Provisioning E2E Integration

Advantech Integration Camp

Confidential © Arm 2017 Limited

Overview

Confidential © Arm 2017 Limited

arm

Overview

- **Factory Configurator Utility** (FCU): Generates a *device configuration bundle* containing parameters, keys and certificates for Mbed Cloud connectivity.
- **Factory Configurator Client** (FCC): Application that is a counterpart of FCU. It runs on the device, processes and verifies the device configuration bundle, then stores its components in secure storage on the device.
- The FCC relies on the **Key and Configuration Manager** (KCM), a stand-alone C library. KCM runs on the device and stores parameters, key and certificates in the device's secure storage. It can also allow applications to access these parameters, keys and certificates. FCC uses the KCM to store the elements of the device configuration bundle.

Confidential © Arm 2017 Limited

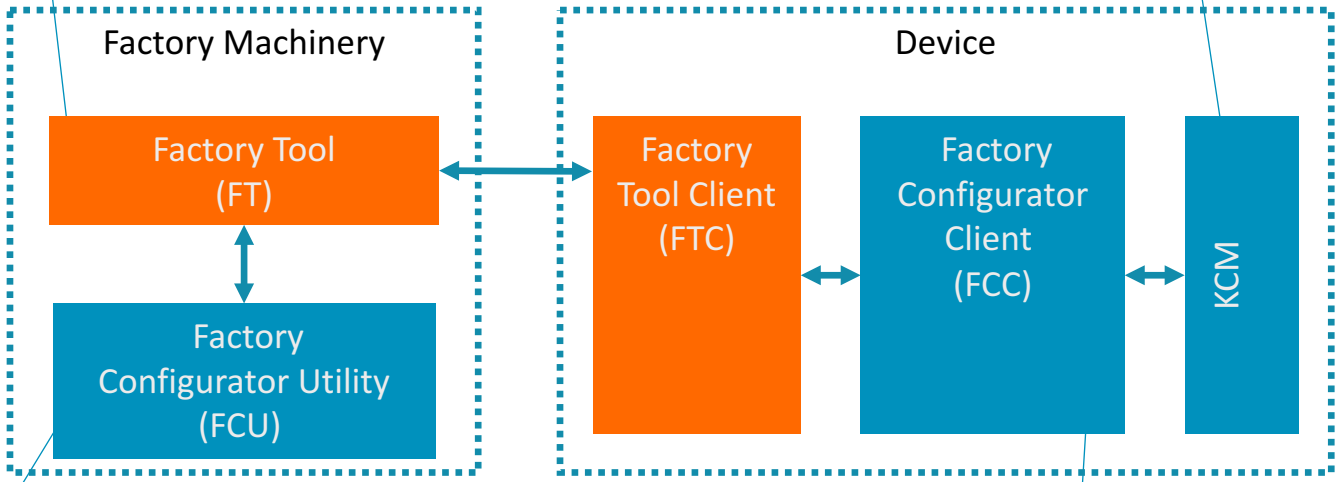
Factory tool and Factory Configurator Client

Commonly used by the manufacturer for one or more of the following:

- Program the software image
- Configuring the device parameters
- Calibrations
- Testing

May use variety of transport protocol to communicate with the device.
May require a client on the device.

Key & Configuration Manager (KCM) stores the keys and certificates



A Python library for configuring the device with parameters, key material and certificates required for connecting with Mbed Cloud.

Expected to be integrated to the Factory Tool for the purpose of configuring the device for Mbed Cloud

Ingest the security information prepared by the Mbed Factory Configurator Utility and configure it into the Key & Configuration Manager (KCM)

Mbed Product component

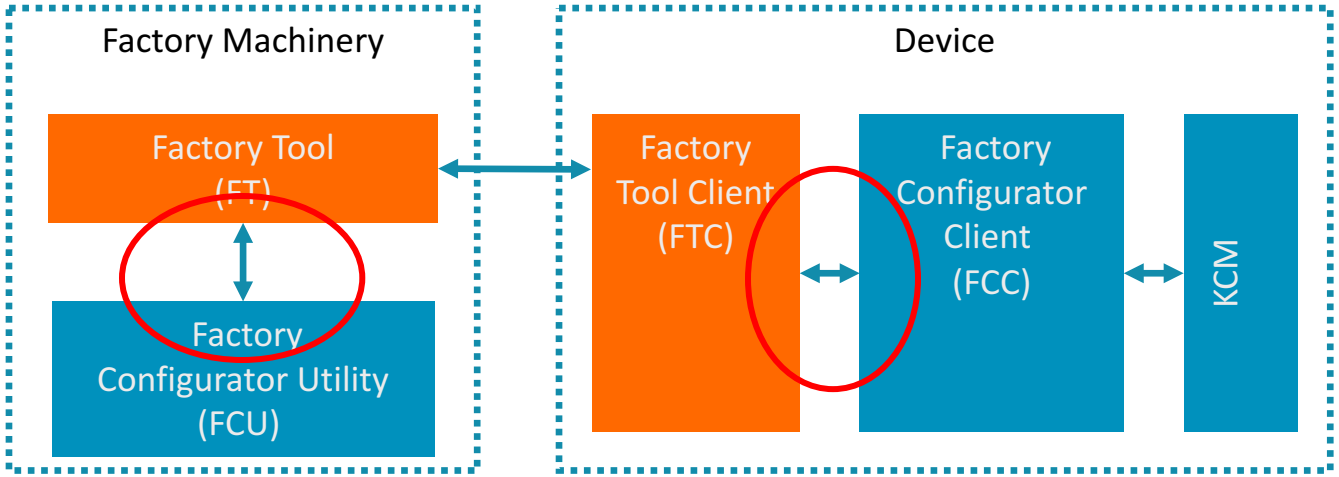
Customer component (Mbed example component)

Confidential © Arm 2017 Limited

Session Focus

- This presentation focuses on integration between FT and FCU as well as between FTC and FCC as described below:

	Factory Machinery				Device	
Example #	Key Generation	Certificate Signing	Validating by FCU	Delivery to device	Validating by FCC	Storing in KCM
1	External Input	External Input	No	FT[1]	Optional: FTC to call	FTC[1]
2	External Input	External Input	Yes	FT[2]	Yes	FCC[2]
3	FCU	FCU	Yes	FT[2]	Yes	FCC[2]
4[3]	Device	Sent to External CA	Yes	FT[2]	Yes	FCC[2]



Confidential © Arm 2017 Limited

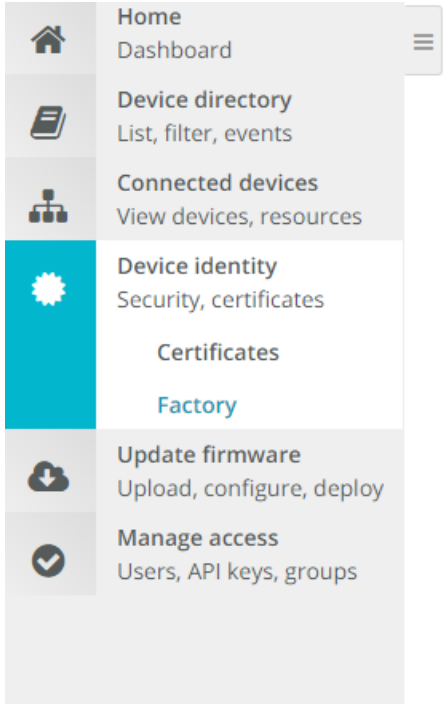
FT-FCU (PC Side) Integration

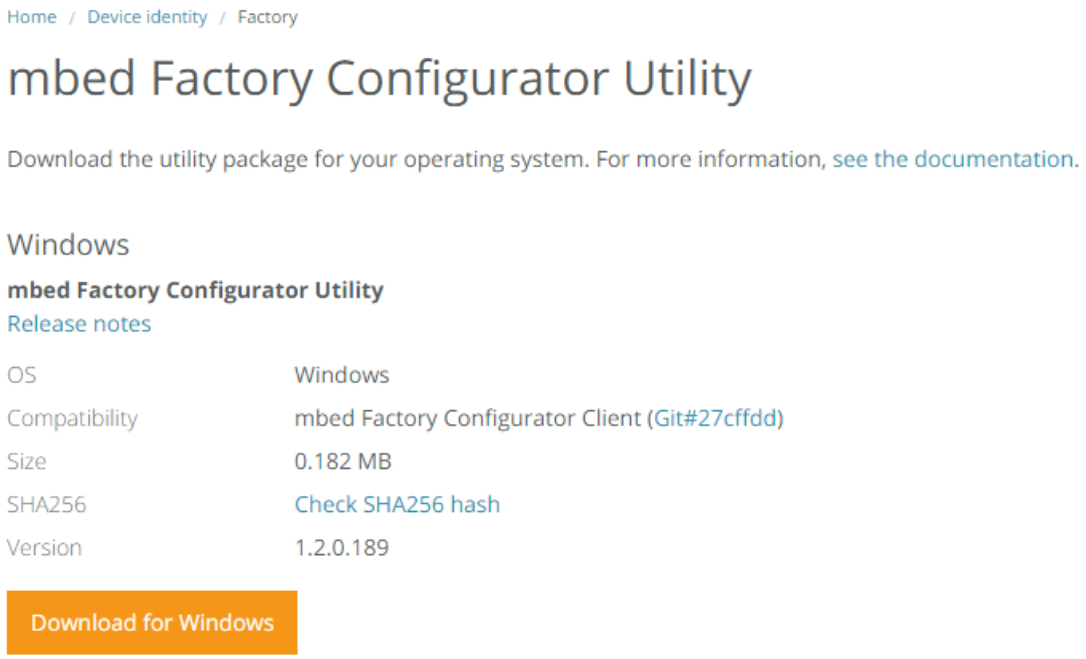
Confidential © Arm 2017 Limited

arm

FCU - Download

- Go to Mbed Cloud Portal: https://portal.us-east-1.mbedcloud.com/login and login
  - FCU is located under Device identity → Factory
- In portal, download the FCU:





Confidential © Arm 2017 Limited

FCU Structure

```
+-fcu_archive/                              # Extracted archive root
        |
        +-fcu/
        | +-fcu-0.0.1-py3-none-any.whl              # The FCU Python package
        | +-sources/
        |   +-fcu/                                  # The FCU source code (Python)
        |   +-requirements.txt                       # The Python packages required for the FCU
        |
        +-config/
        | +-fcu.yml                                 # Configuration file for FCU
        |
        +-resources/                                # Resources used by FCU
        | +-bootstrap_server_ca_cert.pem            # The bootstrap server certificate
        | +-lwm2m_server_ca_cert.pem                # The LwM2M server certificate
        |
        +-ft_demo/                                  # Demonstration of a factory tool that uses FCU
        | +-sources/
        |   +-ft_demo/                              # The factory tool source code (Python)
        |   +-requirements.txt                      # The Python packages required for the factory tool
        |
```


Confidential © Arm 2017 Limited
arm

FCU main configuration steps

1. **YAML configuration**

   - FCU requires a configuration file in YAML format (fcu.yml)

2. **Setup**

   - Make the FCU a CA , generate a private key (fcu_key.pem) and certificate (fcu.crt) under <FCU DIR>/keystore

3. **Inject**

   - Generate device configuration bundle (using FCU) and inject it using Factory Tool to the device.

Confidential © Arm 2017 Limited

Environmental variables

Using environment variables

The FCU configuration file allows the use of environment variables by using `<%= ENV['ENVVAR_NAME'] %>`, where `ENVVAR_NAME` is the name of your environment variable.

For example:

```
bootstrap-server-ca-certificate-file: <%= ENV['FCU_RESOURCES_DIR'] %>/bootstrap_server_ca_cert.pem
```

Some environment variable names are reserved by the FCU. When absent, default values will be used, as follows:

Variable	Description	Default
`FCU_HOME_DIR`	Specifies the home directory of the mbed factory configurator library. The library uses this as the root path of several directories and files it depends on.	The current working directory
`FCU_RESOURCES_DIR`	Specifies the resources directory of the mbed factory configurator library	`/<FCU Home>/resources`
`FCU_KEYSTORE_DIR`	Specifies the keystore directory of the mbed factory configurator library.	`/<FCU Home>/keystore`

Confidential © Arm 2017 Limited

Yaml Configuration

- Yaml configuration file as downloaded from the portal

- Most of the fields are empty, but some of the information is already filled
- The file needs to be filled and/or modified.
  - Details in the next slides

Confidential © Arm 2017 Limited

Yaml Configuration - continue

Functional parameters

These parameters influence the mode of operation of FCU. One example is the key generation mode:

- `device-key-generation-mode`: Defines the source of device keys (device DTLS private key and certificate). This is **mandatory**.
  - `externally_supplied`: Device keys that you supply, and which the tool expects to find under `device-keys-location` (an API parameter).
  - `by_tool`: Device keys that the tool generates.
- `use-bootstrap`: Use bootstrap server (true/false). This is **mandatory**. If it is false, LwM2M is used.

D3: On boarding via bootstrap server?

A: On-boarding via bootstrap

```
use-bootstrap: true
```

B: On-boarding directly to LwM2M?

```
use-bootstrap: false
```

D4: Source of device keys and certificates?

A: Use FCU

```
Device-key-generation-mode: by_tool
```

B:"Bring your own"

```
Device-key-generation-mode: externally_supplied
```


Confidential © Arm 2017 Limited

arm

Yaml Configuration - continue

`time-sync`: Sends time to device for verifying certificate time-stamps

D2: Setting Device RTC

A: Use FCU

```
time-sync : true
```

B: Do it Yourself (DIY)

```
time-sync : false
```


Confidential © Arm 2017 Limited

arm

Yaml Configuration - continue

FCU as CA configuration

These parameters make up the subject of the generated CA certificate. They are aggregated under the **certificate-authority** YAML section:

- `common-name`: The **CN** subject field.
- `organization-name`: The **O** subject field.
- `organizational-unit-name`: The **OU** subject field.
- `locality-name`: The **L** subject field.
- `state-or-province-name`: The **ST** subject field.
- `country-name`: The **C** subject field.

Example:

```
certificate-authority:
        common-name: Sample Cert
        organization-name: Company Ltd.
        organizational-unit-name: Mbed Factory
        locality-name: Madison
        state-or-province-name: Wisconsin
        country-name: US
```


Confidential © Arm 2017 Limited

Yaml Configuration - continue

Bootstrap configuration

When `use-bootstrap` is true, use this configuration:

- `bootstrap-server-uri`: The bootstrap server's URL. For example, `coaps://bootstrap.arm.com`. This is **mandatory**.
- `bootstrap-server-ca-certificate-file`: File location for the bootstrap server's CA certificate. This is **mandatory**.
- `bootstrap-server-crl-file`: File location for the bootstrap server's certificate revocation list (CRL).

LwM2M configuration

When `used-bootstrap` is set to false, you can use this configuration:

- `lwm2m-server-uri`: The LwM2M server's URL. For example, `coaps://connector.arm.com`. This is **mandatory**.
- `lwm2m-server-ca-certificate-file`: File location for the LwM2M server's CA certificate. This is **mandatory**.
- `lwm2m-server-crl-file`: File location for the LwM2M server's certificate revocation list (CRL).

These values are pre-configured, no need to change

Confidential © Arm 2017 Limited
arm

Yaml Configuration - continue

Firmware integrity (for firmware update)

- `firmware-integrity-ca-certificate-file`: File location for firmware integrity CA certificate.
- `firmware-integrity-certificate-file`: File location for firmware integrity certificate.
- `firmware-integrity-crl-file`: File location for firmware integrity CRL.

- **These certificates are external input to FCU. They can be generated either by manifest tool or by OEM certificate generator.**
- **More details in Mbed cloud client Linux update session**

Example:

```
# File location for firmware integrity CA certificate
firmware-integrity-ca-certificate-file: <%= ENV['FCU_RESOURCES_DIR'] %>/firmware-integrity-ca-certificate.crt
# File location for firmware integrity certificate
firmware-integrity-certificate-file: <%= ENV['FCU_RESOURCES_DIR'] %>/firmware-integrity-certificate.crt
# File location for firmware integrity CRL
firmware-integrity-crl-file: <%= ENV['FCU_RESOURCES_DIR'] %>/firmware-integrity-crl.pem
```


Confidential © Arm 2017 Limited
arm

Yaml Configuration - continue

LwM2M Device Object information

These parameters are aggregated under the **device-info** YAML section:

- `manufacturer-name`: Manufacturer name (string). This is **mandatory**.
- `device-type`: Device type (string) This is **mandatory**.
- `model-number`: Model number (string). This is **mandatory**.
- `hardware-version`: Hardware version (string). This is **mandatory**.
- `memory-total`: Total memory size in kilobytes (integer). This is **mandatory**.
- `timezone`: Device timezone. Default: UTC.

D2: Setting Device RTC

A: Use FCU

`timezone` should hold value

B: Do it Yourself (DIY)

`timezone` should be empty

Example:

```
device-info:
  # Manufacturer name (String)
  manufacturer-name: SomeManufacturer
  # Device type (String)
  device-type: Sensor-A
  # Model Number (String)
  model-number: TEMP-SENSOR-MVMF7IF
  # Hardware version (String)
  hardware-version: 1A
  # Memory total size in kilobytes.
(Integer)
  memory-total: 10240
  # Device timezone
  timezone: GMT
```


Confidential © Arm 2017 Limited

Yaml Configuration - continue

Configuration for generation of device certificates

These parameters make up the subject of the generated device certificate. They are aggregated under the **device-certificate** YAML section:

- `organization-name`: The **O** subject field.
- `organizational-unit-name`: The **OU** subject field.
- `locality-name`: The **L** subject field.
- `state-or-province-name`: The **ST** subject field.
- `country-name`: The **C** subject field.

Example:

```
device-certificate:
  organization-name: Sample Org Ltd.
  organizational-unit-name: IoT-Temp-Sensors
  locality-name: Texas City
  state-or-province-name: Texas
  country-name: US
```

Trust Level

- **Used to note specific security properties of the device. Simply use default values, for the current product.**

Confidential © Arm 2017 Limited
arm

Setup

- The basic mode of operation when using FCU python package is to create an instance of the `fcu.factoryToolAPI` class
  - Initialize an instance of the FactoryToolApi class by calling its constructor.

Parameters

The `FactoryToolApi` constructor accepts and initializes the following parameters:

Parameter	Optional or mandatory	Description
`home_dir`	Optional	Specifies the home directory of the factory configurator library. The library uses this as the root path of several directories and files it depends on. If not provided, the environment variable `FCU_HOME_DIR` will be used if it is set. Otherwise, *the current working directory* will be used.
`resources_dir`	Optional	Specifies the resources directory of the factory configurator library. If not provided, the environment variable `FCU_RESOURCES_DIR` will be used if it is set. Otherwise, the `/<FCU Home>/resources` folder will be used.
`keystore_dir`	Optional	Specifies the keystore directory of the factory configurator library. If not provided, the environment variable `FCU_KEYSTORE_DIR` will be used if it is set. Otherwise, the `/<FCU Home>/keystore` folder will be used.
`config_file`	Optional	Configuration file name for FCU (including path). If not provided, `<FCU HOME>/config/fcu.yml` is assumed.

Confidential © Arm 2017 Limited

Setup - continue

- **Setup API** (FT to call FCU API)
  - This FCU API should be called by FT (Factory Tool):
    - `FactoryToolApi.setup_factory_configurator_utility`
  - The API performs the initial setup of FCU as a certificate authority (CA). It creates a keystore folder under <FCU HOME> and creates in that folder:
    - A private key named `fcu_key.pem`.
    - A self-signed x509 certificate named `fcu.crt`
    - Properties taken from the .yml configuration file, as described in FCU as CA, make up the subject of the x509 certificate.
  - The certificate must be uploaded to the portal.
    - https://cloud.mbed.com/docs/v1.2/mbed-cloud-deploy/instructions-for-factory-setup-and-device-provision.html#uploading-the-certificate-using-the-apis
  - The API returns `setup_status` object which includes status and errors and warnings lists
  - `FactoryToolApi.get_setup_status` API can be called to verify the setup status

Confidential © Arm 2017 Limited
arm

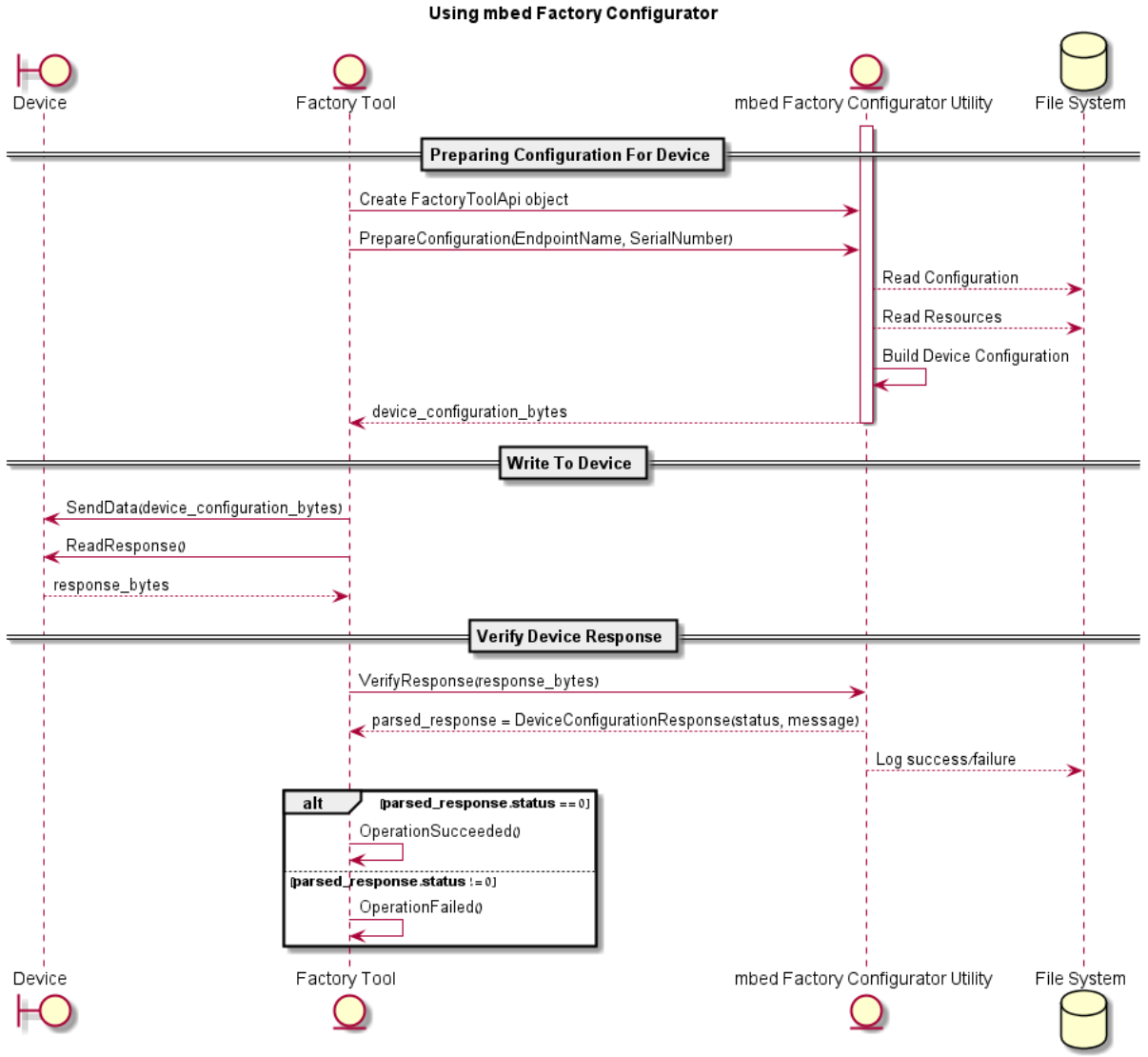
Setup - continue

- Setup API example

```
import fcu

factory_tool_api = fcu.FactoryToolApi()
setup_status = factory_tool_api.setup_factory_configurator_utility(override_existing_ca=False)
```


Confidential © Arm 2017 Limited

Inject – UML diagram



Confidential © Arm 2017 Limited

Inject

The injects flow includes three major substeps:

1. **Preparing configuration for device**– (FT to call FCU API)

   - This FCU API should be called by FT:
     - `FactoryToolApi.prepare_device_configuration`
     - The API receives following parameters:
       - `endpoint_name` (str): Device's endpoint name.
       - `serial_number` (str): Device's serial number.
       - `device_keys_location` (str): optional
     - The API generates device configuration bundle in a CBOR format
     - Most of the information is collected from the configuration file and from input parameters "as is". DTLS keys are generated by FCU.
     - This API returns `DeviceConfigurationRequest` object. It lists the configuration data to be sent to device:
       - `config_data` (bytes): Returns the serialized CBOR data to be sent to device.
       - `warning_list` (list): Returns validation notice list in warning level

D4: Source of device keys and certificates?

A: Use FCU
`device_keys_location` ignored
B:"Bring your own"
`device_keys_location` specifies the location of the device's DTLS keys and certificates

Confidential © Arm 2017 Limited
arm

Inject - continue

2. **Write to device** (FT to Device, FCU not involved)
   - FT sends `config_data` bytes (in CBOR format) to the device (injection)
     - FT must send data size too
   - The device returns `device_response` bytes (in CBOR format) to FT.
3. **Verify device response** – (FT to call FCU API)
   - This FCU API should be called by FT:
     - `FactoryToolApi.verify_device_response`
       - This API receives the device's response (a CBOR formatted blob) after injection, processes and verifies it and returns the status.
         - Parameters
           - `endpoint_name(str)`: The device's endpoint.
           - `device_response(bytes)`: The CBOR blob received from the device in response to injection.
     - `DeviceResponseStatus` object returned to FT.

Confidential © Arm 2017 Limited
arm

Inject - continue

- `DeviceResponseStatus` object returns the following parameters:

  `status (int):` The status code retrieved from device. Non-zero value indicates failure.

  `message (str):` The info message returned from the device.

  `errors (list<ErrorInfo>):` Returns a list of ErrorInfo objects if any error has occurred.

  `warnings (list<WarningInfo>):` Returns a list of WarningInfo objects if any warning has occurred.

- **Errors** generally cause the provisioning flow to fail and the failure reason will be elaborated as part of the object
- **Warnings** are generally notifications to the user. Some warnings may occur upon every injection.

Confidential © Arm 2017 Limited

Notes

- The **YAML Configuration** step is generally intended to be executed once per account
  - The configuration is common to same devices under the same account
- The **Setup** step is generally intended to be executed once per factory machinery
  - For each machinery separate certificate should be uploaded
- The **Inject** step (with its substeps) should be executed once for each device.
- The **Inject** substeps can be separated, so each one of them can be performed on a group of devices (e.g. “**Preparing configuration for device”** substep for all devices, than perform **”Write to device”** substep for all devices and so on)
- Full documentation for the FT-FCU integration can be found here:

  https://cloud.mbed.com/docs/v1.2/mbed-cloud-deploy/instructions-for-factory-setup-and-device-provision.html#when-and-how-to-install-the-factory-configurator-utility

Confidential © Arm 2017 Limited
arm

Error Codes

- Below is a list of FCU error codes that are returned to FT:

100 General error: <ERROR_TYPE>
101 Device <serial number/endpoint name> is mandatory
102 Failed to get <DTLS_KEY_NAME> DTLS private key
103 Failed to get <DTLS_KEY_NAME> DTLS certificate
104 Failed to get <DTLS_KEY_NAME> DTLS private key and certificate
110 Device protocol <PROTOCOL_VERSION> does not match tool protocol <PROTOCOL_VERSION>
180 Device response does not contain the <FIELD_NAME> field
190 Error loading configuration file: <FILE_NAME>
200 Invalid configuration file <FILE_NAME>. The file must be in the YAML format
210 CA certificate file <FILE_NAME> not found. Please run setup and try again
211 Failed to load CA certificate file <FILE_NAME>. The file may be corrupted or in wrong format. Please run setup and try again
220 CA key file <FILE_NAME> not found. Please run setup and try again
221 Failed to load CA private key file <FILE_NAME>. The file may be corrupted or in wrong format. Please run setup and try again
222 The CA certificate file <FILE_NAME> does not match the CA private key <FILE_NAME>. Please run setup and try again

Confidential © Arm 2017 Limited

Error Codes – cont.

230 Invalid signature for certificate signing request
240 Failed to load private key from file <FILE_NAME>
245 Failed to load certificate from file <FILE_NAME>
250 Failed to create private key for <DTLS_KEY_NAME>
260 Failed to create certificate signing request for <DTLS_KEY_NAME>
270 Failed to create certificate for <DTLS_KEY_NAME>
290 File <FILE_NAME> already exists
320 Failed to create certificate
330 Failed to create private key
360 Could not load file: <FILE_NAME>
380 Validation error in field: <FILE_NAME>. Message: <VALIDATION_MESSAGE>
390 <FILE_NAME> is not an X.509 certificate or not formatted in Pem or Der

- More information can be found here:

  https://cloud.mbed.com/docs/v1.2/mbed-cloud-deploy/instructions-for-factory-setup-and-device-provision.html#factory-configurator-utility-error-codes

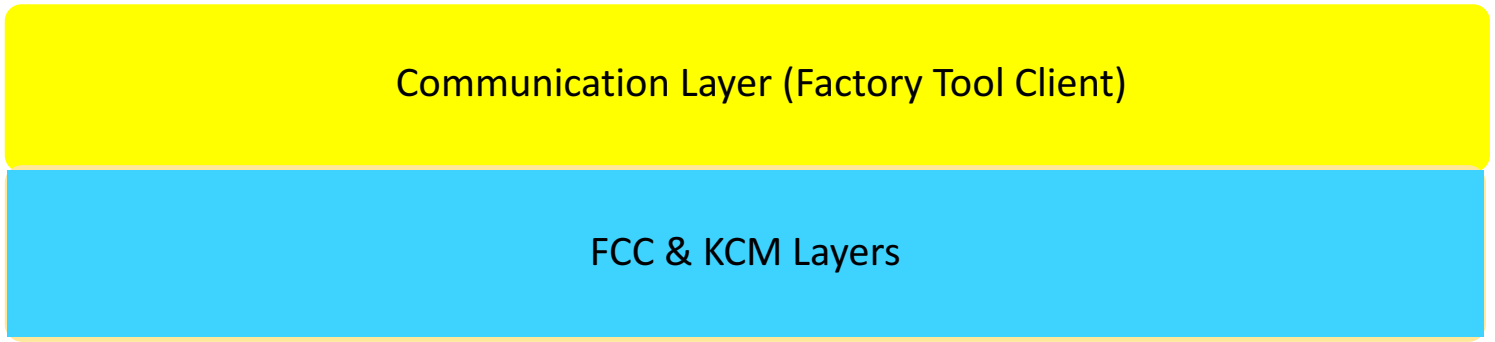
Confidential © Arm 2017 Limited
arm

FTC-FCC (Device Side) Integration

Confidential © Arm 2017 Limited

arm

FCC high level design



Legend

Transport Layer –
Customer component

FCC & KCM Core –
Mbed Product Component

Confidential © Arm 2017 Limited

Process overview

- Initialize FCC.
- Use FCC to process the bundle that was sent from FT to FTC.
- Finalize the FCC.

Confidential © Arm 2017 Limited

FCC initialization and finalization

- Initialization
  - When first used, FCC needs to be initialized with the `fcc_init()` function

```
#include "mbed_factory_configurator_client.h"
#include "fcc_status.h"

int main() {
    ...
    fcc_status_e fcc_status;

    fcc_status = fcc_init();
    if(fcc_status != FCC_STATUS_SUCCESS) {
        return 1;
    }
    ...
}
```


Confidential © Arm 2017 Limited

Processing the FCU bundle

- Single FCC API should be called by the Factory Tool Client:

```
fcc_status_e fcc_bundle_handler (  const uint8_t *   encoded_bundle,
                                   size_t            encoded_bundle_size,
                                   uint8_t **        bundle_response_out,
                                   size_t *          bundle_response_size_out
                                )
```

Decodes and processes an inbound device configuration bundle created by FCU. Also creates an outbound bundle that should be sent to FCU. The function assumes that the bundle includes four groups represented as cbor maps. The names of the groups are `SchemeVersion`, `Keys`, `Certificates` and `ConfigParams`. Each group contains a list of items, and for each item, there are a number of relevant parameters.

Parameters

encoded_bundle	The encoded FCU bundle that is written into a secure storage.
encoded_blob_size	The encoded FCU bundle size in bytes.
bundle_response_out	The encoded outbound bundle. It may contain data such as CSR and different types of key schemes. The response associates a descriptive error in case of a fault.
bundle_response_size_out	The encoded outbound bundle size in bytes.

Returns

FCC_STATUS_SUCCESS in case of success or one of the `::fcc_status_e` errors otherwise.

Confidential © Arm 2017 Limited
arm

Processing the FCU Bundle – cont.

- The API receives the encoded bundle, parses it, verifies its content, saves the data in the device's secure storage and creates an encoded response bundle.
- The `encoded_bundle` parameter is the CBOR byte array sent by the FT (`config_data` from "Interaction with the device" substep in the FCU integration section)
- The `bundle_response_out` parameter is the CBOR byte array that is sent to FT (`device_response` bytes from "Interaction with the device" substep in the FCU integration section)
  - The response bundle contains `status`, `errors` and `warnings` lists that are returned by FCU in `DeviceResponseStatus` object

Confidential © Arm 2017 Limited

Processing the FCU Bundle – cont.

- The function allocates the response bundle; you are responsible to free the response bundle using the free() function, as illustrated in the example code.

```
#include "fcc_bundle_handler.h"

uint8_t fcu_blob[2000] = {...}; // Buffer with FCU blob
uint8_t *response_blob = NULL;
uint32_t response_blob_size;

// Storing bootstrap configuration parameter
fcc_status = fcc_bundle_handler(fcu_blob,
                                sizeof(fcu_blob),
                                &response_blob,
                                &response_blob_size);
// Check whether the function succeeded and whether the response blob was allocated
if(fcc_status != FCC_STATUS_SUCCESS || response_blob == NULL || response_blob_size == 0) {
    // Free the response blob before exiting the function, if allocation was successful
    free(response_blob)
    return 1;
}
```


Confidential © Arm 2017 Limited

Finalization

- Finalization is required when workflow is concluded. `fcc_finalize()` should be used

```
    fcc_status = fcc_finalize();
    if(fcc_status != FCC_STATUS_SUCCESS) {
        return 1;
    }
    ...
}
```


Confidential © Arm 2017 Limited

Error codes

- Below is a list of FCU error codes that are returned by FCC to FTC:

FCC_STATUS_SUCCESS	Operation completed successfully.
FCC_STATUS_ERROR	Operation ended with an unspecified error.
FCC_STATUS_MEMORY_OUT	An out-of-memory condition occurred.
FCC_STATUS_INVALID_PARAMETER	A parameter provided to the function was invalid.
FCC_STATUS_ENTROPY_ERROR	Entropy wasn't initialized correct.
FCC_STATUS_INVALID_CERTIFICATE	Invalid certificate found.
FCC_STATUS_INVALID_CERT_ATTRIBUTE	Operation failed to get an attribute.
FCC_STATUS_INVALID_CA_CERT_SIGNATURE	Invalid ca signature.
FCC_STATUS_EXPIRED_CERTIFICATE	Certificate is expired.
FCC_STATUS_INVALID_LWM2M_CN_ATTR	Invalid CN field of certificate.
FCC_STATUS_KCM_ERROR	KCM basic functionality failed.
FCC_STATUS_KCM_STORAGE_ERROR	KCM failed to read, write or get size of item from/to storage.
FCC_STATUS_KCM_FILE_EXIST_ERROR	KCM tried to create existing storage item.
FCC_STATUS_KCM_CRYPTO_ERROR	KCM returned error upon cryptographic check of an certificate or key.
FCC_STATUS_BUNDLE_ERROR	Protocol layer general error.
FCC_STATUS_BUNDLE_RESPONSE_ERROR	Protocol layer failed to create response buffer.
FCC_STATUS_BUNDLE_UNSUPPORTED_GROUP	Protocol layer detected unsupported group was found in a message.
FCC_STATUS_BUNDLE_INVALID_GROUP	Protocol layer detected invalid group in a message.
FCC_STATUS_BUNDLE_INVALID_SCHEME	The scheme version of a message in the protocol layer is wrong.

Confidential © Arm 2017 Limited
arm

Error codes - cont.

FCC_STATUS_ITEM_NOT_EXIST	Current item wasn't found in the storage.
FCC_STATUS_EMPTY_ITEM	Current item's size is 0.
FCC_STATUS_WRONG_ITEM_DATA_SIZE	Current item's size is different then expected.
FCC_STATUS_URI_WRONG_FORMAT	Current URI is different than expected.
FCC_STATUS_BOOTSTRAP_MODE_ERROR	Wrong value of bootstrapUse mode.
FCC_STATUS_OUTPUT_INFO_ERROR	The process failed in output info creation.
FCC_STATUS_WARNING_CREATE_ERROR	The process failed in output info creation.
FCC_STATUS_UTC_OFFSET_WRONG_FORMAT	Current UTC is wrong.

- More information can be found here:
  - https://cloud.mbed.com/docs/v1.2/factory-client/fcc__status_8h.html#abc705e29420ffbd4e6e2505f198a1025

Confidential © Arm 2017 Limited
arm

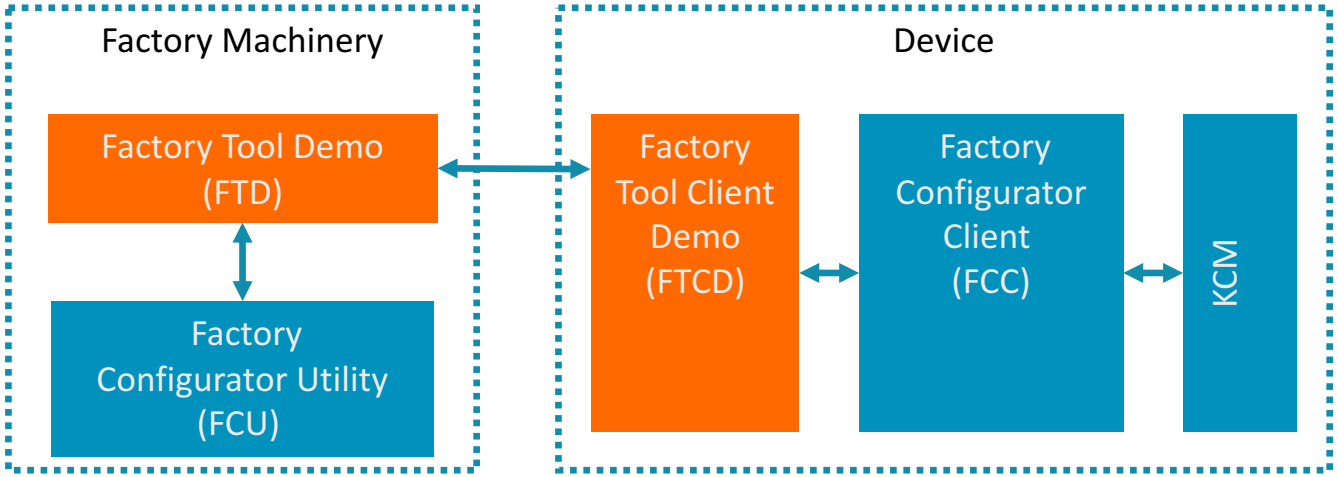
FT Demo

Confidential © Arm 2017 Limited

arm

Introduction

- Mbed Cloud provides components that integrate with your own factory tool. However, to allow you to try the FCC and FCU, we provide a demo factory tool as a reference. It contains a subset of the functionality you might expect a factory tool to have:
  - **Factory tool demo (FTD):** A demo factory tool running on your computer (Python program). It uses the FCU to generate a device configuration bundle, then sends it to the device over either Ethernet or the serial port.
  - **Factory tool client demo (FTCD):** The FTD's counterpart. A C library that runs on the device and processes the configuration bundle it received from the FTD.
- The FTD integrates with FCU in a similar manner your FT should integrate with FCU and FTCD integrates with FCC in a similar manner your FTC should integrate with FCC
  - Similar integration stages are applied



Confidential © Arm 2017 Limited

FTD - The setup command

- Integrates with the setup of FCU. It sets up the FCU as a certificate authority, so it creates a private key and matching certificate (`fcu_key.pem` and `fcu.crt`)
  - The mechanism is to wrap the `fcu.FactoryToolApi` object's `setup_factory_configurator_utility` method
  - Structure:
    - `py -3 ft_demo.py [GLOBAL_OPTIONS] setup [SETUP_OPTIONS] [<status>]`

```
ft_demo [GLOBAL_OPTIONS] setup [SETUP_OPTIONS] [<status>]
```

The available options are:

Options	Required or optional	Explanation
`-f`, `--force`	Optional	If you have already configured FCU as your factory tool's certificate authority, use this option to override the CA key and certificate.

Confidential © Arm 2017 Limited

FTD - The Inject command

- The inject command demonstrates the factory line scenario where the FCU aggregates device configuration, then the factory tool injects it to the device and verifies the device result.
- The command uses the `fcu.FactoryToolApi` object's `prepare_device_configuration` method to configure the device, and the `verify_device_response` method to check whether the configuration was properly applied on the device.

Confidential © Arm 2017 Limited

FTD - The Inject command – cont.

- Structure:
  - `py -3 ft_demo.py [GLOBAL_OPTIONS] inject [CONFIGURATOR_OPTIONS] TRANSPORT [TRANSPORT_OPTIONS]`

The configurator options

Options to pass to the FCU. The available options are:

Options	Required or optional	Explanation
`--endpoint-name=\[ENDPOINT_NAME]`	Required	`ENDPOINT_NAME`: the name of the device endpoint. This parameter is later used for accessing the device.
`--serial-number=\[SERIAL_NUMBER]`	Required	`SERIAL_NUMBER`: the device's serial number. This parameter is used to generate the LwM2M device object.

Confidential © Arm 2017 Limited

FTD - The Inject command – cont.

- The inject command has a TRANSPORT argument that determines the transport method with which the FTD will inject the data to the device.
- The available options are :**TCP, Serial** and **to_file.**
  - Example for injecting over TCP connection:

To send inject data over a TCP connection, use `tcp` with the following options:

Options	Required or optional	Explanation
`--ip=\[IP_ADDRESS]`	Required	`IP_ADDRESS`: the IP address for TCP communication.
`--port=\[PORT_NUMBER]`	Required	`PORT_NUMBER`: the TCP port number to send to. Default: 7777.
`-t`, `--connection-timeout=\[TIMEOUT]`	Required	TCP connection timeout in seconds. Default: 30.

Example:

```
$ python ft_demo/sources/ft_demo.py
    -vv
    --log-file=/tmp/log/tmp-execution.log
    --config-file=/local/arm/mbed/factory-configurator/config.yml
    inject
    --serial-number=SOME-OTHER-SN
    --endpoint-name=EP_some_other_device_endpoint
    --device-keys-location=/local/arm/mbed/factory-configurator/per-device-resources/ip-159
    tcp --ip=10.10.10.159
```


Confidential © Arm 2017 Limited

FTCD

- FTCD is a part of **factory-configurator-client-example-wise-3610** image that can be downloaded from: https://github.com/ARMmbed/factory-configurator-client-example-wise-3610
- FTCD is launched after device reset and waits for incoming packet from FT/FTD and the following prints could be observed in the stdout console:

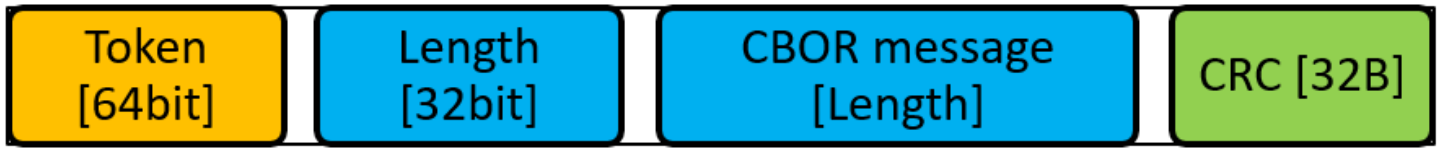
```
[0]factory_configurator_client.c:103:fcc_storage_delete: ===>
[0]factory_configurator_client.c:110:fcc_storage_delete: <===

Factory Client IP Address and Port :  192.168.127.129:7777


Factory Client is waiting for incoming connection...
Factory flow begins...
```

- After FT/FTD sends the packet to the device, FTCD processes the packet and automatically calls the `fcc_bundle_handler API`.
- FTCD processes the response bundle and sends it back to the FT/FTD

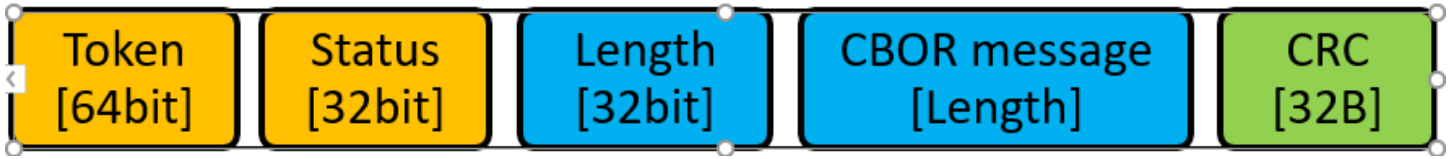
Confidential © Arm 2017 Limited
arm

FTD to FTCD transport message format

- **FTD to FTCD Message Format**



- **FTCD to FTD Message Format**



TOKEN (64 bit) – Unique number agreed by the Factory Tool and Factory Client.
STATUS (32 bit) – The Status of the Factory Tool to Factory Client message.
LENGTH (32 bit) – The message length in bytes.
MESSAGE (LENGTH bytes) – The message raw bytes (encoded bundle) in CBOR format
MESSAGE CRC (32B) – hash SHA256 of the CBOR message

Confidential © Arm 2017 Limited
arm

Error codes for the demo

Error code	Message
1000	General error on factory tool: <ERROR_TYPE>
1001	Communication error received from device
1002	Device response verification failed due to an invalid checksum
1003	Timeout while waiting for device token
1004	Execution failed on device with error #<DEVICE_ERROR_CODE>
1005	Setup failed unexpectedly
1011	Failed to connect to serial port <COM_PORT>
1012	Failed to read data from serial connection
1013	No serial port is available on system
1014	Serial port <COM_PORT> not connected. Available ports: <AVAILABLE_COM_PORTS>
1021	Error connecting to device on IP <IP_ADDRESS> port <TCP_PORT>
1022	Failed to receive data from TCP connection
1023	Failed to send data over TCP connection

Confidential © Arm 2017 Limited

Logging

- During injection, the device will output logs, that look similar to this example:

example_log_device.txt

- FTD outputs the final status, like in the following example:

```
{"status": 0, "message": "Command executed successfully", "deviceResponse": {"status": 0, "message": "The Factory proces
s succeeded", "warnings": ["The parameter is missing:mbed.FirmwareIntegrityCACert", "The parameter is missing:mbed.Firmw
areIntegrityCert"]}}
```

- If **FTD** doesn't print errors (as above), than the provisioning process completed successfully. (Warnings are expected)

Confidential © Arm 2017 Limited

Notes

- FT Demo serves as a reference only. It runs as a single process, communicates with single device and terminates.
- When using FT Demo, inject substeps can not be separated (e.g. all 3 substeps are performed sequentially on a single device, after the inject command is sent).
- It is possible to use `virtualenv` to isolate the python environment
- Full documentation can be found here: https://cloud.mbed.com/docs/v1.2/mbed-cloud-deploy/the-factory-tool-demo-full-reference.html
- Reference to E2E tutorial of the demo: https://cloud.mbed.com/docs/v1.2/mbed-cloud-tutorials/the-factory-tool-demo.html

Confidential © Arm 2017 Limited

Operational Aspects

Confidential © Arm 2017 Limited

arm

FCU Private Key

- The private key used by FCU to sign device certificates is sensitive – if stolen, it can be used to generate fake devices - it is recommended to limit access to private key.
- If the private key is lost, it would be needed to generate a new one and upload the public key to the cloud. It is recommended to keep a backup.
  - Devices that were already manufactured would continue working, as long as the old public key is not removed from the cloud.

Confidential © Arm 2017 Limited

What’s Next?

Confidential © Arm 2017 Limited

arm

Next steps

- Advantech should –
  - Integrate FCU with your factory tool.
    - Decide how to allocate serial numbers, and implement.
    - Decide how to allocate endpoint names, and implement.
  - Integrate FCC with your factory tool on the client.
    - Decide how it would be activated, and implement.
  - Implement transport layer between FCU and FCC (possibly using ft-demo as reference for Ethernet).

Confidential © Arm 2017 Limited

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
תודה

arm

Confidential © Arm 2017 Limited