Pepper x Azure
Cloud Robotics APIハンズオンワークショップ
～Microsoft Azureを簡単に使いこなそう～

2017/2/27 ver 1.0

アジェンダ

	顔認識API	写真説明API
1　デモ	●	●
2　Cloud Robotics APIの仕組み		
2　Cloud Robotics APIを利用したアプリ開発		
3　【Step.1】ストレージの情報を取得する	●	●
4　【Step.2】顔を登録する	●	
5　【Step.3】顔を認識する	●	
6　【Step.4】写真を分析する		●

本日のハンズオンのスコープ

顔認識APIと写真説明APIをハンズオンの対象とします。

Cloud Robotics API の提供

Azure **が無くても、**Azure **を知らなくても、**Choregraphe **さえあれば、容易に接続！**
API **利用の為のコードは全て同じ手順、通信フォーマットが変わるだけ**

翻訳 API

「この商品について、詳しく教えて欲しいのですが」

↕

「想要知道更多關於這種產品。」

顔認識 API

「眼鏡お似合いですね。少々無精ひげも、お疲れですか？」

「サティアさん、あなたは、５回目の来日ですね」

メイン

時間があれば

写真説明 API

「写真のためにポーズをとる人々のグループのようです」

「屋外に、５人、それぞれの性別と年齢は、.....です」

Device to Device API

「吉田様、ルーム #10をお使いください。今、部屋の鍵をお開けしました」

API コール結果の処理を実行する。

会話理解 API (デモ版)

「ホテルを予約されたいのですね。それでは、予約に必要な事をお尋ねします」

「お探しのものは、２Ｆにあります。フロア地図を表示いたします」

デモ

サンプルアプリのデモ

① 顔認識

右手：Pepperがカメラで撮った顔写真を登録（※名前は「田中」）

左手：Pepperがカメラで顔認識

② 写真説明

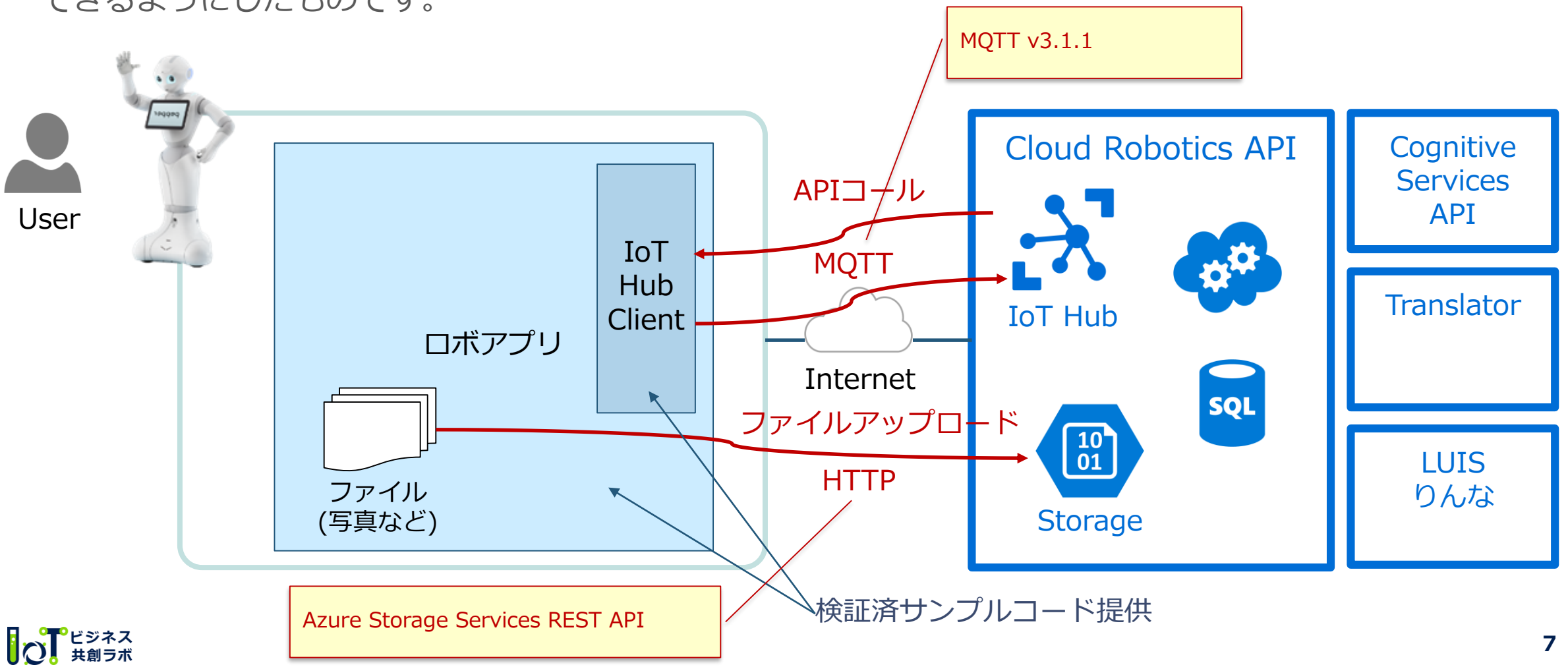
右手：4枚の写真をランダムに説明

左手：Pepperがカメラで撮った写真を説明

Cloud Robotics APIの仕組み

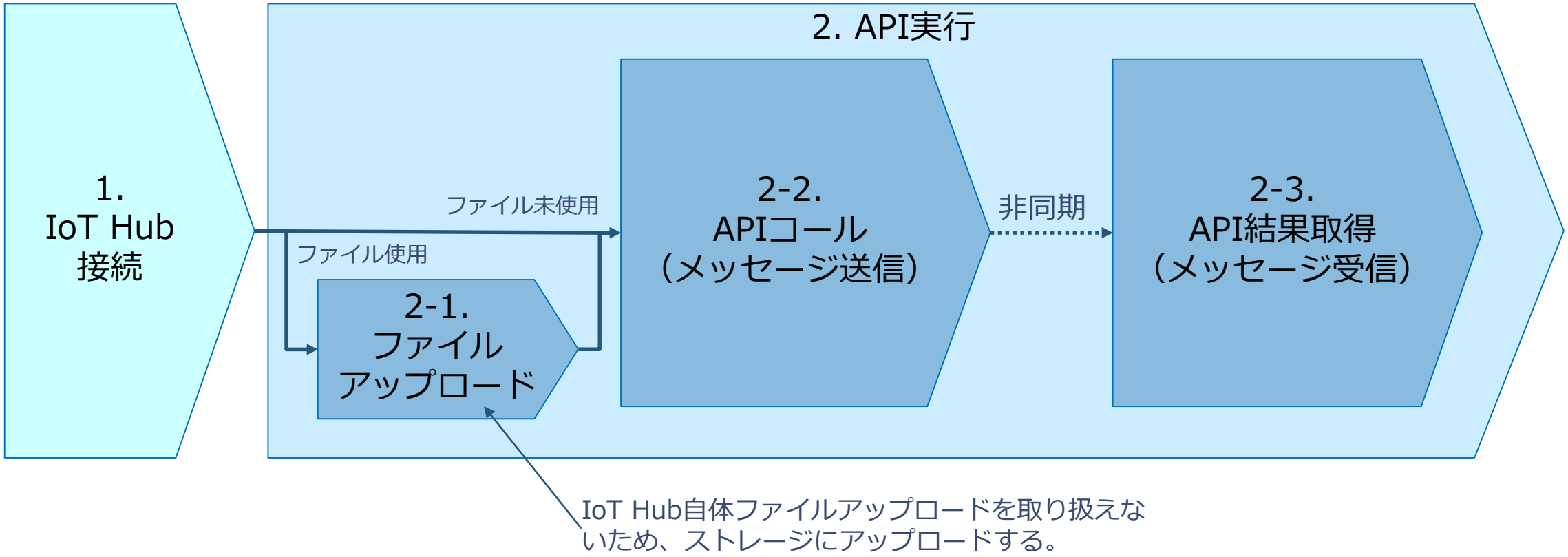
Cloud Robotics API

Cloud Robotics APIは、MS社提供の各種APIやサービスをラップし、共通の通信フォーマットで利用できるようにしたものです。



Cloud Robotics APIを使う時の処理フロー

APIを利用すると、以下のような処理フローになります。



Cloud Robotics APIを利用したアプリ開発

検証済コードの利用

Cloud Robotics APIを使った開発を簡易化するためのサンプルコードを利用することができます。

サンプルコード	使用目的	概要	Pythonモジュール・ クラス・関数
API Client	APIをコールするため	IoT HubとMQTTで通信することができる。 APIのメッセージフォーマットに準じた処理が実装できる。	cloudrobotics.client CRFXClient cloudrobotics.message CRFXMessage
Blob Storage Uploader	写真などのファイルをAPIのインプットとして与えるため	Blob Storageにファイルをアップロードすることができる。	cloudrobotics.storage upload_to_storage()

Cloud Robotics APIとの接続

Cloud Robotics APIとの接続は以下のようなコードを書きます。

- API ClientとなるCRFXClientクラスを利用します。Azure IoT Hubのホスト名、デバイスID、デバイスのSharedAccessKeyが必要となります。
- CRFXClientのstart()メソッドで接続を開始します。接続に失敗した場合は5回リトライします。
- CRFXClientにコールバックを指定して、処理を実装します。

```
import cloudrobotics.client as crfx
client = crfx.CRFXClient('<Your Azure IoT Hub Hostname>','<Your Device Id>','<Your Device Key>')
# コールバックの設定
client.on_connect_successful = on_connect_successful
client.on_connect_failed = on_connect_failed
client.on_disconnect = on_disconnect
client.on_message = on_message
client.on_publish = on_publish
# 接続開始
client.start()
```

on_connect_successful	接続成功時
on_connect_failed	接続失敗時
on_disconnect	切断時
on_message	メッセージ受信時
on_publish	メッセージ送信時

Cloud Robotics APIへのメッセージ送信

メッセージ送信は以下のようなコードを書きます。

- CRFXClient、CRFXMessageクラスを利用します。
- CRFXMessageクラスはRbHeader、RbBodyの値をdictionaryで保持します。APIのメッセージ仕様に合わせて値をセットしてください。 ※Cloud Robotics APIの仕様書参照
- CRFXClientのsend_message()メソッドをコールし、メッセージを送信します。
- RoutingKeyword、AppId、MessageSeqno（送信連番）、SendDateTime（送信日時） は設定不要です。
- 戻りのメッセージとの突合せが必要な場合は、MessageSeqnoを使用してください。

```
# メッセージオブジェクトの生成
message = message.CRFXMessage()
message.header['RoutingType'] = 'CALL'
message.header['AppProcessingId'] = 'RbAppFaceApi'
message.header['MessageId'] = 'init'
message.body['...'] = '....'

# メッセージの送信（自動発行された送信連番が返却される）
seqNo = client.send_message(message)
```

Cloud Robotics APIからのメッセージ受信

メッセージ受信は以下のようなコードを書きます。

- CRFXClientクラスを利用します。
- CRFXClientクラスのon_messageにコールバックを指定します。

```
# API Clientにメッセージ受信時のコールバックをセット
client.on_message = on_message

# メッセージ受信時の処理
def on_message(received_message):
  if received_message.header['MessageId'] == 'init': # APIの種類＝MessageIdにより分岐
    account = received_message.body['storageAccount']
    ...
```

Cloud Robotics APIと連携するためのBox

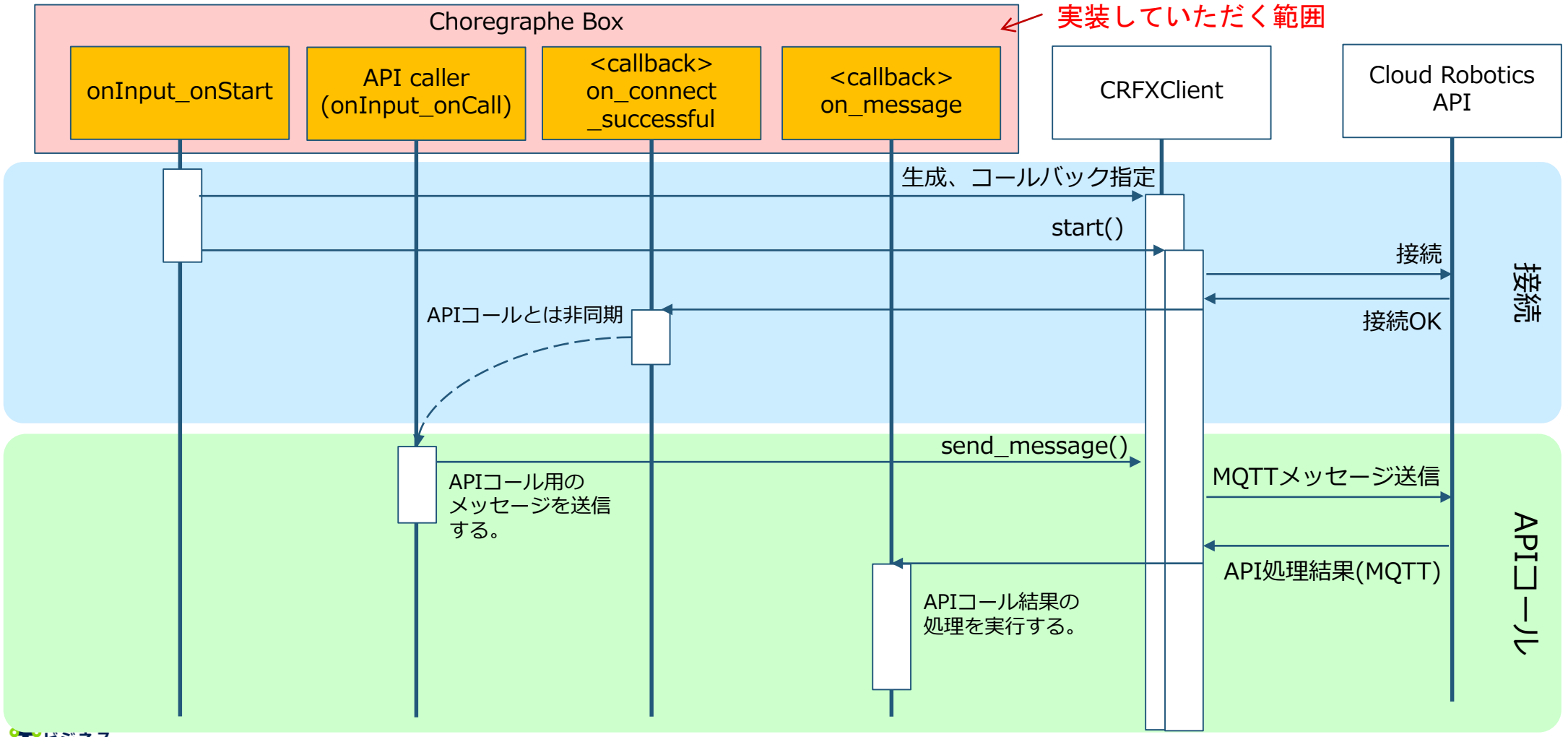
Choregrapheで作成するBoxでは、以下のようなコードを書きます。

```
def onInput_onStart(self): # Boxの開始でAPIクライアントの生成・コールバック設定・接続開始を実施
  import cloudrobotics.client as crfx
  self.client = crfx.CRFXClient(<Your Azure IoT Hub's Hostname>,<Your Device Id>,<Your Device Key>)
  self.client.on_connect_successful = self.on_connect_successful
  self.client.on_message = self.on_message
  self.client.start()

def onInput_onCall(self): # Cloud Robotics APIをコールする
  message = CRFXMessage()
  ...
  self.client.send_message(message)

def on_connect_successful(self): # 接続成功時の処理
  ...

def on_message(self, received_message): # APIコール結果(メッセージ受信時)の処理
  ...
```

API ClientはBox(Class)のインスタンス変数としてください。
ローカル変数とすると、メッセージ受信ができません。
また、使用中はBoxは破棄されないように、
性質がonStoppedの出力を実行しないでください。

Cloud Robotics APIのコールシーケンス

Cloud Robotics APIをコールするためのシーケンス例は以下のとおり。



気を付けるポイント

1. メッセージングなので非同期処理です。
   REST APIのようにAPIコール結果を同期的に取得できません。

2. 顔認識APIについては、顔が傾いていると認識できません。

ハンズオン

ハンズオンを始めるにあたり

1. お手元に Choregrapheがインストールされたpc をご用意ください。
2. Pepperを1台ずつご利用いただきます。
3. API仕様書、ハンズオン用サンプルコード をご準備ください。
4. APIの接続情報をご準備ください。
    1. Azure IoT Hubホスト名　pephackiothub.azure-devices.net
    2. デバイスID
    3. デバイスキー

【Step.1】ストレージの情報を取得する

課題: 1-1

サンプルコードを用いて、APIClientを生成し、Azure IoT Hubに接続して、Pepperに「接続しました」と発話させてください。

```
import time
import os.path

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)
        self.behaviorPath = os.path.normpath(ALFrameManager.getBehaviorPath(self.behaviorId))
        # 写真画像のパス
        self.filepath = os.path.join(os.path.normpath(self.behaviorPath), 'image.jpg')

    def onLoad(self):
        self.client = None

        # TODO:【Step.1】1.API Clientの生成
        self.azure_iot_hub_hostname = '<...>'
        self.azure_iot_hub_device_id = '<...>'
        self.azure_iot_hub_device_key = '<...>'

        self.storage = {}
```

インスタンス変数にIoT Hub接続情報を指定してください。

課題: 1-1

サンプルコードを用いて、APIClientを生成し、Azure IoT Hubに接続して、Pepperに「接続しました」と発話させてください。

```
def onInput_onStart(self):
    import cloudrobotics.client as crfx
    import cloudrobotics.message as message

    # TODO:【Step.1】1.API Clientの生成
    self.client = '<...>'
    self.client.on_connect_successful = self.on_connect_successful
    self.client.on_connect_failed = self.on_connect_failed
    self.client.on_disconnect = self.on_disconnect
    self.client.on_message = self.on_message
    self.client.on_publish = self.on_publish

    # 処理の開始
    self.client.start()
```

IoT Hub接続情報を使い、APIClientクラスを生成してください。

```
#
# コールバック
#

# 接続が成功した時
#
def on_connect_successful(self):
    self.logger.info('started.')
    self.onStarted('<...>')
```

発話内容を入力してください。

課題: 1-2

API仕様書の「“init” (ファイル アップロード先のストレージ情報の取得)」のメッセージを送信し、結果のメッセージを受信してストレージ情報を取得し、Choregrapheの情報ログにstorageAccount、storageKey、storageContainerの3つの情報を出力してください。

- ✓メッセージの送信は
  onInput_onStart(self) に
  記述してください。
- ✓メッセージの受信は
  on_message(self, received_message) に
  記述してください。

```
def onInput_onStart(self):
    import cloudrobotics.client as crfx
    import cloudrobotics.message as message

    # TODO:【Step.1】1.API Clientの生成
    self.client = '<...>'
    self.client.on_connect_successful = self.on_connect_successful
    self.client.on_connect_failed = self.on_connect_failed
    self.client.on_disconnect = self.on_disconnect
    self.client.on_message = self.on_message
    self.client.on_publish = self.on_publish

    # 処理の開始
    self.client.start()

    # TODO:【Step.1】2.ストレージ情報の取得
    init_message = message.CRFXMessage()
    init_message.header['RoutingType'] = '<...>'
    init_message.header['MessageId'] = '<...>'
    init_message.header['AppProcessingId'] = '<...>'

    # メッセージの送信(APIコール)
    self.client.send_message(init_message)
```

API仕様書の送信メッセージの定義に従い、値をセットしてください。

課題: 1-2

API仕様書の「“init” (ファイル アップロード先のストレージ情報の取得)」のメッセージを送信し、結果のメッセージを受信してストレージ情報を取得し、Choregrapheの情報ログにstorageAccount、storageKey、storageContainerの3つの情報を出力してください。

```
    def on_message(self, received_message):
        self.logger.info('received.')
        self.logger.info(str(received_message.header) + ', ' + str(received_message.body))

        # メッセージヘッダーのMessageIdに応じて処理を実装
        if received_message.header['MessageId'] == '<...>':
            # TODO:【Step.1】2.ストレージ情報の取得
            self.storage['account'] = received_message.body['<...>']
            self.storage['key'] = received_message.body['<...>']
            self.storage['container'] = received_message.body['<...>']

            self.logger.info('<...>')
            self.logger.info('<...>')
            self.logger.info('<...>')

        elif received_message.header['MessageId'] == '<...>':
            # TODO:【Step.2】2.顔登録
```

API仕様書の受信メッセージの定義に従い、
値をセットしてください。

【Step.2】顔を登録する

課題: 2-1

Step.1で取得したストレージ情報を使い、ストレージに写真をアップロードしてください。

- ✓ 写真の撮影は、サンプルコードの「Take Picture」Boxを使用してください。
  このBoxを使用すると、ビヘイビアファイルと同階層に image.jpg という名前で保存されます。
  ファイルのパスはBoxの self.filepath に格納されています。

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)
        self.behaviorPath = os.path.normpath(ALFrameManager.getBehaviorPath(self.behaviorId))
        # 写真画像のパス
        self.filepath = os.path.join(os.path.normpath(self.behaviorPath), 'image.jpg')
```

- ✓ 写真の撮影は、Pepperのタッチセンサーをトリガーとしてください。

- ✓ 写真のストレージへのアップロードは、以下のモジュールの関数を利用してください。
  cloudrobotics.storage.upload_to_storage(storageAccount, storageKey, storageContainer, filePath)

- ✓ try ～ except で例外が発生しないかを確認してください。

課題: 2-1

Step.1で取得したストレージ情報を使い、ストレージに写真をアップロードしてください。

```
    def onInput_onCallRegister(self):
        # TODO: 【Step.2】 1.ストレージへの写真アップロード
        import cloudrobotics.message as message
        from cloudrobotics.storage import upload_to_storage

        if not os.path.isfile(self.filepath):
            self.logger.warn('There is no picture file. :' + self.filepath)
            return

        try:
            '<...>'
        except Exception as e:
            self.logger.error('Failed to upload this file to the Azure Blob storage, because ' + str(e))
            return
```

ストレージへのアップロード処理を行う関数を記述してください。

課題: 2-2

API仕様書の「“registerFace” (顔特徴量抽出＆登録)」のメッセージを送信し、結果のメッセージを受信して、ご自身の顔登録の成否を判断し、Pepperに「成功しました」「失敗しました」と発話させてください。

- ✓ Step.1の続きにメッセージ送信の処理をコーディングしてください。
- ✓ メッセージの受信は on_message(self, received_message) に記述します。

```
def onInput_CallRegister(self):
        # TODO: 【Step.2】 2.顔登録
        register_message = message.CRFXMessage()
        register_message.header['RoutingType'] = '<...>'
        register_message.header['AppProcessingId'] = '<...>'
        register_message.header['MessageId'] = '<...>'

        register_message.body['visitor'] = '<...>'
        register_message.body['groupId'] = '<...>'
        register_message.body['locationId'] = '<...>'
        register_message.body['visitor_name'] = '<...>'
        register_message.body['visitor_name_kana'] = '<...>'
        register_message.body['blobFileName'] = 'image.jpg' # Blobのファイル名はアップロードしたファイル名となります。パスは不要です。
        register_message.body['deleteFile'] = '<...>'

        self.client.send_message(register_message)
```

API仕様書の送信メッセージの定義に従い、
値をセットしてください。

課題: 2-2

API仕様書の「“registerFace” (顔特徴量抽出＆登録)」のメッセージを送信し、結果のメッセージを受信して、ご自身の顔登録の成否を判断し、Pepperに「成功しました」「失敗しました」と発話させてください。

```
def on_message(self, received_message):
        elif received_message.header['MessageId'] == '<...>':
            # TODO: 【Step.2】 2.顔登録
            if '<...>':
                self.onRegistered('登録に成功しました。')
            else:
                self.onFailRegistration('登録に失敗しました')
```

API仕様書の受信メッセージの定義に従い、値をセットしてください。

【Step.3】顔を認識する

課題: 3-1

Step.2で登録したご自身の顔情報を使い、API仕様書の「“getFaceInfo” (顔属性取得＆顔特定, ロケーション単位の再訪回数の取得)」のメッセージを送信して、Pepperに顔を認識させて「○○さん、こんにちは」と発話させてください。

```
def onInput_onCallRecognize(self):
    # TODO: 【Step.3】 1.顔認識
    import cloudrobotics.message as message
    from cloudrobotics.storage import upload_to_storage

    if not os.path.isfile(self.filepath):
        self.logger.warn('There is no picture file. :' + self.filepath)
        return

    try:
        '<...>'
    except Exception as e:
        self.logger.error(str(e))
        return

    recognize_message = message.CRFXMessage()
    recognize_message.header['RoutingType'] = '<...>'
    recognize_message.header['AppProcessingId'] = '<...>'
    recognize_message.header['MessageId'] = '<...>'

    recognize_message.body['visitor'] = '<...>'
    recognize_message.body['groupId'] = '<...>'
    recognize_message.body['locationId'] = '<...>'
    recognize_message.body['blobFileName'] = 'image.jpg' # Blobのファイル名はアップロードしたファイル名となります。パスは不要です。
    recognize_message.body['deleteFile'] = '<...>'

    self.client.send_message(recognize_message)
```

ストレージへのアップロード処理を行う関数を記述してください。

API仕様書の送信メッセージの定義に従い、値をセットしてください。

課題: 3-1

Step.2で登録したご自身の顔情報を使い、API仕様書の「“getFaceInfo” (顔属性取得＆顔特定, ロケーション単位の再訪回数の取得)」のメッセージを送信して、Pepperに顔を認識させて「○○さん、こんにちは」と発話させてください。

```
def on_message(self, received_message):
        elif received_message.header['MessageId'] == '<...>':
            # TODO: 【Step.3】 1.挨拶
            if '<...>':
                self.onRecognized('<...>')

                # TODO: 【Step.3】 2.その他の接客
                pass
            else:
                self.onFailRecognition('認識に失敗しました。')
```

API仕様書の受信メッセージの定義に従い、
値をセットしてください。

課題: 3-2

getFaceInfoの結果には、年齢、性別、笑顔か？、眼鏡をかけているか？、再訪回数などの情報が取得できます。その情報を使って自由にPepperに接客させてみてください。

```
def on_message(self, received_message):
        elif received_message.header['MessageId'] == '<...>':
            # TODO: 【Step.3】 1.挨拶
            if '<...>':
                self.onRecognized('<...>')

                # TODO: 【Step.3】 2.その他の接客
                pass
            else:
                self.onFailRecognition('認識に失敗しました。')
```

【Step.4】写真を分析する

課題

1. 新しいビヘイビアを作り、Step.1～Step.3のコードを流用して、Pepperのカメラで撮影した写真や、プロジェクトファイル内に配置した画像ファイルを認識して説明できるようにしてください。
   - ✓ このAPIの元となっているCognitive ServicesのAPIは英語での結果が返却されるため、Translatorを使用して日本語化しています。
   - ✓ htmlフォルダ内のpicture1.jpg～picture4.jpgの写真サンプルをご利用いただけます。

〈説明〉
新鮮な果物や野菜の山

〈タグ〉
食品 food 0.997
野菜 vegetable 0.958
果物 fruit 0.87
フレッシュ fresh 0.301
バラエティ variety 0.232
...

〈説明〉
床に敷設された犬のクローズアップ

〈タグ〉
犬 dog 0.999
屋内 indoor 0.994
茶色 brown 0.89
動物 animal 0.877
日焼け tan 0.163
...

sample2/behavior.xarに、サンプルコードを用意しています。
このコードでは、CRFXClientを継承したCRFXPictureRecognitionClientを実装し、利用しています。
CRFXPictureRecognitionClientのコードは、lib/cloudrobotics/picturerecognition/client.py にあります。

Appendix

サンプルコードの説明

ハンズオンで提供されるファイルは以下のとおりです。

Choregrapheプロジェクトファイル群				説明
cloudrobotics api_handson	lib	cloudrobotics		Cloud Robotics APIモジュール
			facerecognition	顔認識API用
			picturerecognition	写真説明API用
		paho		MQTT通信モジュール
	sample1			顔認識APIサンプル用ビヘイビア
	sample2			写真説明APIサンプル用ビヘイビア
	handson1			顔認識API ハンズオン用 課題ビヘイビア
	handson1_ans			顔認識APIハンズオン用 サンプルビヘイビア


IoT
ビジネス
共創ラボ