4  API Connectors (T4.3)

4.1  Scope

API Connectors are vf-OS Components designed to exchange data with software systems and applications in factories. An API Connector acts as an interface between two different applications so that they can communicate with each other. API Connectors define the exact methods for one software program to interact with other vf-OS Components.

4.2  M18 Report

The main activities planned for this period are:

• Development of the API Connectors Core Functionalities R1
• Development of the ODBC Connector R1

4.2.1  Progress

At M18, the core functionalities to implement API Connectors and the development resources to connect to legacy software using the ODBC protocol – the vf-OS SDKs for ODBC Connector development R1 – are available. Annex D.4 includes details about the implementation and installation of this SDK within API Connector templates. Similar to Device Driver implementations, the development of API Connectors using this technology consists of running the IO Toolkit generator with the available templates and SDKs. This development framework will support the development of the first ODBC based API Connectors to be used in the project pilots.

4.2.2  Next Activities

The main activities planned for the next period are:

• Development of the STEP File Connector R1
• Development of the Classification Service Connector
• Conduct Validation tests

It is expected that adjustments, improvements and new functionalities will arise during the validation tests. The implementation of these adjustments will result in the vf-OS ODBC Connector development R2 at M24.

4.3  M24 Report

4.3.1  Planned activities

The main activities planned for this period are:

• Development of the STEP File Connector R1
• Development of the Classification Service Connector
• Conduct Validation tests

4.3.2  Progress

Content to be added at M24.
4.3.3 Next Activities
Content to be added at M24.

4.4 M30 Report
Content to be added at M30.

4.5 M33 Report
Content to be added at M33.
5 Security (T4.4)

5.1 Scope

The main goal is the development, installation and deploying of the main components of the vf-OS Holistic Security and Privacy Concept described in D2.4. This process will be accomplished in two stages. In the first stage the main security components will be developed and deployed progressively. The sequence of the progress will be based on the criticality of the component, that is, the first components will be the identity management system and the policy administration system which include the first access control component of the system. Subsequently the security content automation protocol and the resource control systems will be developed and deployed.

The second stage will be reserved to all additional security component associated to the native OS as previously explained.

5.2 M18 Report

5.2.1 Planned Activities

The planned activities for this period are:

- Design and deploy a realistic scenario in order to validate the development of the Holistic Security and Privacy Concept
- Develop the core security functionalities (Identity Management, Security Policy Management, and Continuous Security Policy Monitoring)
- Definition of security profiles and sandboxes
- Conduct Validation tests

5.2.2 Progress

In this period, a first version of the Identity Management Service, and the Security Policy Management, based on Fi-Ware security enablers are available. The installation details are described in Annex E:

In addition to this, the following activities have been completed:

- Partial definition of the list of security profiles, corresponding to security services and host security
- Partial definition of sandboxing for the vf-OS platform
- Initial tests of Continuous Security Policy Monitoring Service implementations
- Interoperability checks for the upgrade of critical security components, since these components have significant and critical dependencies in new operating system releases

5.2.3 Next Activities

For the next period, the main planned activities are:

- Finalise the implementation of the Continuous Security Policy Monitoring Service
- Conduct validation tests
- Finalise the definition of the security profiles and vf-OS sandboxing
5.3 M24 Report

Content to be added at M24
6 Conclusions

The IO Toolkit is the basis for the development of Device Drivers and API Connectors and provides software libraries and implementation documentation for their development. Annex C: IO Toolkit Specifications contains the specifications of the IO Toolkit and describes the different software resources that it provides. This annex sets the basis for the development of IO Components, identifying common functionalities to all IO Components, specific functionalities that need to be implemented to comply with the project pilot requirements and the overall project objectives, and describing how the IO Toolkit resources will facilitate their development.

This document describes the results to date of the development tasks of WP4 providing an overview of the status at M18. The document presents the definitions and key concepts around vf-OS IO and builds a common understanding for all vf-OS partners, especially those involved in WP4. The document also presents the roadmap for the different development tasks in WP4 and reports the status at M18. The development roadmap provides the definition of the functionality of every release of the different IO Components to be implemented within the project.

Annex D: IO Toolkit R1 Software Packages describes the software deliverables for the development of Device Drivers and API Connectors, which is the result of the following tasks:

- T4.1 – Device Drivers and API Toolkit
- T4.2 – Device Drivers
- T4.3 – API Connectors

Annex E: Security Command Center R1 Software Packages describes the software deliverables to secure inter-component communication and data access. These developments correspond to task T4.4 Security & Data Access.
## Annex A: History

### Document History

<table>
<thead>
<tr>
<th>Versions</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.0.2,3,4</td>
<td>- Updating Sections 2.3, 3.2, 4.2, and 5.2 with M18 reporting.</td>
</tr>
<tr>
<td></td>
<td>- Writing of Annex C</td>
</tr>
<tr>
<td></td>
<td>- Writing of Annex D</td>
</tr>
<tr>
<td>V0.0.6</td>
<td>- Pre-internal review version</td>
</tr>
<tr>
<td>V1.0.1</td>
<td>- First Coordinator review</td>
</tr>
<tr>
<td>V1.0.2,3,4,5,6</td>
<td>- Redrafting according to first Coordinator Review</td>
</tr>
<tr>
<td>V1.0.6-8</td>
<td>- Redrafting according to second Coordinator Review</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contributions</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPV: Francisco Fraile</td>
<td>coordination and document writing</td>
</tr>
<tr>
<td></td>
<td>Victor Anaya, contributions to Devices Drivers definitions</td>
</tr>
<tr>
<td>ALM:</td>
<td>Ludo Stellingwerff, IO Toolkit workflows and interactions</td>
</tr>
<tr>
<td></td>
<td>Andries Stam, Internal revision</td>
</tr>
<tr>
<td>LYON2:</td>
<td>Néjib Moalla, contributions to API Connectors definitions</td>
</tr>
<tr>
<td></td>
<td>Abdelhadi Belfadel, contributions to API Connectors definitions</td>
</tr>
<tr>
<td>ICE:</td>
<td>Stuart Campbell, Coordinator revision</td>
</tr>
<tr>
<td></td>
<td>Linda Grimshaw, Coordinator revision</td>
</tr>
<tr>
<td>IKERLAN:</td>
<td>José Luis Fuentes, contributions to Security definitions</td>
</tr>
<tr>
<td>UNINOVA:</td>
<td>Joao Sereipa, internal revision</td>
</tr>
<tr>
<td>KBZ:</td>
<td>Raquel Melo de Almeida, internal revision</td>
</tr>
</tbody>
</table>
Annex B: References

None
Annex C: IO Toolkit Specifications

This annex provides detailed documentation of the IO Toolkit specifications (Task T4.1).

C.1. IO Toolkit Features Specifications

This section describes the specifications for the features implemented in the software resources of the IO Toolkit. The specifications also establish the priority of each feature according to the following definitions:

- **MUST**: Are mandatory to fulfil the objectives of the project
- **SHOULD**: Are not mandatory but highly recommended
- **MAY**: Are “nice to have” and would be likely post project candidates

The descriptions of the features are organised in different sections according to the main functionalities of IO Components presented in Section 1.4.1.

C.1.1. Top Interfaces

This section describes IO Toolkit features to interact with other vf-OS Platform components:

- **REST API Server Composition**: The IO Toolkit must provide tools to build the REST APIs implemented by Device Drivers and API Connectors to interact with other vf-OS components. REST API Server composition tools must enable Software Developers to:
  - Create and modify the data models used by the IO component
  - Generate and modify the metadata of the IO component
  - Create and configure the API endpoints for sensor data access, historic sensor data access, subscribing to APIs and sensor events, or requesting the execution of actuation commands.

- **Messaging client**: The IO Toolkit must provide tools to allow the Device Drivers and API Connectors to interact with the Messaging component, enabling Software Developers to:
  - Send data messages to other vf-OS components
  - Receive data messages from other vf-OS components

- **Pub/Sub client**: The IO Toolkit must provide tools to allow the Device Drivers and API Connectors to interact with the Pub/Sub component, enabling Software Developers to:
  - Provide triggers and events to which other vf-OS components can subscribe
  - Subscribe to existing triggers and events in other vf-OS components and provide local event handlers for them.

- **Metadata template composition**: The IO Toolkit must provide tools to generate a metadata template file containing the skeleton of the metadata of the IO Component according to the implemented features. It must automatically populate several fields to make it easier for the developer to generate the IO Component metadata.
• **Registration client:** The IO Toolkit must provide endpoints to the REST services used to register the Device Drivers and API Connectors to the Execution Environment, within the Platform and within the Marketplace.

• **Logging client:** Provide endpoints to send information regarding the status and the usage of the component to the System Dashboard and the Marketplace.

### C.1.2. Logic

This section describes common functionalities within the internal behaviour of Drivers and APIs.

• **Internal Storage:** The IO Toolkit must provide tools to connect and query the internal databases used to store configuration and operational data. The IO Toolkit must consider both relational and non-relational databases.

• **Key-value store:** Besides databases and internal storage, Device Drivers, and API Connectors can benefit from key-value stores to manage associative key-value pairs. The main advantage of a key value store is more optimised memory use, which can yield significant performance gains, especially in embedded devices.

• **Edge computing:** Edge computing refers to the capability to process sensor data as close to the data source as possible, instead of processing raw sensor data in the cloud. Embedded computing enables edge computing to be performed in the device, but it can also be performed in middleware like vf-OS Drivers. The main advantages of Edge computing are less latency in critical I/O operations, less bandwidth requirements for cloud infrastructure and higher safety. The IO Toolkit must provide development resources to implement edge computing in the internal logic of IO Components.

• **API Access Control:** API access control refers to authentication and security, which can be used alone or in combination to issue credentials and control access. Access control is the essential first step to make sure only API calls with valid, successfully authenticated credentials are able to access any API. Restrict access to specific endpoints, methods, and services as well as apply access policy users of software components can be done through:
  - Standard API keys
  - Application ID and key pair
  - OAuth v1.0 and 2.0

• **API Lifecycle management:** The IO Toolkit must provide API lifecycle management tools to support the management of the different phases in the lifecycle of APIs: prototype, publish, deprecate, and block APIs.

• **Simulation:** Simulation allows the testing of interactions with the IO Component without it being actually connected to a manufacturing asset. Simulation functionalities are important for designing and testing new vApps without the need to arrange physical devices or software licenses. The IO Toolkit must provide development resources to simulate physical devices and APIs and also, to facilitate the development of a simulation mode for the IO Component, so that Software Developers can test vApps without having to connect IO Components to manufacturing assets.
C.1.3. Bottom Interfaces

This section describes the development resources available to implement Bottom Interfaces of Device Drivers and API Connectors.

C.1.3.1. IO Menu Card

The IO Toolkit must provide development resources to develop IO Components based on the following technologies, which represent 20 of the most used technologies to exchange data with manufacturing assets\(^5\).

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description (Wiki)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development resources for Device Drivers</strong></td>
<td></td>
</tr>
<tr>
<td>OPC UA</td>
<td>OPC UA is a machine to machine communication protocol for industrial automation developed by the OPC Foundation. The OPC UA standard is becoming a de-facto standard for interoperability with automation devices. It is also the technology selected to interact with the Tabber Stringer machine in pilot 1, since it is supported by the PLC embedded in the machine. Therefore, this development resource is the basis for the development of the TS Driver used in this project pilot.</td>
</tr>
<tr>
<td>MQTT</td>
<td>MQ Telemetry Transport is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based &quot;lightweight&quot; messaging protocol for use on top of the TCP/IP protocol. MQTT is supported by the most prominent IOT platforms, like IBM Watson IoT, AWS IoT, or Microsoft Azure and it is therefore an important development resource for integrate devices compatible with these platforms.</td>
</tr>
<tr>
<td>Profinet</td>
<td>Profinet is proprietary Ethernet M2M protocol implemented in Siemens automation equipment. Siemens is one of the major providers of automation equipment in Europe and natively supports the Profinet protocol will enable interoperability with Siemens devices that do not support other open communication protocols.</td>
</tr>
<tr>
<td>ADS</td>
<td>ADS is the proprietary Ethernet M2M protocol used by Beckhoff automation equipment. Similar to Profinet, ADS development resources will support the development of native Device Drivers to enable interoperability with Beckhoff automation equipment with limited connectivity.</td>
</tr>
<tr>
<td>Modbus</td>
<td>Modbus is an open Ethernet protocol which provides client/server communication between devices connected on different types of buses or networks. Modbus is a serial communication protocol extensively used in SCADA systems and therefore development resources for this technology will support the development of Device Drivers for automation equipment from different manufacturers.</td>
</tr>
<tr>
<td>Web Thing API</td>
<td>Web Thing API is an open source specification that has been developed by the W3C Web of Things Interest Group to bridge the communication gap between connected devices. Since it is based on web compatible technologies like web sockets, the Web Thing API will provide resources to implement interfaces to Device Drivers based entirely in these technologies.</td>
</tr>
<tr>
<td>GPIO</td>
<td>The General Purpose Input Output is a generic hardware interface integrated in single board computers (eg Raspberry Pi) which is widely used in fast prototyping. GPIO development resources will enable the rapid development of prototypes of devices compatible with the vf-OS ecosystem.</td>
</tr>
<tr>
<td>MICA</td>
<td>The Modular Industry Computing Architecture is a computing platform half-way between PLCs and compact, general purpose single board computers like the Raspberry Pi.</td>
</tr>
</tbody>
</table>

\(^7\) [http://158.42.105.151 mediawiki/index.php/Glossary#tab=Supporting_terms](http://158.42.105.151 mediawiki/index.php/Glossary#tab=Supporting_terms)
MICA provides an optimal platform for the development of ad-hoc automation equipment designed for vf-OS

**CoAP**
Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained networks and nodes for machine-to-machine applications such as smart energy and building automation. CoAP provides a method/response interaction model between application endpoints, supports built-in resource discovery, and includes key web concepts such as URIs and content-type. CoAP development resources will enable the development of Device Drivers that integrate data from CoAP sensor networks.

**MT Connect**
MT Connect is another protocol which, like OPC UA, has been designed to promote interoperability to exchange data between industrial network components. MT Connect support will extend the interoperability features of the IO Toolkit.

### Development resources for API Connectors

**MS Excel**
Microsoft Excel workbooks and spreadsheets are widely used in to build data analysis and reporting applications. Many SMEs used these applications to manage structured data at different organisational levels. Project pilot owners agreed to use Microsoft Excel spreadsheets to manage some of the data that needs to be integrated in the requested applications. Microsoft proposes an open API specification for Excel and there are some open source implementations that can be used as a basis to develop the MS Excel Connector.

**STEP**
Standard for the Exchange of Product Data (STEP) - ISO 10303 standard - is a file format used in CAD to work with 3D models. The STEP file format is widely used in the design of technical parts and is supported by the most popular CAD Software tools. This is the file format used in pilot 3 to exchange the models of the products for collaborative manufacturing. The STEP format is interoperable with several XML based formats (3DXML, IGES, etc.) with some API specifications, so it is expected that this API Connector can be reused or extended to support other standards.

**ODBC**
Open Database Connectivity (ODBC) is a standard API to access database management systems. The purpose is to provide a unified API to access several database management systems and operating systems. The ODBC Driver can be therefore used to exchange data from different database servers and also reused and extended to integrate on premise software applications. In this perspective, Microsoft offers a generic ODBC API reference specification which can be used to develop this API Connector.

**LxP ERP**
LxP is an ERP System developed by the SME CISA Informatique and is currently used in APR. This represents a case of integrating an ERP designed for medium to small businesses. LxP is configured to be connected to a main SQL database and an alternative database for all quotes traceability.

**OData**
The Open Data protocol is a REST protocol developed to create and consume interoperable RESTful APIs. OData has strong industrial backing from companies like IBM or Microsoft. OData development resources allow developers to build API Connectors with several market leading business software applications and systems (eg IBM WebSphere, Microsoft Dynamics CRM, Acumatica ERP, SAP ERP).

**SAP ERP**
SAP is a leading ERP software for large companies in the market worldwide. The SAP ERP Driver will use the OData-based SAP NetWeaver Gateway, reusing the OData development resources and facilitating the development of APIs for other SAP products.

**NAV ERP**
Microsoft Dynamics NAV is an ERP system developed by Microsoft. The NAV ERP API Connector can use the OData-based web API specification and implementation provided by Microsoft Dynamics.

**LIFERAY**
Liferay is a platform to create and manage horizontal portals for different user profiles such as clients, partners or portals. Liferay is the leading Content Management System to structure and share enterprise data. Liferay offers an open API specification and implementation to access the portal services, locally or remotely, using technologies such as SOAP or JSON that can be used to implement the LIFERAY API Connector.
C.1.3.2. Device Integration

This section describes IO Toolkit resources specifically for the development of Device Drivers:

- **Sensor Data Read skeleton**: The IO Toolkit must provide the skeleton (high-level sample code) of a sensor data reading Device Driver. The objective is to use this skeleton as a template for the development of new Device Drivers. The skeleton must use all the Top Interface and Logic functionalities so that a new driver with sensor reading capabilities can be developed just by filling the empty functions in the skeleton with the specific physical device function calls. The sensor data read skeleton will implement a simulation mode that will simulate a connected sensor generating random data, so that it can be used for testing purposes. Moreover, the sensor data skeleton must implement automated unit tests to help validating the development of the Device Driver.

- **Actuation Command skeleton**: Similarly, the IO Toolkit must provide the skeleton of a Device Driver to send commands to actuators. As with the sensor data read skeleton, the actuation command skeleton must integrate all the Top Interface and Logic functionalities to enable the fast development of Device Drivers with actuation capabilities.

- **Low latency stream data sample**: The IO Toolkit must provide a sample of a low latency stream protocol that can be used by time critical vApps to access sensors and actuators directly (bypassing the Pub/Sub or messaging components).

C.1.4. Containerisation

The IO Toolkit provides standard template images, based on the templates provided by the Execution Services, modified to fit the specific requirements of the Drivers and APIs. D11.5 presents the technical infrastructure, common to all the vf-OS platform, where Docker is a core technology to build microservices architectures as the one proposed in vf-OS. Docker is an open platform for developing, shipping, and running applications. Docker enables to separate applications from the infrastructure as well as from each other. In contrast to standard virtualisation where applications run within virtual machines (with Guest OS) hosted in the hypervisor, Docker containers run directly within the host machine’s kernel. This is achieved by the Docker Engine enabling the separation of processes running in the host machine operating system.

Docker isolation of individual services and versatility in selection of programming language for tools development and deployment allows the development of drivers in different programming languages while following Drivers and API connectors specifications. Other benefits that Docker provides the IO Toolkit Execution Services are:

- **Standardised API for connecting various services**: Docker provides Engine API to execute various requests on creating, deploying, managing containers. Therefore, it
is suitable for connecting with IO tools through API connectors. Every container exposes a port that can be connected with other external resources or drivers

- **Portability of application**: The portability of containers allows them to run on various machines regardless of any host operating system
- **Security features**: The need to secure one OS and the Docker engine on top of it. Another major benefit of containerisation from a security point of view is the possibility to validate and sign containers, through the Docker Trust feature

Device drivers and API connectors can be deployed by vf-OS marketplace as Docker containers deployed in Docker environment managed by a vf-OS Platform. Through the IO Toolkit the process of creating such Docker containers can be simplified for the developers of the APIs and Device drivers, by providing standard templates and a streamlined process for packaging these containers.

### C.1.5. Development tools

Although the IO Toolkit does not enforce specific programming languages and environments, there are shared services and solutions between the various Components. Among other potential services, the IO Toolkit will at least provide:

- Containerisation tools
- Template containers for Device drivers and API connectors
- Documentation standard
- IO toolkit documentation
- Standard versioning scheme for the Toolkit and the produced components
- A testing framework for the interfacing with the rest of vf-OS, ie. for the Top Interfaces

Most of these development tools will interact and be available, through the vf-OS OAK.

### C.1.6. IO Toolkit Security

Security is an inherent feature in vf-OS which allows a controlled flow of information between all elements, where every action has been identified, approved, and logged with one major goal: to guarantee a productive and secure environment for developing advanced applications in industrial environments.

The IO Toolkit Security incorporates a mapped set of security requirements and a documented unit testing plan focused on security for the development of new vf-OS IO Components. The IO Toolkit will provide developers with the documentation of the security of data in transit policies adopted in vf-OS, defined in D2.4 Holistic Security and Privacy Concept. The following set of recommendations for developers are obtained to facilitate the configuration of the data in transit policies adopted in vf-OS:

- **Algorithms**:
  - **Symmetric Encryption**: Advanced Encryption Standard 256
  - **Asymmetric Encryption**: Elliptic Curve Digital Signature Algorithm (ECDSA) 512 as first candidate, if not available RSA 4096
  - **Key Exchange**: ECDSA (+Perfect Forward Secrecy)
  - **Encryption Chaining Mode**: Galois/Counter Mode (GCM) with zero starting for all initialization vectors
• **Use of Transport Layer Security (TLS) 1.2**

• **Key period recommendations:**
  - All Usage Originator Period (OUP): 1 Year
  - All Recipient Usage Period (RUP): 1 Year

A developer can always access the information regarding the security requirements the application must satisfy. In addition to this, the IO Toolkit skeletons and samples will implement these security policies for data in transit to facilitate the development of new IO Components.

Regarding user authentication, the IO Toolkit Security provides development resources for developers in order to ensure that the development of IO Components is compliant with the vf-OS Holistic Security Concept. Thus, all IO Components will be required to include the corresponding libraries to interoperate by using OpenID®. In this sense, the OpenID libraries, products and tools are an essential source for leveraging all resources of this technology.

In addition to this, the development of all IO Components must follow at least the following practices:

- Secure Coding Practices Quick Reference Guide (OWASP®)
- Secure Coding Practices Checklist (OWASP)
- All functions must be testable

Based on these resources, the IO Toolkit Security will provide endpoints for the vf-OS Security Command Centre REST APIs (Identity Service, Policy Administration, and Continuous Security Monitoring).

Developers should use the EJBCA® Public Key Infrastructure (PKI) Certificate Authority software in applications when a complete PKI infrastructure is needed.

Finally, installed IO Components must use the security functions of the IO Toolkit Execution Service and the native operating system of the container to improve security:

- **Access Control System:** The IO Component host operating system can provide an additional control to guarantee that any asset follows the general security policy and all prohibited actions are blocked and logged. This additional control is aimed to control those cases where the IO Component has been modified and the behaviour is inappropriate either accidentally or intentionally

- **Firewall:** The vf-OS Secure Network Architecture, vf-OS Security Command Centre, and containerisation guarantee that only authorised connections from/to allowed assets reach the IO Component through Top Interfaces. Policy Enforcement Points (PEPs) are Next-Generation Firewalls that implement the vf-OS Secure Network Architecture Defence-in-depth strategy. Additionally, IO Components native host operating systems must implement a firewall system to prevent unauthorised network connections

---

8 [http://openid.net/](http://openid.net/)
9 [https://www.owasp.org](https://www.owasp.org)
10 [https://www.ejbca.org/](https://www.ejbca.org/)
• **IO Component management, Safety and availability**: The vf-OS Platform instances must guarantee the availability of the platform services in cases where IO Component has been incorrectly designed and/or developed or where the component may compromise the overall performance of the vf-OS Platform (e.g., using excessive computational resources). vf-OS provides mechanisms for managing automated notifications about the status of IO Components (System Dashboard) and services to manage both IO Components and hosts (vf-OS Platform), in order to guarantee safety and availability of applications.
Annex D: IO Toolkit R1 Software Packages

This annex provides the detailed documentation of the R1 software packages developed for the IO Toolkit, in the context of T4.1, T4.2, and T4.3.

D.1. IO Toolkit Generator Software Package

The IO Toolkit Generator is a design tool used to automate IO Component development. The generator is used to create a complete working copy of the IO Component binaries, so that developers need only implement the specific code associated with the component they want to build. Since all IO components will have many common functionalities, the generator automatically creates the code of all common parts of a project, so that developers can focus on the functionality they want to generate in the new component. This technique to reuse code and automate code generation is known as scaffolding\(^{11}\).

As mentioned in Section 1, there is an analogy between the use of the OAK Toolkit for the development of vApps and the use of the IO Toolkit for the development of IO Components. Therefore, there are some dependencies and interactions between the IO Toolkit and the vf-OS Studio. Figure 7 shows the relationship of the IO Toolkit Generator with the OAK SDKs, the IO Toolkit SDK, the vf-OS Studio components, and the vf-OS Marketplace components.

---

\(^{11}\) [https://en.wikipedia.org/wiki/Scaffold_(programming)]
• The Core SDK implements methods for inter-asset communication, such as messaging & the Publish/Subscribe framework, and methods for providing standard RESTful APIs to interact with the execution environment, the System Dashboard and other shared resources. Application logging is a core example of such a resource.
• The IO Toolkit SDK implements methods to exchange data with manufacturing assets, using specific protocols and technologies
• The vApp SDK implements methods for easier application development, such as the interaction with the end-users through a web-based UI, handling of the output of the process designer and basic security functions like logging and registration

Figure 8 is a mapping of the functional specifications of the IO Toolkit in Annex C: into the generator, the different SDKs (Core, IO, vApp) and the Studio components (Packager, Uploader).

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Implemented in</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST API Server Composition</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>Internal storage</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>Key-value store</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>Edge computing</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>API Access Control</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>API Lifecycle management</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>Simulation</td>
<td>IO Toolkit</td>
<td>R1</td>
</tr>
<tr>
<td>Messaging client</td>
<td>Core SDK</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Logging service client</td>
<td>Core SDK</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Metadata composition</td>
<td>Packager</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Registration service client</td>
<td>Uploader</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

Figure 8 Roadmap overview

The IO Toolkit Generator provides a framework to generate a working copy of the component, to integrate the core SDK, and to integrate the specific libraries of the IO SDK into this working copy, based on the technology that the developer would like to use. The end result is a working NodeJS web application that implements the REST API of the IO Component. The application logic of the resulting web application uses the core SDK to connect to platform components (Top Interfaces) and IO Toolkit SDK methods to connect to the manufacturing asset it is designed for. The developer can then modify the generated source code to implement the specific functionalities to be added.

Figure 9 shows the structure of the IO Toolkit Generator R1. The generator is based in Yeoman\(^\text{12}\), which is a popular scaffolding framework, with lots of generators available to scaffold NodeJS web applications. Yeoman has a feature called Composability, which allows to reuse available generators in the creation of a new generator. The IO Toolkit uses this feature to build upon the Loopback:swagger\(^\text{13}\) generator. This allows a developer to create a working web application from a Swagger REST API definition. The generator

\(^{12}\) [http://yeoman.io/](http://yeoman.io/)
\(^{13}\) [https://loopback.io/](https://loopback.io/)
integrates a series of Templates, which provide further instructions and binaries for the generator to customise the generated web application for specific use cases. The IO Toolkit Generator R1 has a command tool UI that prompts a set of questions to the developer to collect the required user input (eg “what kind of component you would like to build?”) and creates the web application binaries following the instructions provided by each template.

Figure 9 IO Toolkit Generator structure

The next subsections comprise the requirements, installation instructions and usage guidelines for the IO Toolkit Generator. Section D.2 describes how Templates are created from the available IO Toolkit SDK modules, and Section D.3 and D.4 describe the available SDKs to develop Device Drivers and API Connectors respectively.

D.1.1. IO Toolkit Generator Requirements

The IO Toolkit Generator requires the following frameworks / packages to be installed:

- **Nodejs and NPM**: the interpreter of all the frameworks and the IO Toolkit itself, along with the package manager
- **Yeoman**: the Nodejs scaffolding framework used in the IO Toolkit Generator
- **Loopback**: the Nodejs API framework used by the IO Toolkit Generator
- **Swagger-loopback**: the Nodejs loopback extension used to generate the APIs interfaces of the vf-OS components according to a swagger definition of the interfaces defined in Deliverable D2.3

D.1.2. IO Toolkit Generator Installation

At the time of writing (M18), the IO Toolkit Generator has not been published in the NodeJS registry and package manager Node Package Manager (NPM). To test it, developers need to download the source code, install it and link the package so that it can be tested in the local environment. Figure 10 shows the NPM commands used to do this.

```
> cd generator-vfos-io
> sudo npm install -g
> sudo npm link
```
D.1.3. IO Toolkit Generator Execution

When the IO Toolkit Generator is installed, the Software Developer must navigate to a folder where the code of the IO Component will be generated and run the yo command with the vfos-io generator, as shown in Figure 11.

```
> sudo mkdir driver-test
> cd driver-test
> sudo yo vfos-io --loopback
```

The IO Toolkit generator has no argument and only one option, the loopback option. When present, this option configures the IO Toolkit Generator to create a working web application using the Loopback framework. Otherwise, it only generates the specific IO Component code. The IO Toolkit Generator shows the user prompts that ask the developer for the required input. The current version contains three main user prompts:

- **Name**: input prompt to specify the name of the IO Component
- **Type**: List prompt to define the type of IO Components in vf-IO: API Connectors or Device Drivers
- **Template**: List prompt to specify which template of the available templates for the selected type the developer would like to use

Figure 12 shows a screenshot of the shell window with the command interface of the IO Toolkit generator and the three main prompts.

The result is a working copy of the IO Component, which implements the REST interfaces provided to the other platform components, and the inner logic and the bottom interfaces, using the IO Toolkit SDKs to connect to legacy devices and software. These libraries are instantiated, so that developers can easily extend the provided functionalities. Section D.2 describes the inner structure of the templates.
As an example, Figure 13 shows the resulting output of the generator when the empty driver-skeleton option is selected.

D.2. IO Toolkit Templates Software Package

This section describes the inner structure of templates and provides instructions to author new templates to add them to the IO Toolkit Generator.

D.2.1. IO Toolkit Templates Requirements

As explained in Section D.1, the IO Toolkit Generator shows the user the list of available templates that he would like to use to scaffold the working copy of the IO component. Thus, templates control the generation of the code to connect with manufacturing assets using a certain technology.

Templates must contain a “files” folder, which stores the template files that are used to generate the code, and a `data.json` file, containing the data that describes the template and provides instructions to the generator.

The template files will be processed by the Yeoman file utilities. The basic usage is to copy the files from the files folder into a destination folder in the generated code file structure. The instructions to control the Yeoman writing task are provided in the `data.json` file, which has the following properties:

- **Name**: Name of the template as shown in the generator list prompt
- **Description**: Text description for developers
- **Type**: IO Component type (“driver” or “connector”)
- **Package**: Information to write to the package file (Optional)
- **Swagger**: Location in files folder of the Swagger definition file of the REST API of the component
- **Files**: Array of IO objects (Optional)

14 http://yeoman.io/authoring/file-system.html
• **IO**: Object containing a source field with the location of the file to copy and a destination field with the location of the output file in the generated code where it will be copied

• **Prompts**: Array of questions to ask the user for additional information. Questions follow the Inquirer\(^{15}\) question object notation and are used to add new prompts to the generator to ask the user for further information to generate the code (Optional)

• **Dependencies**: Array of NPM modules to add to the Yeoman installation task (Optional)

• **TransformStream**: Gulp transform stream to use the user answers to the questions to transform the output files (Optional)

Figure 14 shows the source code of a simple template.

```javascript
var template = module.exports;

// Name that appears in the generator list
template.name = "Empty-driver (no configured M2M protocol or datasource)";

// Long description
template.description = 'An empty Device Driver, without any configured ' + 'protocols or datasources';

// IO Component type: 'driver' or 'connector';
template.type = 'driver';

// Version
template.package = {
  'version': '1.0.0'
};

// Swagger file location in 'files' folder.
// The Swagger file defines the API of the REST application using the Open API
// specification
template.swagger = '/swagger/Driver.yaml';

// Files to process with the file utilities. The source property specifies the location of the file to copy and the destination property specifies the location in the generated code
template.files = [{
  source: 'drivercomponent/driverTemplate.js',
  destination: 'common/drivercomponent/driverTemplate.js'
}];

// Inquirer prompts using the Inquirer.js question notation
template.prompts = {
};

template.dependencies = {
};
```

\(^{15}\) [https://github.com/SBoudrias/Inquirer.js](https://github.com/SBoudrias/Inquirer.js)
D.2.2. IO Toolkit Templates Installation

Available templates are located in the templates folder of the IO Toolkit Generator. Developers need only to copy the template to this folder to add them to the IO Toolkit Generator.

D.3. IO Toolkit Device Drivers SDKs Software Package

This section contains information to use the Device Drivers SDK of the IO Toolkit Generator.

D.3.1. IO Toolkit Device Drivers SDKs Requirements

The IO Toolkit Device Driver SDK has the same requirements and dependencies than the IO Toolkit (see section D1.1 and D1.2).

D.3.2. IO Toolkit Device Drivers SDKs Usage

The IO Toolkit Device Driver SDKs consists of a set of tested libraries that Software Developers can use to develop new Device Drivers. When the developer runs the IO Toolkit generator and selects a template, the selected template configures the IO Toolkit Generator to copy the SDK library to the generated code, to install the required dependencies, and to add the corresponding callbacks to the event loop of the REST application, so that the right SDK function is called when the REST application receives a new request.

The generated code will implement the common functionalities, such as Logging or Pub/Sub and Messaging, as well as the specific functionalities to connect to the device using the selected technology, so that Device Drivers developers need only to implement new functionality on top of this, such as edge data processing.

There are two Device Drivers SDKs in the current version, the Empty-Driver SDK, which is a generic library that can be extended to connect to a device using a technology which is not supported by any SDK, and the OPC UA SDK, which is a library designed to interact with operational technology using the OPC UA protocol.

D.3.3. Empty-Driver SDK

As explained above, the Empty-Driver SDK is instantiated when a Device Driver developer wants to connect to a device from scratch. The Empty-Driver provides the skeleton of a Device Driver SDK with the callbacks that are added to the event loop of the REST application:

- **readSensorData**: Method that will receive a sensorConf object with the configuration information of the sensor that should be read. The developer should code the protocol specific code, that after connecting and reading the sensor data, will call the callback function passing a string value with the data read from the sensor as a parameter
- **sendCommand**: Method that will receive the command object, and will implement the protocol-specific code that will send that command to a device
- **pushSensorData**: This method is passed as a parameter by the SDK as a callback to push sensor data asynchronously. The method will receive a string parameter with the sensor data.
- **logger**: This method is passed as a parameter by the SDK and represents a function callback to log messages to the platform.

Every other SDK implementation will have the same basic structure and the same methods as the Empty-Driver SDK.

```javascript
// Specific driver
// it covers the mappings to specific protocols
// the methods will be used by devices
// require libraries that you are using

module.exports = function(loggerCB){

  // Callback method to send error messages back to the platform to be logged
  this.logger = loggerCB;

  // proprietaryParameters: list of proprietary parameters that will be part of the
  // configuration necessary
  // to handle to work properly
  let proprietaryParameters = [];

  // mandatory methods of the driver. Any developer must code them
  return {
    /**
     * method called by the Devide Driver SDK when reading a sensor according to the
     * protocol
     * @param {object} sensorConf configuration of a sensor containing the
     * proprietary parameters to connect to a server and read data
     * @callback {Function} callback Callback function
     * @param {string} result string with the value of the sensor data
     */
    readSensorData: function(sensorConf, callback){
      let result=0;
      // TODO: the developer will write here the driver and will call the callback
      passing in result a string with
      // the retrieved value
      callback(null, result);
    },
    /**
     * method called by the Devide Driver SDK when subscribing to changes of a sensor
     * @param {object} sensorConf configuration of a sensor containing the
     * proprietary parameters to subscribe to a server monitoring sensor changes
     * @callback {Function} callback Callback function
     * @param {string} result string with the value of the sensor data
     */
    subscribe: function(sensorConf, callback){
      let result=0;
      // TODO: the developer will write here subscription code to the server of
      monitoring the device so that the server
      // will call the callback function with the value pushes whenever a trigger
      occurs
      callback(null, result);
    },
    /**
     * method called by the Devide Driver SDK when sending a command action to a
     * sensor
     * @param {object} command command for the sensor
     * @callback {Function} callback Callback function
     * @param {string} acknowledge string with a message of success or failure
     */
    sendCommand: function(command, callback){
      // TODO: the developer will turn the command object into a protocol specific
    }
  };
};
```
D.3.4. OPC UA SDK

The OPC UA SDK is an implementation of an OPC UA Device Driver library based on NodeOPCUA\(^\text{16}\). The library uses this open source OPC UA stack to implement an OPC UA client with the SDK functionalities: readSensorData and sendCommand functions and using pushSensorData and logger call-back functions. The OPC UA SDK retrieves the specific configuration parameters of devices and sensors and uses this information to set up the connection to an OPC UA server (eg embedded in a device). The sensorConfiguration object parameter contains the following information:

- **sid**: Id of the sensor
- **name**: String representing the name of the sensor
- **properties**: An array of key values with descriptive properties that could be attached to the sensor, such as a unit of measure, the location of the sensor, etc
- **driver**: An object that contains the communication protocol implemented by the device, the version of the protocol, the name, and the description
- **triggers**: Configuration of sampling intervals and threshold values to configuration notifications
- **historicalData**: Boolean used to confirm the need to store historical data measures
- **computingExpression**: String with the computing expression that will be processed by the controller on the Device Driver SDK, substituting \%v by the value of the measure retrieved and computing the value after
- **actuator**: Boolean used by the driver controller to know if the sensor can receive a command
- **proprietaryParameters**: An array of key values with proprietary parameters necessary to configure specific sensor. In the case of OPC UA, the proprietary parameters are:
  - **resourcePath**: Where the opc_ua server is gathering data from different sensors (eg “opc.tcp://ip:port/server_route”)
  - **sensorPropietaryId**: URI of the sensor according to the OPC UA protocol (eg “ns=1;s=free_memory”)

D.4. IO Toolkit API Connectors SDK Software Package

This section contains instructions to use the IO Toolkit API Connectors SDK.

\(^{16}\) [http://node-opcua.github.io](http://node-opcua.github.io)
D.4.1. IO Toolkit API Connectors Requirements

The IO Toolkit API Connector SDK has the same requirements as the IO Toolkit Generator (see section D1.1 and D1.2).

D.4.2. IO Toolkit API Connectors SDK Usage

The IO Toolkit API Connectors SDK allows developers to generate a working web application that connects to a database using the ODBC protocol and exposes the REST API defined with the Swagger Open API specification. Thus, this SDK provides a simple means to scaffold an Asset to access different database management systems.

The IO Toolkit API Connectors uses the IO Toolkit template framework presented in Section D.2. The IO Toolkit API Connectors SDK defines a series of user prompts to get from the user the Swagger definition of the REST API and the configuration of the ODBC connection to the database.

An example of questions asked to the developer are presented in the UML sequence diagram in Figure 15.

Figure 16 API template UML diagram
Annex E: Security Command Center R1 Software Packages

This annex provides the detailed documentation of the R1 software packages developed for the Security Command Center Packages task. The vf-OS Security Architecture is based on several key security components. Next sections provide instruction to install the Identity Management System and the Authorisation Policy Decision Point Generic Enabler.

E.1. Identity Management Service Software Package

This section comprises the instructions on how to download, deploy and execute the Identity Management Service. For convenience, the source code provides a shell script to install the Security Command Center that includes all the shell script commands described below.

E.1.1. Identity Management Service Requirements

The Identity Management System runs in Ubuntu Linux and is based in the Keyrock17 Fiware Identity Manager. First, it is necessary to install the Ubuntu dependencies, as in the script in Figure 17.

```
$ sudo apt-get install -y libopenscap8
sudo apt-get update && \
$ sudo apt-get install -y wget python git vim expect && \
wget https://bootstrap.pypa.io/get-pip.py && \
sudo python get-pip.py
sudo apt install gcc
```

Figure 17 Installing Identity Management Service Ubuntu dependencies

The second step is to install the Ubuntu SCAP Ubuntu profiles, as in Figure 18.

```
git clone https://github.com/GovReady/ubuntu-scapy.git
git clone https://github.com/OpenSCAP/openscap-daemon.git
  cd openscap-daemon
  sudo python setup.py install
  cd ..
git clone https://github.com/OpenSCAP/scap-security-guide.git
git clone https://github.com/mpreisler/oscap-ssh.git
```

Figure 18 Installing Identity Management Service Project Ubuntu dependencies

E.1.2. Authentication Service Installation

Then, the Keyrock code can be installed from the corresponding Git repositories, as depicted in Figure 19.

17 https://catalogue.fiware.org/enablers/identity-management-keyrock
The next step is to configure the Keyrock Identity Manager. This version uses the configuration samples provided in the source code.

```
$ sudo apt-get install -y python-dev python-virtualenv libssl-dev libffi-dev libjpeg8-dev libxml2-dev libxslt1-dev libasl2-dev libssl-dev libldap2-dev libffi-dev libsqlite3-dev libmysqlclient-dev python-mysqldb
$ git clone https://github.com/ging/keystone
$ cd keystone && git checkout tags/keyrock-6.0.0
$ cd HOME
$ git clone https://github.com/ging/horizon
$ cd horizon && git checkout tags/keyrock-6.0.0
$ cd HOME
$ cp keystone/etc/keystone.conf.sample keystone/etc/keystone.conf
$ cp horizon/openstack_dashboard/local/local_settings.py.example horizon/openstack_dashboard/local/local_settings.py
$ sed -i s/\$IDM_PASS/\${IDM_PASS}/g horizon/openstack_dashboard/local/local_settings.py
```

**Figure 19 Installing Keyrock**

Once that Keyrock is configured, it is necessary to install the Python dependencies, as highlighted in Figure 21.

```
$ sudo horizon/tools/with_venv.sh pip install six==1.9.0
$ sudo python keystone/tools/install_venv.py
$ sudo python horizon/tools/install_venv.py
```

**Figure 20 Configuring Keyrock**

Then, it is possible to sync the Identity Management Service database, as depicted in Figure 22.

```
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync && \\
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync --extension=endpoint_filter && \\
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync --extension=oauth2 && \\
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync --extension=roles && \\
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync --extension=user_registration && \\
$ sudo horizon/tools/with_venv.sh keystone/bin/keystone-manage db_sync --extension=two_factor_auth
```

**Figure 21 Installing Python dependencies**

**Figure 22 Sync database**
E.1.3. Authentication Service Execution

To activate and run Keystone, it is necessary to run the commands in Figure 23.

```
$ sudo ./keystone/tools/with_venv.sh ./keystone/bin/keystone-all -v &
$ sudo ./horizon/tools/with_venv.sh python ./horizon/manage.py runserver 0.0.0.0:8000
```

Figure 23 Start services

E.2. Authorisation Policy Decision Point Generic Enabler

E.2.1. Authorisation Policy Decision Point Generic Enabler Requirements

The Authorisation Policy Decision Point Generic Enabler requires a Tomcat server and the Java Development Kit installed in the System. Additionally, it is necessary to install the gdebi and curl packages to complete the installation.

```
$ sudo aptitude install openjdk-7-jdk
$ sudo aptitude install tomcat7
$ sudo aptitude install gdebi curl
```

Figure 24 Authorisation Policy Decision Point Generic Enabler download

E.2.2. Authorisation Policy Decision Point Generic Enabler Installation

The first step is to download the latest version from the repository, as illustrated in Figure 25.

```
$ wget http://repo1.maven.org/maven2/org/ow2/authzforce/authzforce-ce-server-dist/5.4.1/authzforce-ce-server-dist-5.4.1.deb
```

Figure 25 Authorisation Policy Decision Point Generic Enabler download

Then, run the `gdebi` command with the directory where the code is downloaded, as in Figure 25.

```
$ sudo gdebi authzforce-ce-server-dist-5.4.1.deb
```

Figure 26 Running gdebi command.

Before the service is executed, it is necessary to check the Tomcat settings and make sure that the service is allocated enough memory to avoid errors. The following command allocates 1G, which is the minimum recommended.

```
$ sudo sed -i "s/-Xmx128m/-Xmx1024m/" /etc/default/tomcat
```


E.2.3. Authorisation Policy Decision Point Generic Enabler Execution

Just restart the Tomcat service to execute the Authorisation Policy Decision Point Generic Enabler.

```bash
$ sudo service tomcat7 restart
```

Figure 28 Authorisation Policy Decision Point Execution