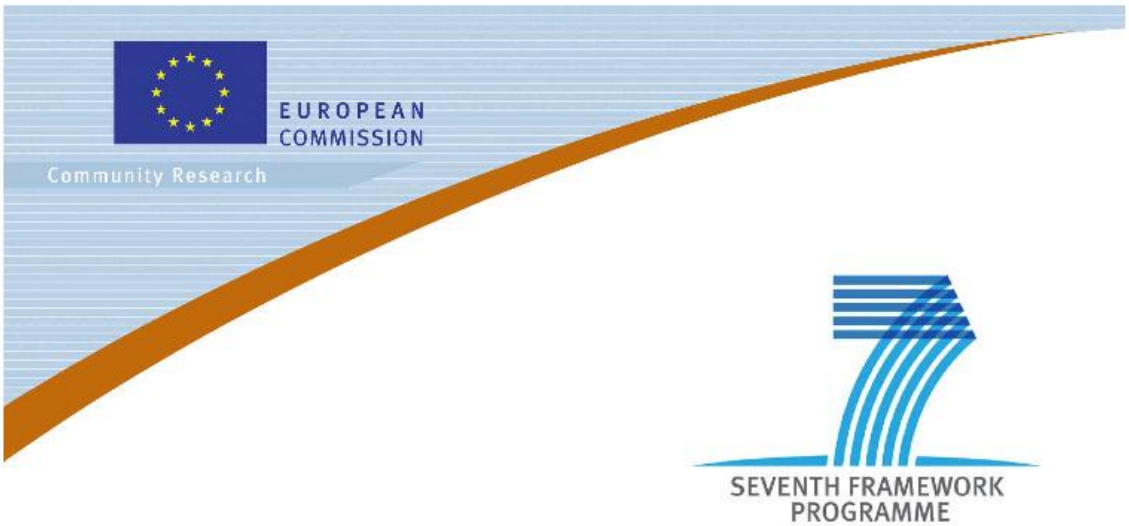


Private Public Partnership Project (PPP)
Large-scale Integrated Project (IP)

D.7.4.1b: FI-WARE User and Programmers Guide

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP7-D.7.4.1b
Project Document Date: 2012-11-08
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the usage of each Generic Enabler provided by the "Interfaces to Network and Devices" (I2ND) chapter. The document also details the required software elements and the procedure to setup an environment for the development of applications that can exploit the capabilities of the I2ND Generic Enablers.
This document consolidates new contents and also contents in previous issues of Release 1. The reason for re-delivering parts that were already issued is twofold:

- FI-WARE has made an effort to create a unified and improved format. The parts generated in the past are also provided in the new enhanced format for the sake of uniformity and readability.
- A single reference document per chapter is clearer and easier to handle that two incremental issues.

1.2 About This Document

This document comes along with the Software implementation of components, each release of the document being referred to the corresponding Software release (as per D.x.2), to provide documentation of the features offered by the components and interfaces to users/adopters. Moreover, it explains the way they can be exploited in their developments.

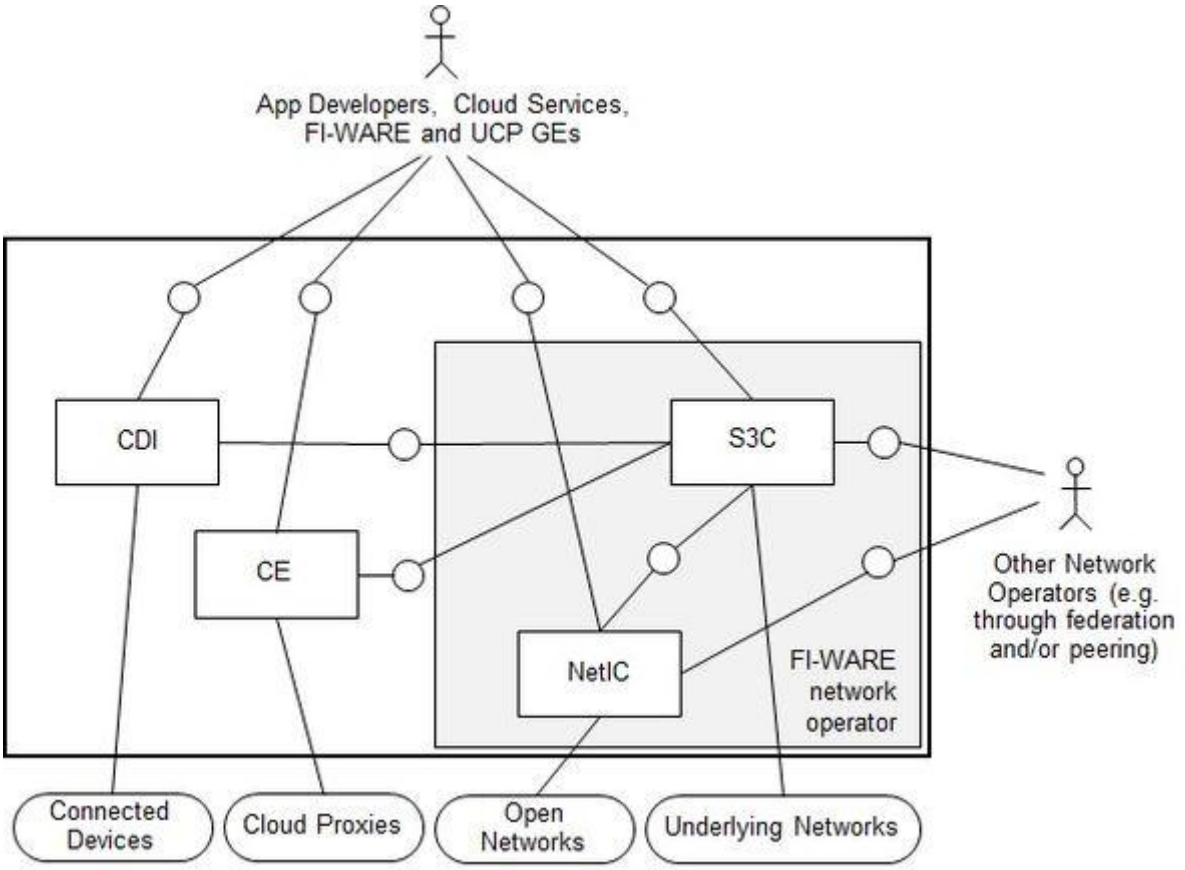
1.3 Intended Audience

The document targets users as well as programmers of FI-WARE Generic Enablers.

1.4 Chapter Context

The I2ND chapter defines an enabler space for providing Generic Enablers (GEs) to run an open and standardised network infrastructure. The infrastructure deals with highly sophisticated terminals, as well as with highly sophisticated proxies, on one side, and with the network operator infrastructure on the other side. This latter will be implemented by physical network nodes, which typically will be under direct control of an operator, or the node functionality will be virtualised – in this case the I2ND functionality can be accessed by further potential providers, like virtual operators. The I2ND chapter defines four GEs, as represented in the figure:

- CDI (Connected Device Interface) towards the Connected Devices. These devices include, but are not limited to, mobile terminals, tablets, set top boxes and media phones.
- CE (Cloud Edge) towards the Cloud Proxies. Cloud Proxies are gateways, which will connect and control a set-up of nodes towards the Internet or/and an operator network.
- NetIC (Network Information and Control) towards Open Networks. Open Networks are following the idea of flow based controlled networks, and can be used for virtualisation of networks.
- S3C (Service Capability, Connectivity and Control) towards Underlying Networks. The underlying networks are following standards such as Next Generation Networks (NGNs) or Next Generation Mobile Networks (NGMNs). In the case of the S3C specified in I2ND the baseline underlying network will be the Evolved Packet Core (EPC) by 3GPP.



Each of the GEs of I2ND have specific interfaces which can be accessed by Application Developers, Cloud Services, FI-WARE and third party Enablers. Besides typical interfaces to functionality offered by suche GEs, it is worth mentioning that:

- The GE S3C is the central point of the I2ND architecture. I2ND develops an enabling environment which can be used by network operators. Together with NetIC, both GEs build the environment of an operator, which might even be a virtual operator. S3C can be seen as the GE to run and steer the network infrastructure.
- The interfacing between S3C and CDI provides status and control information exchange of the device and remote control capabilities.
- Cloud proxies can be part of an operator infrastructure. Therefore it is necessary to have access to these network nodes through a standardised interface.

1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/ The following resources were used to generate this document:

D.7.4.1b FI-WARE User and Programmers Guide front page
Cloud Edge - User and Programmers Guide

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki.

The project is currently working on the migration of as many deliverables as possible towards the new system.
This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content.
Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

1.7 Acknowledgements

The following partners contributed to this deliverable: TRDF, TI

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor

v1	First draft of deliverable submission generated	2012-11-08	TI, TRDF
v2	Finalisation of document	2012-11-09	TI, [P. Garino]

1.10 Table of Contents

2 Cloud Edge - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

2.1 Goal of the document

This document describes the necessary environment to develop a software application targeted for a Linux container running on Ubuntu 12.04 LTS (cloud proxy). In this document, we will discuss about:

- The software and hardware environment.
- The advantage of using a container.
- A brief description of the environment provided.
- How to test a container.
- Start up application
- How to create an image.
- Download and install the image in the cloud proxy.
- How to generate an image from the rootfs used to develop an application.
- Limitations.

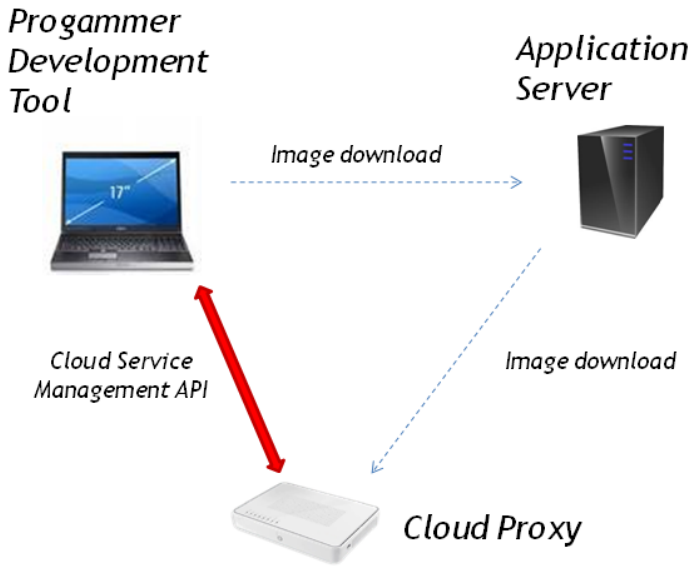
2.2 Hardware and software environment

As User and Programmer tool, we propose as hardware basis to use a PC i686 PC. We propose to install on it an Ubuntu 12.04 Long Term Support Linux distribution (note that particular care has to be done to the distribution version; otherwise the system may not work correctly). In addition we need additional software packages which mainly are LXC and associated packages (debootstrap, bridge-utils,..), sql and phpmyadmin related packages (mysql-server, phpmyadmin,…) for the data base and additional packages for convenience like ssh, vim,… The installation process is detailed in the “Installation Guide”. Please refer to it for further information.

Equipment: We use 2 PCs:

- The first PC is the cloud proxy on which the application can be designed. In the document, the hostname of this PC is hostdisk with ip address = 192.168.1.53. The ip address of the container running into our host is 10.0.3.243. The port used for port forwarding in the host machine is 30 000
- The second PC is a PC used to test an SSH connection to the container.

Principle: The user develops an application by using the software environment provided. Once the application is ready to be used, the user builds a so called “image”. This image is uploaded onto a server, accessible by an URL address. Once the image is available on the server at a dedicated address, any authorized user can use the Service Platform Management API to control the services that can be installed and run on the cloud proxy. By using those APIs or using the client tool that implement those APIs, the user can register (i.e. push) the image from the server to the cloud proxy. Those APIs (or client provided) allows also to start/stop any of those images installed on the cloud proxy so that the services running locally on the Cloud Proxy



Container system: Linux Containers (LXC) are an operating system-level virtualization method for running multiple isolated server installs (containers) on a single control host. LXC does not provide a virtual machine, but rather provides a virtual environment that has its own process and network space. It is similar to a chroot, but offers much more isolation (https://wiki.archlinux.org/index.php/Linux_Containers).
The application will be executed into a container hosted by the Cloud Proxy. The containers principle allows execution of different applications running onto the same platform with a very clear isolation between each other. This system allows a safe execution, protecting any application from a malicious application or unstable application running onto the same platform. Ubuntu provides all necessary packages for containers installation: LinuX Container (LXC).
Software environment provided: We make the assumption that Ubuntu 12.04 LTS is installed on the PC. We make the assumption that the user has already installed the software environment described into the document “installation guide”. In particular, Ruby tools and LXC packages have to be installed, as well as a rootfs is created into /var/lib/lxc/name_of_the_container/rootfs. Be sure there is a complete root file system under this directory. The directories “fiware” and cloudedge must exist too.

```
stdisk@hostdisk-07062012:~$ ls

cloudedge  Desktop  Documents  Downloads  examples.desktop  fiware
Music   Pictures  Templates  Videos

hostdisk@hostdisk-07062012:~$
```

cloudedge contains mainly 3 directories:

```
hostdisk@hostdisk-07062012:~$ cd cloudedge/
hostdisk@hostdisk-07062012:~/cloudedge$ ls
client  Gemfile  Gemfile.lock  README  spm  ves
hostdisk@hostdisk-07062012:~/cloudedge$
```

- Client
- Service plateforme management (spm)
- Virtual Environment Service (ves)

fiware contains several directories. The most important is ~/fiware/script

```
hostdisk@hostdisk-07062012:~/fiware/scripts$ cd
hostdisk@hostdisk-07062012:~$ cd fiware/scripts/
hostdisk@hostdisk-07062012:~/fiware/scripts$ ls
install.sh    log
hostdisk@hostdisk-07062012:~/fiware/scripts$
```

As already mentioned in the installation guide, the script directory contains scripts to install a cloud proxy.

Testing an empty container locally:
We use container principle. So to use container, please verify that a sample test_vm is well installed in the system. The rootfs of the container is located at /var/lib/lxc/test_vm/rootfs. Check the path:

```
hostdisk@hostdisk-07062012:~$ cd /var/lib/lxc
hostdisk@hostdisk-07062012:/var/lib/lxc$ ls
test_vm
```

We can see that a rootfs “test_vm” exists. This rootfs could be used for creating and generating containers. Test if the container is running:

```
hostdisk@hostdisk-07062012:~$ sudo lxc-ls
test_vm
hostdisk@hostdisk-07062012:~$ sudo lxc-info -n test_vm
state:   STOPPED
lxc-info: 'test_vm' is not running
pid:        -1
hostdisk@hostdisk-07062012:~$
```

The container is created and ready to be started and not running.

Launch the container: Before developing any application, we must make sure the container works correctly. To do so, start the container. In the host, write the command lines:

```
hostdisk@hostdisk-07062012:~$ sudo lxc-start -n test_vm
Ubuntu 12.04 LTS test_vm console


test_vm login: ubuntu
Password: ubuntu
Last login: Thu Jun 14 15:40:20 UTC 2012 on lxc/console
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-24-generic-pae i686)


Documentation:  https://help.ubuntu.com/
ubuntu@test_vm:~$
```

From that point, execution is done into the container. The prompt of the terminal becomes ubuntu and the password used for connection is also ubuntu. Make sure the container has a valid ip address (ifconfig command), in our example, 10.0.3.243.
The container is running and now the environment is the container environment. All commands executed into that environment will be executed into the container and nothing will occur into the host rootfs. There is a clear isolation between the container environment and the host environment. From that location, the user can develop the desired application by using all Linux software facilities. Any application developed into that environment will run into the targeted container in the cloud proxy. It is possible to install any packages provided by Ubuntu, compiling, modifying the rootfs is necessary etc into that environment. In the container environment:
Ping google to make sur the ip connection works correctly.Quit the container environment:
To quit the container environment, write the command line:

```
ubuntu@test_vm:~$ sudo poweroff
[sudo] password for ubuntu:


Broadcast message from ubuntu@test_vm
        (/dev/lxc/console) at 16:00 ...
The system is going down for power off NOW!
ubuntu@test_vm:~$  * Asking all remaining processes to terminate...
[ OK ]
 All processes ended within 1 seconds....
[ OK ]
 Deconfiguring network interfaces...
[ OK ]
 Deactivating swap...
[fail]
umount: /run/lock: not mounted
mount: cannot mount block device /dev/disk/by-uuid/7165451d-88bb-
4d27-ae94-f2dfa4efa78b read-only
 Will now halt
```

```
hostdisk@hostdisk-08062012:~$
```

The new prompt is the prompt of the host user of the machine, in our case hostdisk.

2.3 Service start up

The application must be designed to start automatically once the container is started. When the image is downloaded to the cloud proxy and installed into the cloud proxy, the application must run at the start up of the container. There is several ways to launch a program automatically at start up. In Ubuntu/Debian environment, a common way is to create a new linux service and register it in the rc system. Please find here after a very simple example on how to do it. In the host environment:

```
cd /var/lib/lxc/test_vm/rootfs/etc/init.d
vi MyTest.sh
```

Copy the following text in MyTest.sh (replace <myexecutable> by the full path executable you want to launch)

```
#!/bin/bash
start()
{
        echo "start"
        myexecutable
}

stop()
{
        echo "stop"
}

restart()
{
   stop;
   sleep 1;
   start;
}
case $1 in
start)
     start;;
stop)
```

```
     stop;;
restart)
     restart;;
*)
     start;;
esac
```

You need to register this new service into the rc of the container. Either you can do it from the container or using a chroot command if you are in the host environment (see example below).

```
cd /var/lib/lxc/test_vm
sudo chroot rootfs /bin/bash
cd /etc/init.d
sudo update-rc.d MyTest.sh default 80
exit
```

2.4 Image creation

When an application runs into the container (in our example into the rootfs located in /var/lib/lxc/test_vm/rootfs), it becomes possible to build a so-called “image”. In this first release, the image corresponds to the rootfs of any LXC container compatible with the targeted Cloud Proxy.
To generate the image, you need to create a compressed tar file of the full system (make sure the container is stopped before doing any operation on the rootfs):

```
hostdisk@hostdisk-08062012:~$ sudo cp -ar /var/lib/lxc/test_vm .
hostdisk@hostdisk-08062012:~$ tar czvf image.tar.gz test_vm
```

The image is ready. Future additional information (metadata) will be added to this image bundle to specify dedicated features or requirements (ex: storage needed capacity, memory size required, access to local devices, ..), that will be used to configure the service on the Cloud Proxy (to be defined later in the next releases).

2.5 Store the image on a server

The file has to be stored on a public available HTTP or FTP server so that any client that needs to install this Application can download the file. This image file must be accessible via an HTTP or FTP request. This address will be used by the client application to download the image from the server on the cloud proxy.

2.6 Install the image in the cloud proxy

To install the image into the cloud proxy, we use the client application provided by Cloudedge. The client application is connected to the cloud proxy application. From that point, it is possible to install the image into the cloud proxy. For further details on how to register the image, please read the document “testing guide”.

2.7 Limitations and recommendations

Depending of the bandwidth available between the image server and the Cloud Proxy, it could take time to download the full image file on the cloud proxy. So, it is important to limit the size of the image: when designing a service based on a LXC container, a particular care must be taken to limit the size of the image by removing all the packages or files that are not necessary for running the service. Also considering that several of those LXC Services are supposed to run concurrently on small PCs or embedded devices with a limited set of resources (in term of CPU, memory, local storage, …), the design of those services should take those limits into consideration. A good implementation of a service, should consume a very little CPU, and limit the usage of memory.