Infrafon DataView Developer's Guide

This document describes overall workflows used to design and define communications between the Infrafon backend application server and the Infrafon device and the Infrafon dispenser. It does not provide detailed reference information or cover all available options – this information is available in the *Infrafon Developer Reference Guide.*

INFRAFON DATAVIEW DEVELOPER'S GUIDE - V. 1.0 - JANUARY 2022

infrafon
Infrafon GmbH
info@infrafon.com
+49 176 244 98 160
Bismarckallee 22
79098 Freiburg

Contents

Introduction to DataViews

An introduction to the way in which Dataviews fit into the overall Infrafon system. See the *Infrafon System Overview* for introductory information about the other elements of the Infrafon system.

Infrafon Terminology

Term	Meaning in Infrafon Context
Admin Portal	An easy-to-use web-based user interface
Allocation/ Enrollment	Process by which an Infrafon Device User is provided with an Infrafon Device configured with all needed apps and data.
Application Back-end	The application defines the specific customer process that the Infrafon system is being used for, such as visitor control, employee messaging or journey management.
DataViews	A collection of application data displayed on the Infrafon device. Consists of a number of InfoWidgets, grouped into one or more DataPages.
Digitalization	The process of using digital technologies to enable a workflow. Infrafon's smart terminal for the IoT world
Entity	The organization for whom the Infrafon system is being deployed. May not be the direct customer of Infrafon.
IFAP API	A REST API used for Platform manager. Allows management of users, devices, dispensers, kiosk modeling and uses tokens for application server connections. Generally not required for application integration
Infrafon Device	The physical handheld unit that guides the device user through the workflow

Infrafon Terminology

Term	Meaning in Infrafon Context
Infrafon Device User	The person using the device to interact with one or more applications. Typically (but not always) a visitor to a site.
Infrafon Entity Server (IES)	Provides the core communication and config/state setup for a specific installation using REST APIs and MQTT protocol
Infrafon Kiosk and Dispenser	Separate but linked units that store, charge, dispense and configure devices. Can you automatic or manual
JSON	JavaScript Object Notation: a lightweight data-interchange format
Kiosk REST API	Used by application to request allocation and return of device for given user.
MQTT	A lightweight opensource protocol designed for the Internet of Things (IoT)
Operator	The company responsible for the overall integration/project for the Entity.
System User	Person or Code accessing the IFAP through an Account

Introduction to DataViews

The Infrafon Device

The Infrafon System has the following parts:

The Infrafon Device

A sealed handheld battery powered unit with an ePaper display and touch buttons, and multiple radio networks, sensors, outputs.

The Infrafon Kiosk and Dispenser

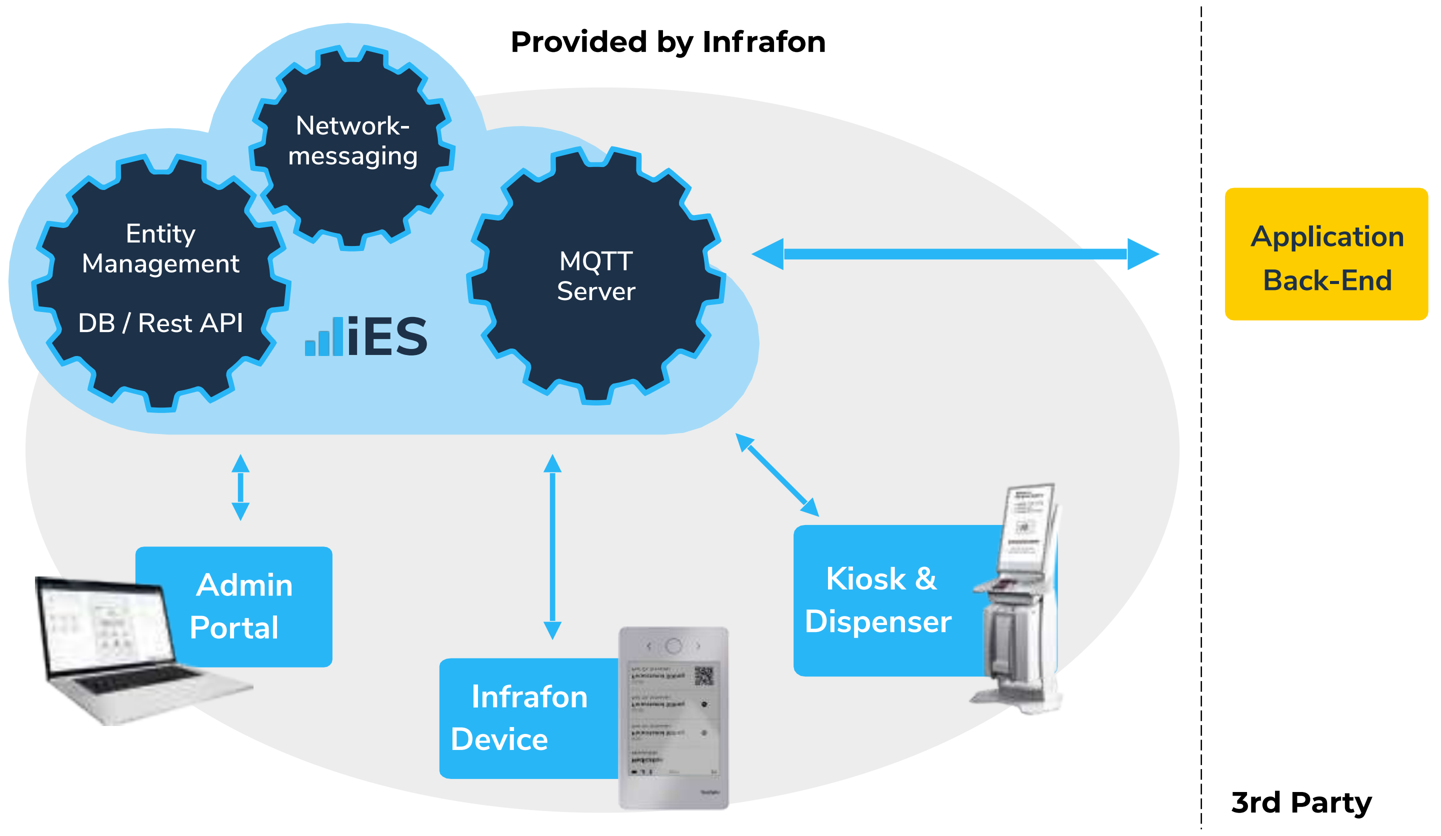
Stores, recharges, dispenses and configures devices.
Can be automated or manual

The Infrafon Entity Server (IES)

Provides the core communication and config/state setup for a specific installation using REST APIs and MQTT protocol.

The Admin Portal

The elements provided with Infrafon interact with an Application back-end, provided by third parties (ISVs or clients).



The Infrafon Device

Unlike a smartphone:

- No execution of application logic or functionality at device level
- No data processing on device

Instead:

- The device displays data that belong to applications running on the application back end.
- The device handles Input, validation, formatting and presentation

- Device-users can view data related to multiple different applications, and navigate between the them using the local UI management system
- The Home screen lets the user navigate between all active application data views, and display specific data in 'widgets' (home screen layout defined by the operator of the site)
- Developers build DataViews using the DataView Designer in the admin portal. These define:
- how to display and interact with data from a backend application
- how data exchanged using the MQTT protocol should be presented and/or requested on the device
- Developers can also hand edit the JSON file
- In practice, many prefer to create the initial definition using the DataView Designer, then refine it by editing the JSON file.

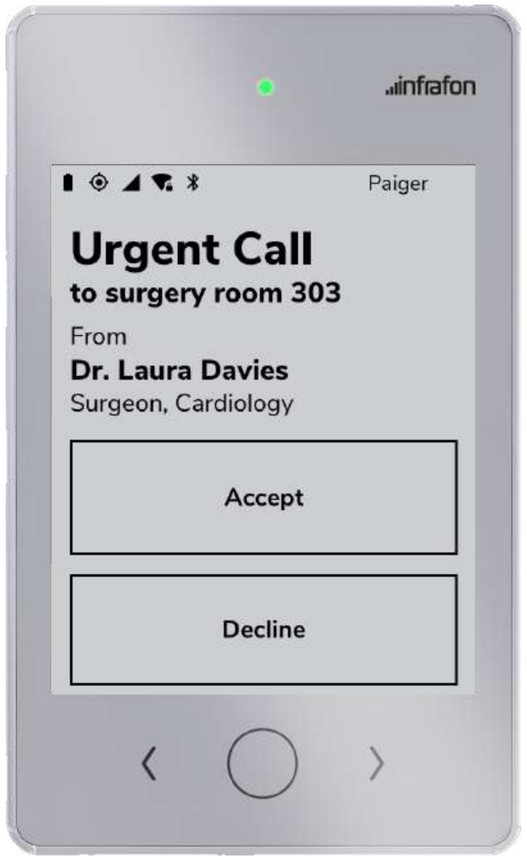
Example DataView Display

System bar

at the top of the display. Displays system icons and the current status, including the DataView name and the current page number. Always present unless a DataEntryDialog is being displayed.

InfoPage

occupies the rest of the display. Shows a single page of InfoWidgets, as defined by either an application or by the system info application.



InfoWidgets

These are the UI widgets that display a type of information, which is either essentially a text string, or a graphic image.

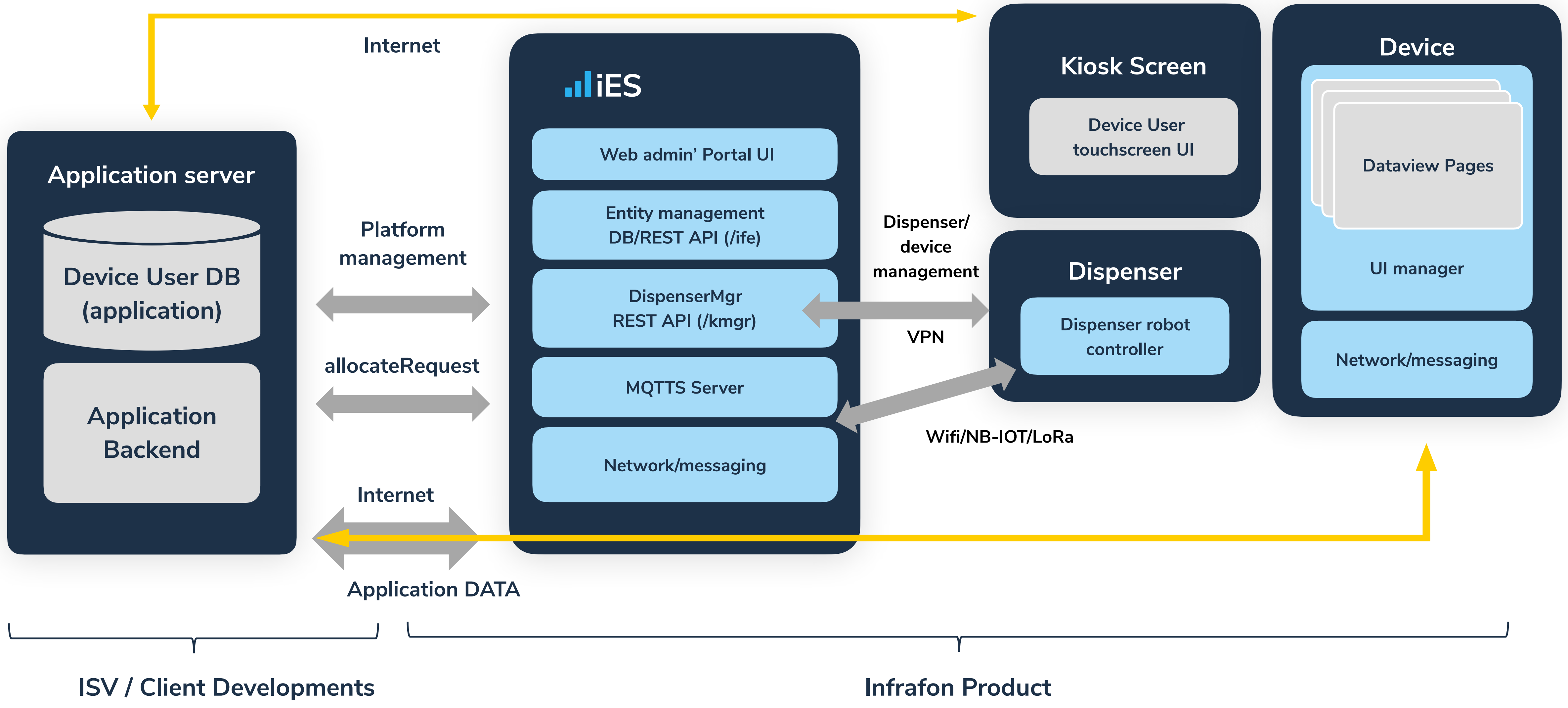
DataEntryDialog

a dialog box used to edit a value. Takes up the whole of the screen and obscures the System bar.

Navigation buttons

There are standard 3 navigation buttons, designated as “back”, “OK/Home” and “forwards”

Infrafon Architecture



Important Concepts

Before you start developing Dataviews you need to understand:

- The various component parts and how they contribute to determining what is displayed
- The relevant JSON file formats

Important Concepts

DataViews, InfoPages and InfoWidgets

A DataView is a collection of data, in the form of InfoWidgets, grouped into one or more InfoPages. DataViews define:

- Display content
- Display layout
- Permitted interactions (dynamic or not)

DataViews are created using a JSON definition of how to display and interact with data received from a backend application

You also need to define how data exchanged using the MQTT protocol is to be presented or requested to the device user through the User Interface.

InfoPages display defined groups of InfoWidgets

- Use the Infrafon UI/UX style
- Exchange data with the devices using the secured MQTTS server on the platform server (cloud or local installation)

InfoWidgets

- Are datatype specific (text, numbers, buttons, images.....)
- Data values can be fixed (for example: text, icons, etc) or dynamic (in general the application data)
- The display format is controlled by specifying the data type and a formatter
- Can trigger page changes
- Can send updates to backend applications

Important Concepts

Identifying Data

Important:

Every *data element* within the application scope must have a ***unique name***.

This name is used when referring to the data either in a data widget (to display it) or when setting / receiving its value via MQTT messages.

Unique names must be alpha-numeric ***only.***

Important Concepts

Integration with the Application Backend Communication, Formats and MQTT

Communication between the app and the DataView pages on the Infrafon device uses the MQTT protocol

The application must connect as an MQTTS client and subscribe/publish JSON format messages to specific topics. See the *DataView Reference Guide* for a full list of available topics.

Authentication using MQTT:

- The user / password required are defined in the web admin portal
- The user should be a valid 'app user', and have defined a permanent 'apikey' token
- This apikey should be used as the password for the connection
- clientId is the same as the user; note this means only 1 application can connect simultaneously to the MQTT server with this user id.

Topics to subscribe/publish to communicate app data to device

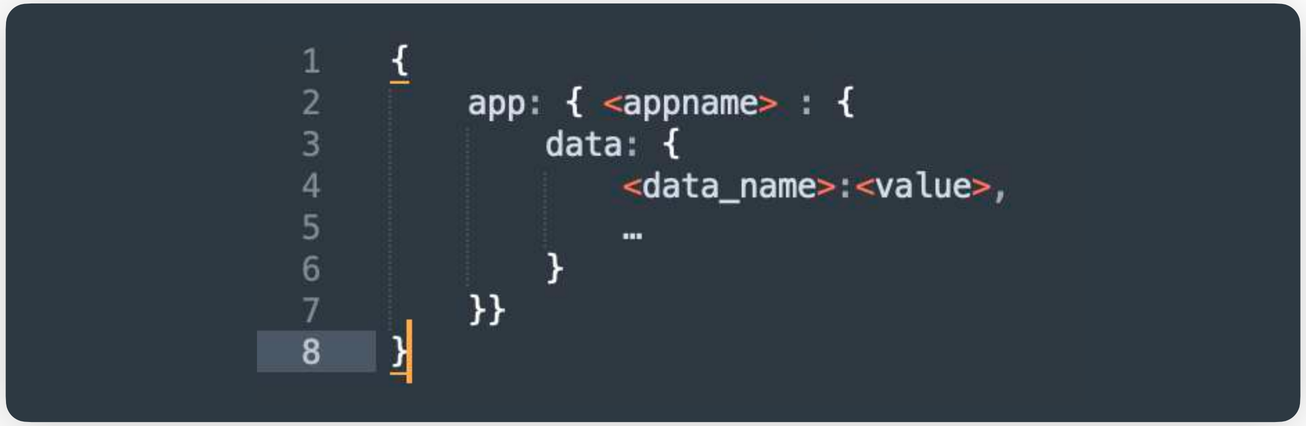
- /ife/app-up/<appname>/<deviceId>
- /ife/app-down/<appname>/<deviceId>

The MQTTS server carries out access validation so that only specific applications/devices that the given user has permission to see in the web admin portal are visible.

Important Concepts

Integration with the Application Backend
MQTT JSON message format

- Used for exchanges between the application and the Infrafon Device
- Contains data for a list of named data elements
- Each data element in an application scope must have a unique name
- Note device general sensor/outputs are also data elements with dev.X names
- Same format for sending and receiving data



```
{
    app: { <appname> : {
        data: {
            <data_name>:<value>,
            …
        }
    }}
}
```

Important Concepts

REST APIs

Platform Management (IFAP API)

- Used by the Infrafon Admin webportal
- Allows management of users, devices, dispensers, kiosk modeling
- Tokens for application server connections
- Generally, not required for application integration

Kiosk/Dispenser Management (Kiosk REST API)

Used by application to request allocation and return of device for given user

- User identified by visitorId (assigned by application) mapped to deviceId
- Either specific device (manual allocation) or 'best available' (dispenser)

Important Concepts

Kiosk REST API

The KIOSK REST API exposes end-points for both KioskUI and devices:

Kiosk end-points use the URL:

```
https://kmgr.infrafon.club/kiosk/<kioskId>
```

Device end-points use the URL:

```
https://kmgr.infrafon.club/device/<devId>
```

Device allocation endpoint:

```
https://kmgr.server.xx/device/<devid>/return (GET only)
```

Device return endpoint:

```
https://kmgr.server.xx/device/<devid>/allocate (GET only)
```

Important Concepts

Internal Device Architecture

DeviceUI Manager:

Manages the device's display. Centralizes access, draws widgets. Provides a high-level API for use at an InfoPage and InfoWidget level.

ApplicationUI Manager:

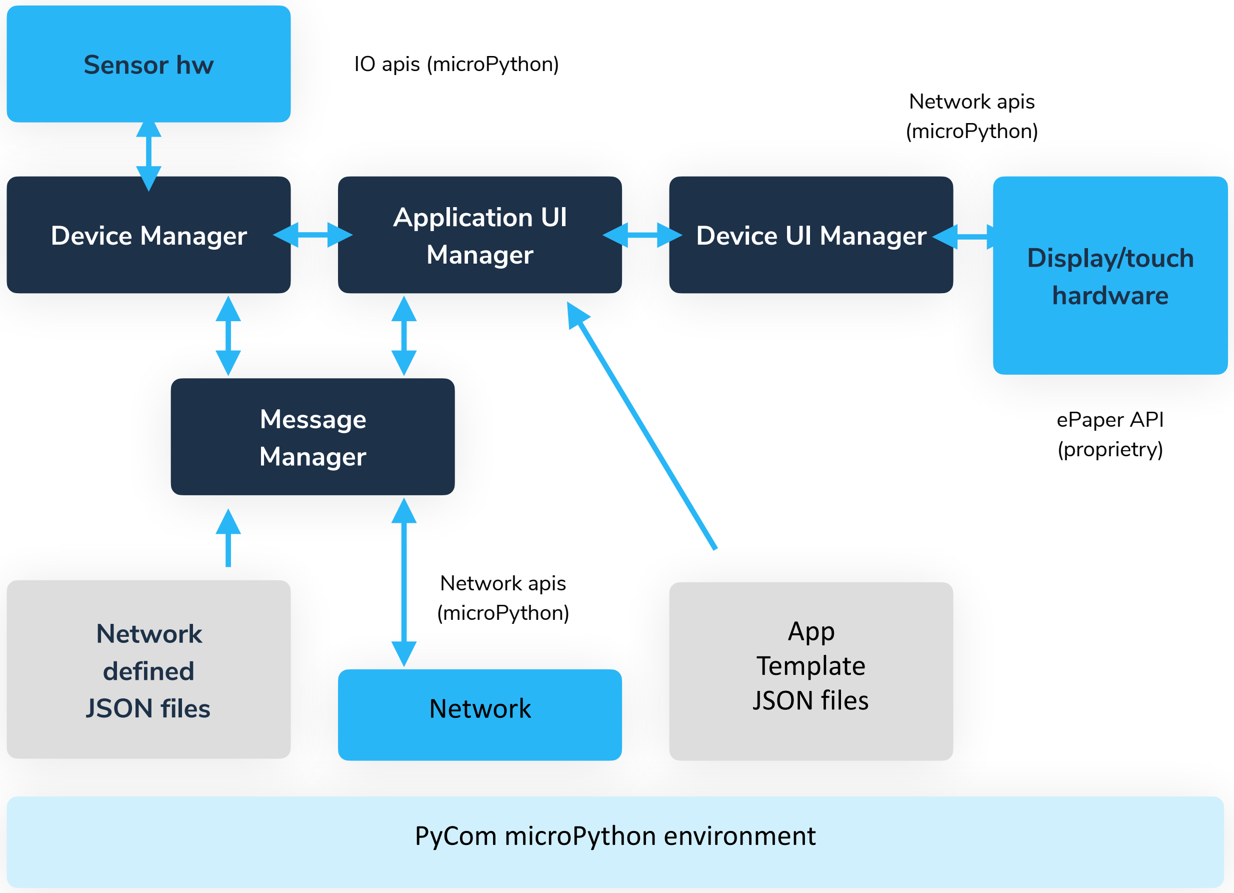
Parses the application UI template files and defines InfoPages displayed by the DeviceUI Manager. Updates pages when it receives messages from the network and transmits messages to the backend when the user enters new data.

DeviceManager:

Handles the operation of the hardware (LEDs, sensors, etc) and provides the data to the ApplicationUI Manager as required and pushes new data to the backend in the UL messages.

MessageManager:

Formats, sends and receives messages to the backend, selects networks and encodes messages as appropriate.



Creating DataView Pages

An overview of the steps necessary to define which data is displayed on a device and how it is laid out.

Refer to the *InfraFon DataView Reference Guide* for full information on syntax

Building Dataview pages : Using the DataView Designer

DataView Designer:

- The DataView Designer is available though the web-based admin portal user interface
- Use it to define the InfoWidgets you want to include in each DataView and determine their layout
  - ➡ Enforces respect of the UX operation, and of the touch button zones

InfoWidgets

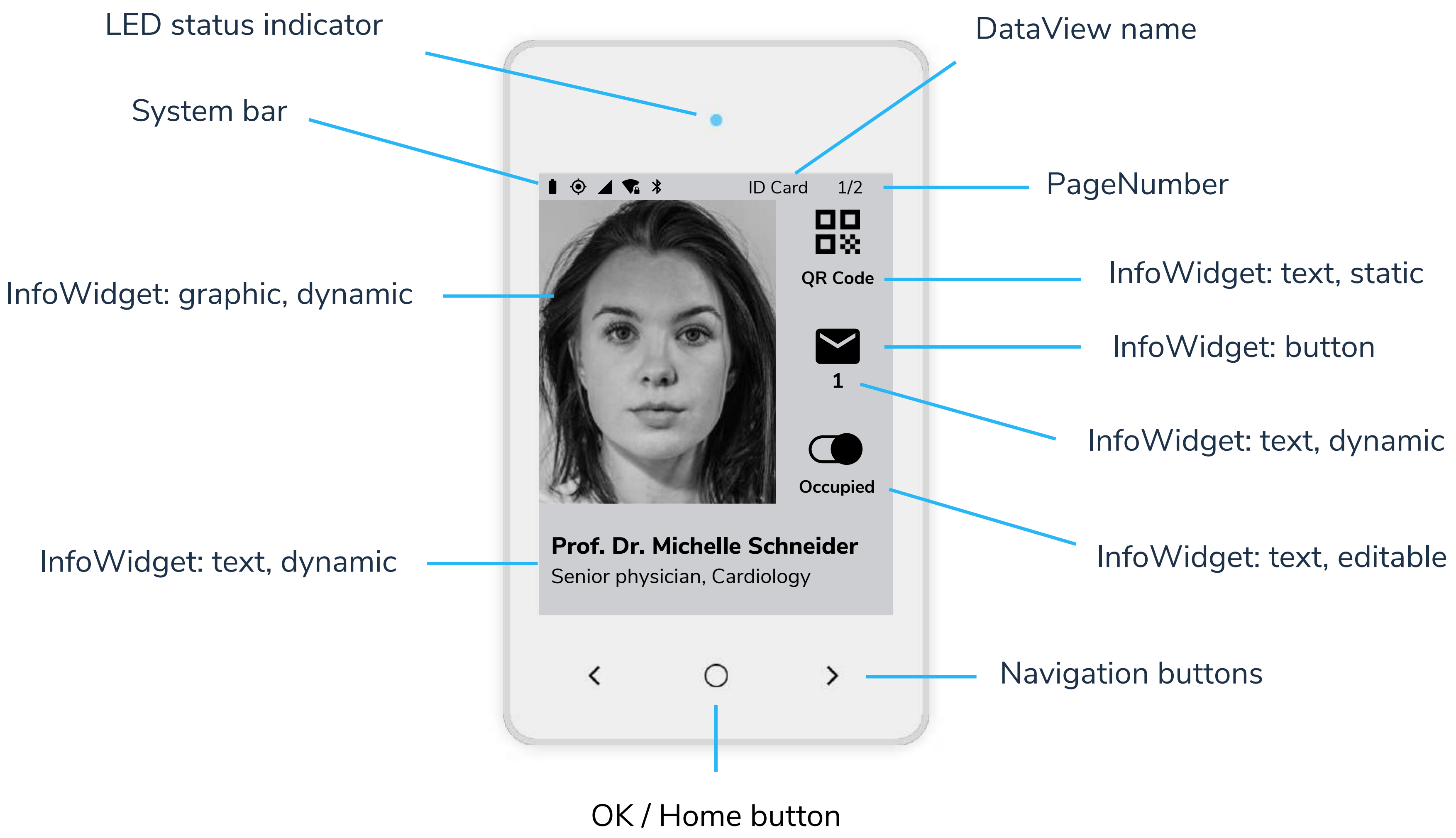
InfoWidgets define how data is displayed and/or entered by the device user, including:

- display numbers as integers or decimals
- Enter numeric values, or chose between a set of alternatives
- Map a boolean value to one of 2 icons and allow user to change it
- Display a graphic such as an id photo, or a data graph

Deployment:

- Once the set of DataViews for an application have been defined, it can be deployed to a device for a specific device user
- Different device users can have a different set of device views, individually or by group membership
- Also allows the creation of the home screen layouts, which can then also be allocated to device users individually or by group
- See the DataView Reference Guide for a complete list of available InfoWidgets

Example InfoWidget Types


LED status indicator
DataView name
System bar
ID Card
1/2
PageNumber
QR Code
InfoWidget: text, static
InfoWidget: graphic, dynamic
InfoWidget: button
1
InfoWidget: text, dynamic
Occupied
InfoWidget: text, editable
InfoWidget: text, dynamic
Prof. Dr. Michelle Schneider
Senior physician, Cardiology
Navigation buttons
OK / Home button

InfoWidget Values and Formatters

Each InfoWidget has a value. The value has:

- an underlying format of either ‘text’ or ‘graphic’ which defines how the value is to be interpreted,
- a ’formatter’, used to interpret the value to render it onto the screen.

Each formatter has a ‘name’ which defines

- its action,
- a set of possible attributes.

Note:

A “text” value could be either a string or a number and will be displayed depending on the formatter selected.

When the value is a text string, then it may be a key into a language specific dictionary, which then supplies the actual text to be shown.

See the DataView Reference Guide for a complete list of available formatters by format types.

InfoWidget Interaction types

The Infrafon System has the following parts:

Static:

Defined when the InfoWidget is defined and not modifiable during allocation or use of the Infrafon Device

Dynamic:

Defined when the InfoWidget is defined and not modifiable during allocation or use of the Infrafon Device

Editable:

The Device User can select the InfoWidget and change the value. The updated value can be sent to the application backend using the MQTT connection.

Button:

When a Device User touches and releases the area containing this InfoWidget, then the defined action will be taken. For example, navigate to a new page, run a questionnaire.

Dynamic and Editable InfoWidget Names

Dynamic and Editable InfoWidgets InfoWidgets must have variable names to allow:

- referencing by the caller to update their contents using the API
- auto update from received messages.

Note:

A set of reserved names beginning 'dev.' are variables defined by the device manager; these can be used for application specific InfoWidgets as well, in which case the value displayed is retrieved from the device manager.

In addition, **Editable** InfoWidgets must contain:

- the definition of a DataEntryDialog to use, including the relevant displayed text, help, value sets/bounds and so on. See the *DataView Developer Reference Guide* for more details.

Setting Data Values

Data values for each widget can be set from a variety of sources, depending on the specified Interaction type:

- **In the DataView definition**, as a default value
- **From initial user-specific data** loaded during device allocation process
    - ➡ Provided by the application at the 'user validation' phase
- **Dynamically from the backend** during use
    - ➡ Sending a JSON message to the
    - ➡ */ife/device/app-down/<deviceId>/<appname>* topic
- **From user entry:** if the widget is marked as 'editable', users can enter new values themselves
- **From device sensors or accessory communication**
    - ➡ from on board sensor eg GPS
    - ➡ from a BLE connected compatible accessory such as a barcode scanner

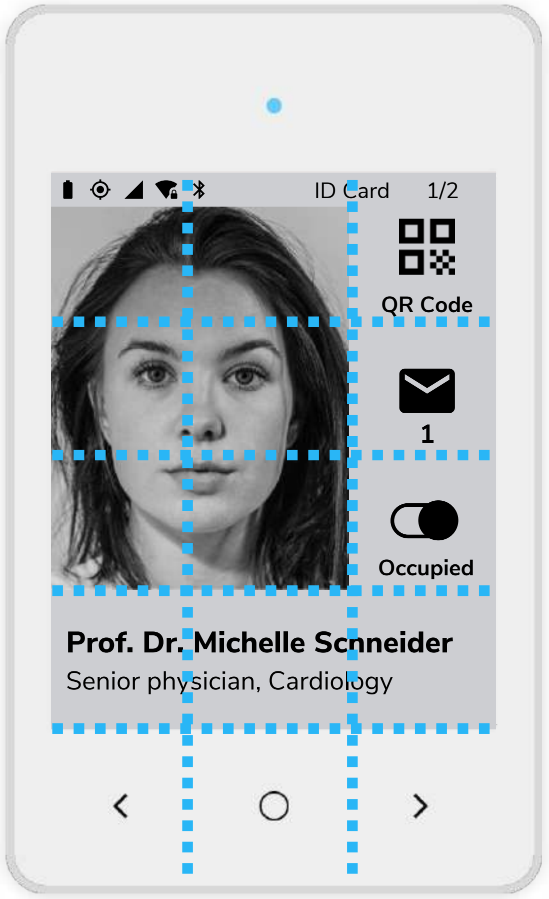
Whenever data is set, the InfoWidget is updated with the new value immediately

Available Interaction Sources	**Static**	**Dynamic**	**Editable**	**Button**
In the DataView definition, as a default value	✓	✓	✓	✓
From initial user-specific data loaded during device allocation process	✗	✓	✓	✓
Dynamically from the backend during use	✗	✓	✓	✓
From user entry	✗	✗	✓	✗
From device sensors or accessory communication	✗	✓	✗	✗

Design and Layout Tips

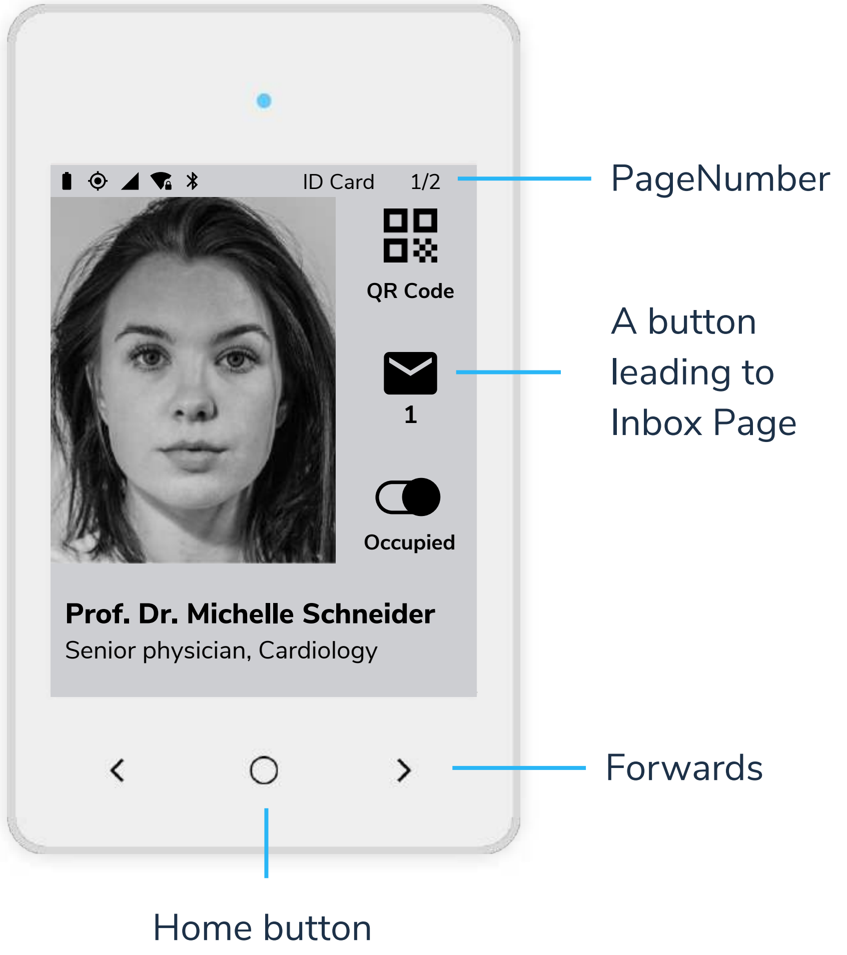
- To avoid ambiguity, place Editable or Button InfoWidgets so that there is only 1 per touch area. This is the responsibility of the UI-Owner designer. If several are present, the UI Manager will activate the first field it finds in the definition list.
- Use the largest font size that fits your text for readability
- Image updates must first be converted to a base 64 format using the tools provided.

Infrafon device with gridlines indicating touch areas



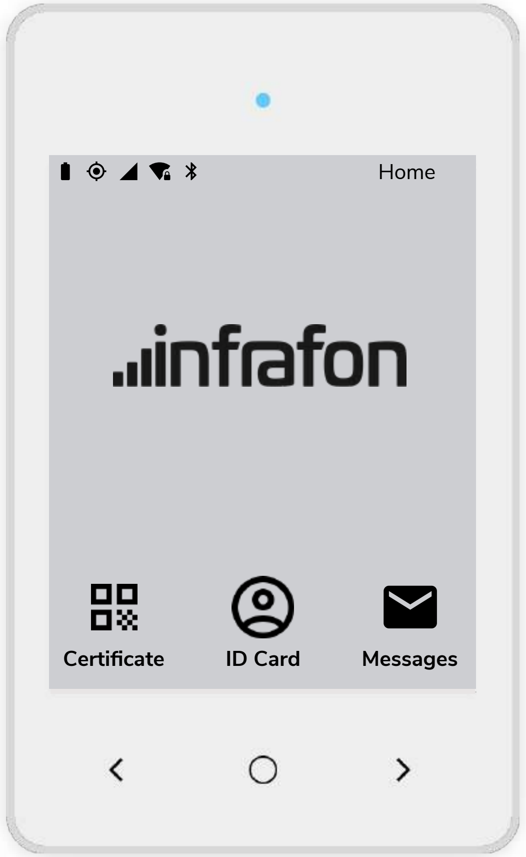
Navigation

- DataViews can contain an unlimited number of InfoPages.
- By default, the navigation buttons let you step through from one to the next in numbered order
- You can define button-type InfoWidgets that let you jump to a specific target location.
- The user can always use the Home button to return to the device home page.
- If there are multiple DataViews on the same Infrafon device, the Device Home Page lets you navigate between them.


ID Card
1/2
QR Code
1
Occupied
Prof. Dr. Michelle Schneider
Senior physician, Cardiology
PageNumber
A button leading to Inbox Page
Forwards
Home button

More about the Device HomePage

- The device home page is displayed on startup and is returned to using the OK/Home button.
- The default home page displays the Infrafon logo (or OEM logo if defined), and one button per application DataView available to the allocated user.
- You can also create a customized homepage creating a DataView called home. In this case its first infopage (id="1") acts as the home page.
- If no application DataViews are present on the device, the user's name and device id are shown. (For example, if the device is used only to run questionnaires pushed from the backend)
- If the device has not been NOT allocated, the home screen shows the device id, and a button to 'self allocate' the device to a user. In this case the backend must have already been configured with the relevant user data associated with the device id.



Example

The example shows how to integrate a ficticious application for use with the Infrafon device. Developing the application itself is out of scope of the Infrafon documentation.

Example Overview

The example shows how to integrate a (ficticious) application that handles visitor identification and control, including actions such as:

Creating a badge displaying the relevant information

Recording their information for the duration of the visit

Registering visitors at entry

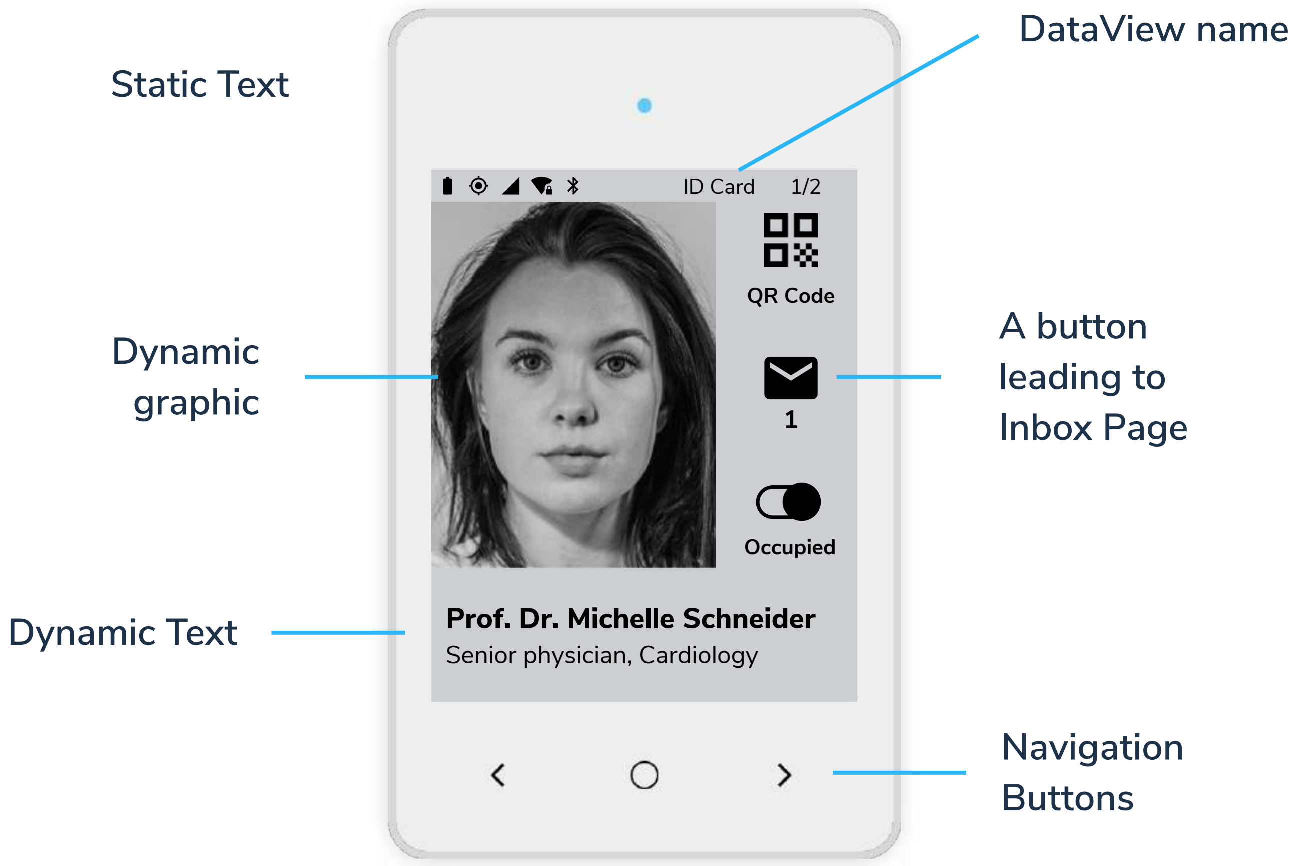
Deleting the information from the database at the end of the visit

The example shows how to use an Infrafon Device to act as a visitor badge. The initial version takes the simplest approach, displaying information that does not change during use. It then goes on to add other more complex features, including dynamic updates and device-user entered data.

Example

Creating a simple badge

- The simplest use of an Infrafon device is to display information that is not updated during use. A visitor badge is a typical example.
- In this example, the InfoWidgets used for the patient's name and photo, and the QR code are Dynamic because the displayed values (ie the name, photo and QR code) must be provided at device allocation time as part of the allocation request. This contrasts with the Static text, which is defined as part of the InfoWidget definition.
- The information for display is formatted in the JSON file as a single DataView page called MyCheckup



Example

Overall File Format

```
"displayName": "ID",
  "displayIcon": "ic_face_unlock_black_48dp",
  "infopages": [
    {
      "widgets": [
        {
          "infowidgets" : [
            {
              "interaction" : "static",
              "name":« InfoWidget1",
              "region": { "x" : 10, "y" :40, "w" : 460, "h" : 36 },
              "value" : "welcome-to",
              "format" : "text",
              "formatter" : {
                "name":"string", "valign": "center", "halign":"center", "autobreak":false,
                "font": { "sz":32 }
              }
            },
            {
              "interaction" : "dynamic",
              "name":"InfoWidget2",
              …
              }

            },
            {
              "interaction" : "dynamic",
              "name":"InfoWidgetN",
              …
              }
           },

          ]
        }
      ]
    }
  ],
}
```

See the *Infrafon DataView Developer Reference Guide* for full details

Example

Sample InfoWidget for static text

```
{ "displayName": "ID",
  „displayIcon": "ic_face_unlock_black_48dp",
  "infopages": [
    { "widgets": [ {
      "infowidgets" : [          {
         "interaction" : "static",
         "name":"NA",
         "region": { "x" : 10, "y" :40, "w" : 460, "h" : 36 },
         "value" : “Patient",
         "format" : "text",
         "formatter" : {
         "name":"string", "valign": "center", "halign":"center", "autobreak":false,
         "font": { "sz": 32 }
    }
    },
```

InfoWidget defined as **static**. The value supplied here is “hard-coded” and will not change. No name is required as there is no need to send updates to the InfoWidget during use.

Region defines ***where*** the content defined in the InfoWidget is displayed on the device screen. Each region is an imaginary rectangle.

Formatter defines ***how*** the content defined in the InfoWidget is displayed on the device screen.

Example

Sample InfoWidget for dynamic text

```
{ "displayName": "ID",
  "displayIcon": "ic_face_unlock_black_48dp",
  "infopages": [
    { "widgets": [ {
      "infowidgets" : [        {
"interaction" : "dynamic",
      "name":“lastname",
      "region": { "x" : 10, "y" :80, "w" : 460, "h" : 60 },
      "value" : "",
      "format" : "text",
      "formatter" : {
      "name":"string", "valign": "center", "halign":"center", "autobreak":false,
      "font": { "sz":52 }
    }
    },
```

InfoWidget defined as **dynamic.** The value here is must be supplied by the application at allocation time for each user as part of the allocation request. As it is dynamic, this InfoWidget has a name.

Region defines ***where*** the content defined in the InfoWidget is displayed on the device screen. Each region is an imaginary rectangle.

Formatter defines ***how*** the content defined in the InfoWidget is displayed on the device screen.

Example

Sample InfoWidget for graphic

```
{
  "interaction" : "dynamic",
  "name":"idphoto",
  "region": { "x" : 10, "y" :150, "w" : 210, "h" : 300 },
  "value" : " ",
  "format" : "graphic",
  "formatter" : {
"name":"basic", "valign": "center", "halign":"center"
  }
  },
```

InfoWidget defined as **dynamic.** The value here is must be supplied by the application at allocation time for each user as part of the allocation request. As it is dynamic, this InfoWidget has a name.

Region defines ***where*** the content defined in the InfoWidget is displayed on the device screen. Each region is an imaginary rectangle.

Formatter defines ***how*** the content defined in the InfoWidget is displayed on the device screen.

Example

Graphical InfoWidgets : bitmap data

- The graphics data for an icon must be provided in a specific format:
  - ➡ PBM format
  - ➡ GS4 (gray scale 4 bits) type
- Convert standard PNG graphics using the Infrafon tool
- Standard icons are provided ‘built in’ to the Infrafon (see list in the DataView designer) and are referenced by setting their ‘id’ in the ‘value’ field of the InfoWidget
- Custom icons, or dynamic graphics must have their data provided in the ‘value’ field as a base64 encoding of the GS4 format binary file

Example

Adding a Second InfoPage

- The initial example has a single InfoPage. However it is simple to add new pages, as the ‘ infopage’ element is an array:

```
"infopages": [
 {"id": "1", "widgets": [ …infowidgets for page 1… ]
 },
 {"id": "2", "widgets": [ …infowidgets for page 2… ]
 },
 {"id": "detailinfo", "widgets": [ …infowidgets for page ‘detailinfo’… ]
 },
…etc
]
```

- You can define as many InfoPages as you like for your application
- By default the navigation buttons let users move from one page to the next in numbered order (the ‘id’ field).
- Pages with non-sequential or non-number ids can be accessed only using specific buttons in another page. If the id field is not present, then the pages are assumed to be numbered in order. Id’s must be unique within the DataView.

Example

Jumping to a Specific Page

- If you want to jump from one page to the next, you need to define a button InfoWidget on the source page, giving the page ID of the destination page.
- The button InfoWidget below lets a user jump to the ‘detailinfo’ InfoPage in the previous example.

```
…"infowidgets" : [{
       "interaction" : "button",
       "onEntry":"gotoPage",
       "target":“detailinfo",
       "region": { "x" : 160, "y" :150, "w" : 100, "h" : 100 },
       "value" : "keypad_1_100dp",
       "format" : "graphic"
          },…
```

- The navigation buttons continue to work as default. The user can always use the Home button to return to the device home page.
- Note this button is an icon, so has format ‘graphic’ and its ‘value’ references the name of a Infrafon provided graphic element. Alternatively, the ‘value’ could be a base64 encoding of the graphic file in a PBM format (see the DataView Developer Reference Guide.)

Example

Multi-lingual dataviews : using dictionaries

- For static text (buttons, labels, etc), it is useful to be able to customise the language used
- Instead of creating a separate DataView project per language, you can use a 'dictionary' for each language which maps the static value (eg ' surname_field') to a localised text string (eg 'LastName' for English, or 'NachName' for German).
- Device language is selected using the 'dev.lang' attribute : using the 'system' DataView, or by setting the value from the application backend
- Add the dictionnaires in the dataview for each language:

```
"dicts": [
  { "lang":"en", "description":"English",
    "dict":{ "welcome-to":"Welcome to"  } },
  { "lang":"fr", "description":"Francais",
     "dict":{ "welcome-to":"Bienvenue à"  } },
  { "lang":"de", "description":"Deutch",
     "dict":{ "welcome-to":"Welcome zu" } }
]
```

- The InfoWidget for the static text label with a "value":"welcome-to" will then show the relevant text from the dictionnary for the currently selected language (dev.lang attribute)

Entering and Updating Data from the backend

MQTT publish to push data and Data Entry Dialogs

Updating Information While the Device is in Use

The values of named InfoWidgets can be updated dynamically while the Infrafon Device is in use.
For example, the below InfoWidget shows the name of a meeting room:

```
{ "interaction" : "dynamic",
  "name":"meetingroom",
   "value" : "room 1",
  "format" : "text",… region and formatter to display text..
}
```

If the meeting location changes after the visitor's arrival, you can push the new location to the badge (named by its device-id) by sending an MQTT message to:

```
/ife/app-down/myvisitorapp/<device-id>
```

With the JSON content:

```
{"app":"myvistorapp" {"data": {"meetingroom" : "room 3"  } } }
```

You can update any named field in a dataview.

Alerting the device user

To attract device user's attention to an update, it is useful to send an audible notification, and/or make the device vibrate. This is done by updating the values of the named 'dev' attributes: ' `dev.led`', '`dev.buzzer`', or '`dev.vibration`'

Example MQTT message:

```
{ "app":{ "dev":{
   "led": { "colour":"0x0000FF"},  -> LED is turned on bright Blue
   "buzzer": { "song":"bb01c-01ab01Ab05Eb10"},  -> plays a short tune
   "vibration": { "song":"V10S10V10"},     -> whilest vibrating to this rythmn
} } }
```

See the *Infrafon DataView Developer Reference Guide* for information on the format for the values.

Using Editable Fields

Data Entry Dialogs (DEDs) let device users input information:

- By tapping on a modifiable InfoWidget
- By responding to a question

Below is the syntax of an InfoWidget that references a DED:

```
{"name":"myvar1", "value":"1", "interaction":"editable",
  "region":{ "x" : 25, "y" :150, "w" : 100, "h" : 100 },
  "format":"text",  "formatter" : {"name":"number", "dp":0,"font":{ "sz":22 } },
  "ded": {
        "type":"int_keypad", "title":"Type a int", "range": { "low":0, "high":99999}
  }
},
```

When a Device User touches this InfoWidget, the dialog ‘int_keypad’ opens to allow the entry of an integer number using a keypad.

You need to select the kind of DED that allows users to enter the appropriate information. See the next slide for available DED types.

Entering and Updating Data from the backend

Data Entry Dialog Types

DED Type	Use
`int_keypad`	A numeric keypad for entering integer numbers
`float_keypad`	A numeric keypad with decimal point for entering floats
`int_stepper`	Increment'/'decrement' buttons for entering integer
`bool_slider`	Aslider graphic to select 'true' or 'false values
`bool_yesno`	A pair of buttons to select a yes or no value
`1oflist_4choice / noflist_4choice`	A list of up to 4 choices shown 1 per line, of which either only one or multiple can be selected
`1oflist_12choice / noflist_12choice`	A list of up to 12 choices shown as buttons, of which either only one or multiple can be selected (inverted colour)

Entering and Updating Data from the backend

Updated Data

- Once modified, a data value can be used in 2 ways
  - ➡ To update one or more display InfoWidgets on the InfoPages of the DataView
  - ➡ Sent to backend platform using MQTT
- The application receives the updated value using an MQTT subscribe request to application specific topic:

```
/ife/device/app-up/<deviceId>/<appname>
```

- Same JSON format message is used to transport data
- You must use the application data element 'name' defined by developer
- It is also possible to receive updated sensor data via a generic ' device' topic:

```
•/ife/device/system-up/<deviceId>
```

Device-controlled Data Updates

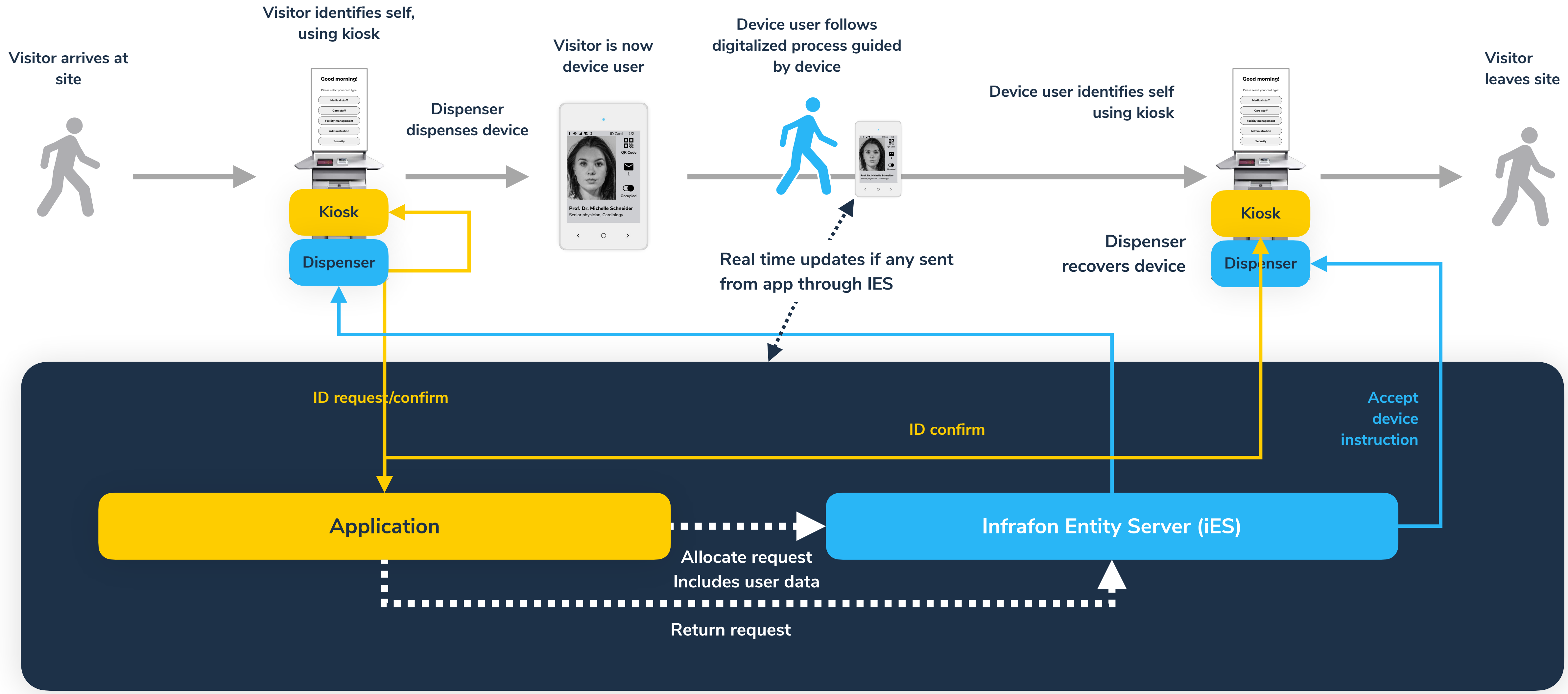
- Sensor data updates (for example movement, temperature, GPS position) are controlled by the device
- The frequency of collection is defined by the device configuration (set at allocation time)
- The backend application can force a read if required (for example, for a BLE scan or GPS position) : send a MQTT message as though to set the data 'dev.<X>' with an empty value, and the new value is sent back.

Allocating a Device : the Kiosk REST API

You need to use the Kiosk REST API to allocate an Infrafon Device to a user of an application. You therefore need to define how this will be done for your DataView pages.

Allocating a Device : the Kiosk REST API

Infrafon Workflow Overview



Visitor arrives at site
Visitor identifies self, using kiosk
Good morning!
Medical staff
Care staff
Facility management
Administration
Security
Kiosk
Dispenser
Dispenser dispenses device
Visitor is now device user
ID Card
QR Code
Occupied
Prof. Dr. Michelle Schneider
Senior physician, Cardiology
Device user follows digitalized process guided by device
Real time updates if any sent from app through IES
Device user identifies self using kiosk
Dispenser recovers device
Visitor leaves site
ID request/confirm
ID confirm
Accept device instruction
Application
Infrafon Entity Server (iES)
Allocate request Includes user data
Return request

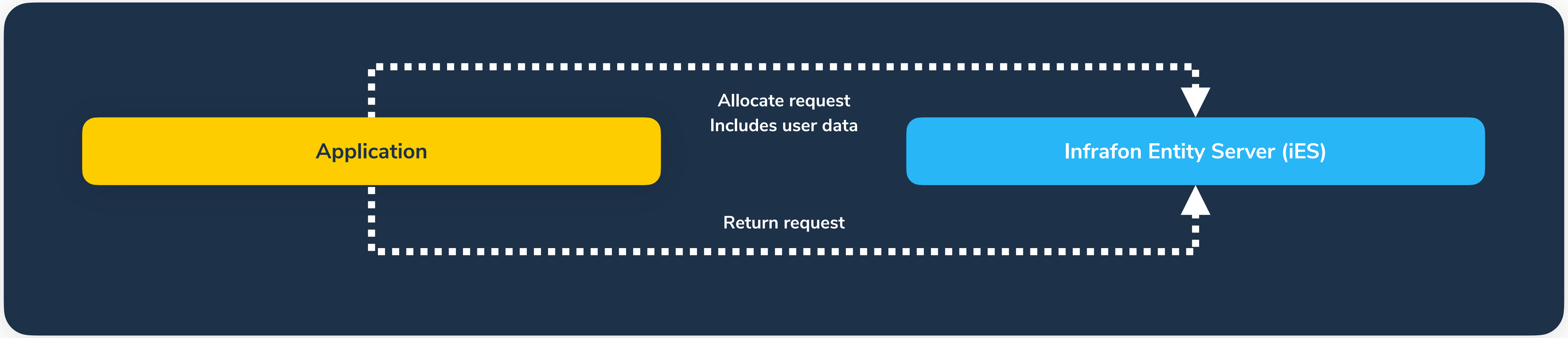
Allocating a Device : the Kiosk REST API

Enabling Allocation for a Specific User

The precise actions depend on the specific application you are integrating. You need to:

- Identify the user in the backend application.
- Ensure the application requests device allocation using the kioskmgr REST api
- The allocation request should contain the list of DataViews to load on the device for this user, and any application specific data to set for them.

For example, in the sample application, the self-service kiosk identifies the visitor, then sends this information to the IES using the Kiosk Manager REST API (allocate request)



The allocation JSON parameter

Your application must generate the “userdata” parameter in a JSON format equivemant to that used for the MQTT messages. All dataviews that are to be available on the device must be listed in the json, even if no specific data is given.

Example:

```
Userdata = {
  "app": {
           "id_card": { "data":{
                            "firstname":"FirstName",
                            "surname":"Surname",
                            "jobtitle":"JobTitle",
                            "company":"Company" }
                       },
            "system" : {},
            "lunchrequest" : {}
  }
}
```

Creating Questionnaire-type DataViews

You can use the Data Entry Dialog described earlier to create Questionnaire-style InfoPages – where Device Users respond to questions.

Including a Questionnaire in an application

- You create questionnaires using the Data Entry Dialogs described earlier.
- Questionnaires can be
  - ➡ defined in advance (at allocation time), an included in the dataview json file (questionnaires attribute),
  - ➡ sent from the backend as an MQTT update.
  - ➡ The json format is the same in both cases.
- Execution of the questionnaire is triggered by one of the following:
  - ➡ An explicit request from the backend using MQTT (this can be in the same message as a newly defined questionnaire or separate)
  - ➡ A Button InfoWidget on an InfoPage
  - ➡ An Alarm time set in the questionnaire
- Questions can have rules associated with them to select the question flow depending on the values entered.

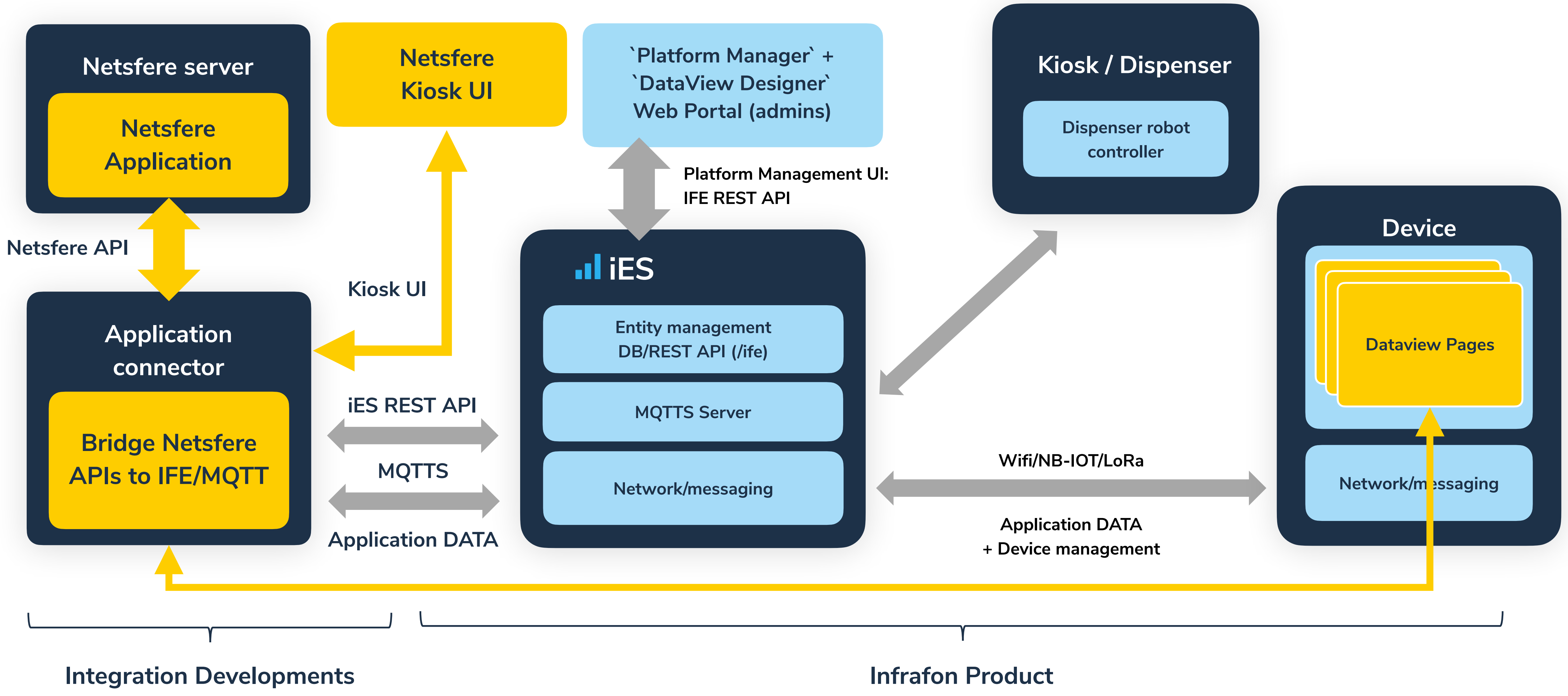
```
"questionnaires": [ { "id":"qlunch", "questions": [

  { "name":"starter", "value":"salad:food-salad:1,terrine:food-terrine:0, soup:food-
soup:0",

       "ded": {  "type":"1oflist_4choice", "title":"q-starter" } },

  { "name":"dessert",  "value":"yogurt:food-youg:1,choco:food-cpud:0, baba:food-
baba:0, panna:food-panna:0, glace:food-ice:0, canelle:food-canelle:0, apple:food-
appletart:0, flan:food-flan:0, banana:food-banana:0, painperdu:food-painperdu:0",

       "ded": {"type":"1oflist_12choice", "title":"q-dessert"},


"rules": [{ "type":"ifSelected", "option":"glace", "target":"ice_cream_flavours"} ],

       "next":"coffee" },

  { "name":"ice_cream_flavours", "value":"vanilla:food-vanilla:0,choc:food-
choc:0, straw:food-straw:0",

       "ded": {"type":"noflist_12choice", "title":"q-icecreams" }},

  { "name":"coffee", "value":true,

       "ded": {"type":"bool_slider", "title":"q-coffee" },

       "rules": [{ "type":"ifF", "target":"tea"} ],

       "next":"end"},

  { "name":"tea", "value":true,

       "ded": { "type":"bool_slider", "title":"q-tea" }   }

 ] } ]
```

Integrating a 3rd party application

Netsfere example

Integrating a 3rd party application

Netsfere messaging example



Integrating with Netsfere

The Netsfere product supports secure messaging. It exposes an API to set a net hook to receive notifications.

Integration requires

- Code to support self service device allocation request
  - ➡ Registration with Netsfere for that user
  - ➡ Allocation of device with appropriate Netsfere DataView and user-specific data
- Creation of DataView to display user name, current message and to select a reply for the current message
- Code to interact between Netsfere server and Infrafon MQTT service.