



API DESIGN AND SPECIFICATION





Device Gateway Agent Services Specification

Version 3.1
Date: 10-Nov-2017



API DESIGN AND SPECIFICATION

Signatures

Names	Role	Signature	Date
Anirudha Gokhale	Wipro (Author)		10-Nov-17
Sanjay Girigoswami	Wipro (Reviewer)		10-Nov-17
Umesh D R	Wipro (Reviewer)		10-Nov-17
Gajanana K Dixit	Wipro (Approver)		10-Nov-2017



Revision History

Version (x.y)	Date of Revision	Description of Change	Reason for Change	Affected Sections
0.1	07-07-2016	All the section	Initial draft	All section
0.2	07-15-2016	All the section	Incorporating FMC comments	All
0.3	07-29-2016	Added GetCurrentUTC API.	Incorporated changed based on review comments	2.1.1
0.4	08-18-2016	Renamed to GetCurrentTime.	To be in sync with the review comments received on for Systech API doc	2.1.1
0.5	09-13-2016	Renaming the APIs to services	Renaming to - Device Gateway Agent Services and Reciprocity Gateway	All
1.0	10-20-2016	All the section	Incorporated review comments of DBT, Mark and Matt	All
1.1	01-27-2017	All the section	Feedback from Matt	All
2.0	02-08-2017	Approval and baselining	Approval and baselining	NA
2.1	16-Mar-2017	Adding new API to Ack Pairing Request and DBT review comments Date format changed, footer added	DBT's requirement to ACK pairing request	1.2, 2.1.1.2.3, added 2.1.1.14, 4.1.1, 2.1.1.5, 2.1.1.6
3.0	27-Mar-2017	Approval and baselining	Approval and baselining	NA
3.1	10-Nov-2017	RXS114 version changed from draft to v1	Reference to outdated reference draft document removed	List of reference document section

Affected Groups

HDM team
Gateway

List of Reference Documents

1. Connected Health Service MCND v1.0 5-25-2016.docx
2. Fresenius Connected Health Cybersecurity Approach 20160628v2.1.docx
3. Reciprocity Gateway & Device Agents User Stories version 1.0
4. Reciprocity Gateway Services Specification version 1.0
5. RXS114_Gateway_Agent_and_Cloud_Agent_URS_v1
6. RCHS_Reciprocity Software Architecture Document_V3.0



Table of Contents

REVISION HISTORY	3
1 INTRODUCTION	5
1.2 OVERVIEW	5
1.2.1 Relation to External Environment	5
2 API DESIGN	7
2.1 API FUNCTIONS	7
2.1.1 Device Agent Client Proxy	7
2.1.1.1 GetAgentProxyVersion – API Function	7
2.1.1.2 Init – API Function	7
2.1.1.3 StartAgentService – API Function	16
2.1.1.4 GetAgentServiceVersion – API Function	17
2.1.1.5 IsHdmPaired – API Function	17
2.1.1.6 SetDeviceDiscoverable – API Function	18
2.1.1.7 DisableDeviceDiscoverable – API Function	18
2.1.1.8 GetCurrentUTC – API Function	18
2.1.1.9 GetBpmMeasurement – API Function	19
2.1.1.10 GetWsMeasurement – API Function	19
2.1.1.11 DelHdmPairing – API Function	20
2.1.1.12 StopAgentService – API Function	20
2.1.1.13 DelInit – API Function	20
2.1.1.14 PairingResponse – API Function	20
2.1.1.15 GetInstance – API Function	21
2.1.1.16 ~CDeviceAgentClientProxy – API Function	22
3 ANNEX – I	23
3.1 CONFIGURATION FILE AND ARGUMENT TO AGENTSERVICE	23
4 ANNEX – II	25
4.1 INTERFACE WITH USER APPLICATION	25
4.1.1 CDeviceAgentClientProxy.h	25
4.1.2 CGatewayServiceCbImpl.h	26
4.2 DELIVERABLES FOR HDM	26

Table of Figures

Figure 1: Device Agent Client Proxy interaction with the user application	5
Figure 2: Sequence of interaction for Pairing HDM from Gateway	21
Figure 3: Agent Proxy – Agent Service -HDM interaction diagram w.r.t. files	27



- GetInstance – to create the object of the DeviceAgentClientProxy
- GetAgentProxyVersion – to get the version of the DeviceAgentClientProxy.
- Init – to Initialize Device Gateway Agent Services.
- StartAgentService – Starts Device Gateway Agent Service.
- GetAgentServiceVersion – to get the version of the Device Gateway Agent Service.
- IsHdmPaired – Check if HDM is paired with Gateway.
- DeleteHdmPairing – delete the pairing of HDM with Gateway.
- SetDeviceDiscoverable – Set HDM's BT in discovery mode.
- DisableDeviceDiscoverable – Stop HDM's BT from discovery mode.
- GetCurrentUTC – gets the UTC time from Reciprocity Gateway.
- GetBpmMeasurement – to receive BPM's Peripheral data from the Reciprocity Gateway.
- GetWsMeasurement – to receive WS's Peripheral data from the Reciprocity Gateway.
- StopAgentService – Stops /kills the Device Gateway Agent Service.
- DelInit – to de-initialize registered callback functions.
- PairingResponse – Acknowledge pairing request of Gateway.

The required APIs can be called in above sequence. Most of the APIs expect the Device Gateway Agent Services to be started. Details are provided in relevant sections.



2 API Design

The following section details out the APIs of the Device Agent Client Proxy class.

2.1 API functions

Please refer Annex – II of this document for the CDeviceAgentClientProxy class definition.

2.1.1 Device Agent Client Proxy

2.1.1.1 GetAgentProxyVersion – API Function

This API returns version of the DeviceAgentClientProxy. User application may want to know the version of the Proxy, even before 'Init' is called. Hence only this API can be called even if DeviceAgentClientProxy is not 'Init'ated.

Signature of Function:

```
std::string GetAgentProxyVersion(void);
```

Return Value:

Type	Description
std::string	<ul style="list-style-type: none">Returns version of the Agent ProxyReturns NULL if error

2.1.1.2 Init – API Function

Init API initializes Device Agent Client Proxy. It will register callback functions. It will create a thread that is responsible to interact with Device Gateway Agent Service and User Application. It will be the only thread in the Device Agent Client Proxy. Once the thread is started, an event "Init Successful" is sent to the User Application.

Signature of Function:

For C++ interface -

```
int32_t Init(CHdmCbImpl *pcHdm,
            void (CHdmCbImpl::*pFnBpm) (CBloodPressureMtr *pBpm),
            void (CHdmCbImpl::*pFnWs) (CWeighingScale *pWs),
            void (CHdmCbImpl::*PFnEvt) (CEvent *pEvt),
            std::string sConfigFilePath);
```

For JSon interface -

```
int32_t Init(CHdmCbImpl *pcHdm,
            void (CHdmCbImpl::*pFnBpm) ( CJson *pBpm),
            void (CHdmCbImpl::*pFnWs) ( CJson *pWs),
            void (CHdmCbImpl::*PFnEvt) ( CJson *pEvt),
            std::string sConfigFilePath);
```

**Input Parameters:**

Parameter Name	Type	Description
pcHdm	CHdmCbImpl	This is the class implemented by HDM having callback functions
pFnBpm	Function Pointer	This is a callback function pointer for sending BPM values to the User Application
pFnWs	Function Pointer	This is a callback function pointer for sending WS values to the User Application
pFnEvent	Function Pointer	This is a callback function pointer for sending Event to the User Application
sConfigFilePath	String	This is the path of configuration file (E.g. "/data/agents/DeviceAgent.conf"). This file will be common to IoT Agent. Please refer: Annex-I of this document for configuration file content.

Return Value:

Type	Description
int	0 – Success -1 – Failed

2.1.1.2.1 pFnBpm

pFnBpm is a function pointer, pointing to the user defined function. Device Agent Client Proxy will call this function for returning BPM data to the user application, asynchronously. The data will be received from Gateway only when a GetBpmMeasurement request is made. Reciprocity Gateway Service will send the BPM measurement reading that was taken recently (e.g. reading taken within past 10 min when the GetBpmMeasurement request was made. This duration will be configured in Gateway).

2.1.1.2.1.1 JSON Output**Signature of Function:**

```
typedef void (CHdmCbImpl::*PtrBpm)( CJson *pBpmMtr);
```

Callback Output Parameters Json format:

Parameter Name	Type	Description
ParamID	Number	Parameter ID – It is ID for the parameter type: 3. Systolic 4. Diastolic 5. Mean 6. Pulse For Each of above ParamID, there will be associated ParamValue and Unit
ParamValue	Number	Parameter value– It is the value sent by the peripheral device



Unit	String	Unit of the parameter: <ul style="list-style-type: none">• “mmHg”• “kPa”• “pulse per min”
Time	String	Time of reading or event in UTC format: “YYYY-MM-DDTHH:MM:SS.SSSZ”
MfgName	String	Manufacturer name – provided device has a capability to send this value (e.g. “A&D Medical”).
ModelNum	String	Model number – provided device has a capability to send this value (e.g. “UA-651BLE”)
SerialNum	String	Serial number – provided device has a capability to send this value (e.g. “5140700001”).
HardwareRev	String	Hardware Revision – provided device has a capability to send this value (e.g. “0.00”).
FirmwareRev	String	Firmware Revision – provided device has a capability to send this value (e.g. “BLP008_d015”).
SystemID	String	System ID – provided device has a capability to send this value (e.g. “xxxxxxFFFExxxxxx” – combination of BT address and some additional number).

Sample Data:

```
class CJson
{
    private:
        string m_sJson;

    public:
        string setJsonString(string);
        string getJsonString();
};
```

Json string will have:

```
{ "Bpm": {
    "Time": "YYYY-MM-DDTHH:MM:SS.SSSZ",
    "MfgName": "Manufacturer Name",
    "ModelNum": "XX-XXXX",
    "SerialNum": "nnnnnnnnn",
    "HardwareRev": "0.00",
    "FirmwareRev": "XXX-RevXX",
    "SystemID": "xxxxxxxxxxxx",
    "Param": [{
        "ParamID": "3",
        "ParamValue": "80.0",
        "Unit": "mmHg"
    }
    {
        "ParamID": "4",
        "ParamValue": "120.0",
```



```
        "Unit": "mmHg"
      }
    {
      "ParamID": "5",
      "ParamValue": "120.0",
      "Unit": "mmHg"
    }
    {
      "ParamID": "5",
      "ParamValue": "120.0",
      "Unit": "pulse per min"
    }
  ]
}
```

Output is shared with the User Application as part of 'pBpmMtr'.

2.1.1.2.1.2 C++ Output

Signature of Function:

```
typedef void (CHdmCbImpl::*PtrBpm)(CBloodPressureMtr *pBpmMtr);
```

Callback Output Parameters C++ format:

Callback will receive CBloodPressureMtr class object.

```
class CBloodPressureMtr: public CPeripheralDevice
{
private:
    std::string m_sTime;
    SFLOAT m_fSysVal;
    uint32_t m_iSysUnit;
    SFLOAT m_fDiaVal;
    uint32_t m_iDiaUnit;
    SFLOAT m_fMapVal;
    uint32_t m_iMapUnit;
    SFLOAT m_fPulseVal;
    uint32_t m_iPulseUnit;
public:
    void SetTime(std::string sTime);
    void SetSysValue(SFLOAT fSysVal);
    void SetSysUnit(uint32_t iSysUnit);
    void SetDiaValue(SFLOAT fDiaVal);
    void SetDiaUnit(uint32_t iSysUnit);
    void SetMapValue(SFLOAT fMapVal);
    void SetMapUnit(uint32_t iMapUnit);
    void SetPulseValue(SFLOAT fPulseVal);
    void SetPulseUnit(uint32_t iPulseUnit);
    std::string GetTime(void);
```



```
SFLOAT GetSysValue(void);
uint32_t GetSysUnit(void);
SFLOAT GetDiaValue(void);
uint32_t GetDiaUnit(void);
SFLOAT GetMapValue(void);
uint32_t GetMapUnit(void);
SFLOAT GetPulseValue(void);
uint32_t GetPulseUnit(void);
};
** SFLOAT is a two Byte value as defined IEEE-11073 16-bit SFLOAT

class CPeripheralDevice
{
private:
    std::string m_sMfgName;
    std::string m_sModelNum;
    std::string m_sSerialNum;
    std::string m_sHardwareRev;
    std::string m_sFirmwareRev;
    std::string m_sSystemID;

public:
    void SetMfgName(std::string sName);
    void SetModelNum(std::string sModel);
    void SetSerialNum(std::string sSerial);
    void SetHardwareRev(std::string sHWRev);
    void SetFirmwareRev(std::string sFWRev);
    void SetSystemID(std::string sSysID);
    std::string GetMfgName(void);
    std::string GetModelNum(void);
    std::string GetSerialNum(void);
    std::string GetHardwareRev(void);
    std::string GetFirmwareRev(void);
    std::string GetSystemID(void);
};
```

2.1.1.2.2 pFnWs

pFnWs is a function pointer, pointing to the user defined function. Device Agent Client Proxy will call this function for returning WS data to the user application, asynchronously. The data will be received from Gateway only when a GetWsMeasurement request is made. Reciprocity Gateway Service will send the WS measurement reading that was taken recently (e.g. reading taken within past 10 min when the GetWsMeasurement request was made. This duration will be configured in Gateway).

2.1.1.2.2.1 JSON Output

Signature of Function:



```
typedef void (CHdmcImpl::*Ptrws)( CJson *pwsMtr);
```

Callback Output Parameters Json format:

Parameter Name	Type	Description
ParamID	Number	Parameter ID – It is ID for the parameter type: 7. Weight
ParamValue	Number	Parameter value– It is the value sent by the peripheral device
Unit	String	Unit of the parameter: <ul style="list-style-type: none">• “Kg”• “lb”
Time	String	Time of reading or event in UTC format: “YYYY-MM-DDTHH:MM:SS.SSSZ”
MfgName	String	Manufacturer name – provided device has a capability to send this value (e.g. “A&D Medical”).
ModelNum	String	Model number – provided device has a capability to send this value (e.g. “UA-651BLE”)
SerialNum	String	Serial number – provided device has a capability to send this value (e.g. “5140700001”).
HardwareRev	String	Hardware Revision – provided device has a capability to send this value (e.g. “0.00”).
FirmwareRev	String	Firmware Revision – provided device has a capability to send this value (e.g. “BLP008_d015”).
SystemID	String	System ID – provided device has a capability to send this value (e.g. “xxxxxxFFFExxxxxx” – combination of BT address and some additional number).

Sample Data:

```
class CJson
{
    private:
        string m_sJson;

    public:
        string setJsonString(string);
        string getJsonString();

};
```

Json string will have:

```
{ "Ws": {
    "Time": "YYYY-MM-DDTHH:MM:SS.SSSZ",
    "Param": {
        "ParamID": "7",
        "ParamValue": "80.0",
        "Unit": "Kg"
```



```

    }
}

```

Output is shared with the User Application as part of 'pwsMtr'.

2.1.1.2.2.2 C++ Output

Signature of Function:

```
typedef void (CHdmcImpl::*PtrWs)(CWeighingScale *pwsMtr);
```

Callback Output Parameters C++ format:

Callback will receive CWeighingScale class object.

```

class CWeighingScale: public CPeripheralDevice
{
private:
    std::string m_sTime;
    SFLOAT m_fWeightVal;
    uint32_t m_iWeightUnit;

public:
    void SetTime(std::string sTime);
    void SetWeightValue(SFLOAT fWtVal);
    void SetWeightUnit(uint32_t iWtUnit);
    std::string GetTime(void);
    SFLOAT GetWeightValue(void);
    uint32_t GetWeightUnit(void);
};

```

2.1.1.2.3 pFnEvent

pcbEvent is a function pointer, pointing to the user defined function. Device Agent Client Proxy will call this function for sending Events to the User Application.

2.1.1.2.3.1 JSON Output

Signature of Function:

```
typedef void (CHdmcImpl::*PtrEvt)( CJson *pEvent);
```

Callback Output Parameters Json format:

Parameter Name	Type	Description
Time	String	Time of reading or event in UTC format:



		"YYYY-MM-DDTHH:MM:SS.SSSZ"
EventID	Number	<p>Event received from the Gateway APIs:</p> <ul style="list-style-type: none"> 1 -> Agent - State 2 -> Agent - Event 3 -> BPM - Event 4 -> WS – Event 5 -> Time Event <p>Each Event will have EventID, EventSubID, and Time.</p>
EventSubID	Number	<p>The below is the list of sub event IDs for the Events:</p> <p>Agent - State</p> <ul style="list-style-type: none"> 1 -> Ready (On Init) 2 -> Running (When GetMeasurment is in Progress) 3 -> Terminated (On Delnit) <p>Agent - Event</p> <ul style="list-style-type: none"> 1-> Init successful 2-> Agent Service Start - Successful 3-> Agent Service Start - Failed 4-> Agent Service Stop - Successful 5-> Agent Service Stop - Failed 6-> BNEP Connection - Successful 7-> BNEP Connection - Failed 8-> Gateway Connection - Successful 9-> Gateway Connection - Failed 10-> HDM Paired now 11-> HDM Delete Pairing - Successful 12-> HDM Delete Pairing - Failed 13-> HDM is paired with Gateway 14-> HDM is not paired with Gateway 15-> HDM Start Discovery - Successful 16-> HDM Start Discovery - Failed 17-> HDM Stop Discovery - Successful 18-> HDM Stop Discovery - Failed 19-> I am alive 20-> Agent Service is not started 21-> Pairing Requested <p>Internal events consumed by Agent</p> <p>Agent – Event</p> <ul style="list-style-type: none"> 22-> HDM Adapter Start – Successful 23-> HDM Adapter Start – Failed 24-> HDM Adapter Stop – Successful 25-> HDM Adapter Stop – Failed 26-> Agent Service PID 27-> Agent Service Version <p>BPM - Event</p> <ul style="list-style-type: none"> 1 -> BPM no measurement available 2 -> BPM Get Measurement request failed 3 -> Body movement during Measurement 4 -> Cuff too loose 5 -> Irregular pulse detected



		6 -> Pulse rate exceeds upper limit 7 -> Pulse rate is less than lower limit 8 -> Improper measurement position 9 -> Low battery (33%) <u>Note:</u> BPM's Event Sub ID 3 to 9 are A&D BPM device specific events. WS - Event 1 -> WS no measurement available 2 -> WS Get Measurement request failed. Time – Event 1-> Current Time 2-> NTP time not synced 3-> Get Current Time Request Failed
EventDesc	String	Can be description.
EventSubIDDesc	String	Description / comments. In case of sub event 'Pairing Requested' (Sub ID: 21), it will have following description: "Device Name": "<BtAddr> i.e. friendly name of device, concatenated by separator ':', concatenated by device's BT Address. E.g.: SL1500-12768D_0::00:17:E9:D1:75:5D. Above string will also be sent in case of sub event 'HDM Paired now' (Sub ID: 10) so that user application can display the friendly name with which HDM has paired. Same is the case of sub event 'HDM is paired with Gateway' (Sub ID: 13).

Sample Data:

```

class CJson
{
    private:
        string m_sJson;

    public:
        string setJsonString(string);
        string getJsonString();
};

```

Json string will have:

```

{ "Event":
  { "Time": "YYYY-MM-DDTHH:MM:SS.SSSZ",
    "EventID": "3",
    "EventSubID": "3"
  }
}

```

Output is sent to the User Application as part of 'pEvent'.



2.1.1.2.3.2 C++ Output

Signature of Function:

```
typedef void (CHdmCbImpl::*PtrEvt)(CEvent *pEvent);
```

Callback will receive CEvent class object.

Callback Output Parameters C++ format:

```
class CEvent
{
private:
    std::string m_sTime;
    int32_t m_iEventID;
    std::string m_iEventDesc;
    int32_t m_iEventSubID;
    std::string m_iEventSubIDDesc;

public:
    void SetTime(std::string sTime);
    void SetEventID(int32_t iEvtID);
    void SetEventSubID(int32_t iEvtSubID);
    void SetEventID(std::string sEvtDesc);
    void SetEventSubID(std::string sEvtSubIDDesc);
    std::string GetTime(void);
    int32_t GetEventID(void);
    int32_t GetEventSubID(void);
    std::string GetEventID(void);
    std::string GetEventSubID(void);
};
```

2.1.1.3 StartAgentService – API Function

This API can be called only if DeviceAgentClientProxy is initiated (i.e. after “Init” function is called). This API `exec GatewayService` executable with runtime parameters, creating demon process. This is the only process of the Gateway Agent. DeviceAgentClientProxy establishes IPC (using Unix Socket ^[5]) with GatewayService. It then gets the process ID of demon process, which will be used for stopping the process. It also gets the version of the Device Gateway Agent Service. So whenever a request is made for Device Gateway Agent Service’ version, DeviceAgentClientProxy returns it to the User Application from the Proxy. The event “Agent Service Start Successful” is returned only after getting the PID and the version of Device Gateway Agent Service. It also establishes BNEP between GatewayService and AgentService.

The Stopped Agent Service can be restarted by calling StartAgentService, provided Dilnit is not yet called.

Signature of Function:

```
int32_t StartAgentService(std::string sCommands);
```


**Input Parameters:**

Parameter Name	Type	Description
sCommands	string	The command will be passed to the Agent Service executable as below: GatewayAgentService --KeepAlive=true --KeepAliveInterval=60 - - ReciprocityGatewayClientCerti="/data/agents/gateway/CliCerti" -- ReciprocityGatewayServCerti="/data/agents/gateway/ServCerti" --ReciprocityGatewayTransport=bluetooth -- ReciprocityGatewayConfig="/data/agents/DeviceAgent.conf"

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 2 - Already started -2 - Init is not yet called -1 - Failed if no parameters passed

2.1.1.4 GetAgentServiceVersion – API Function

This API returns Agent Service version, provided service is 'start'ed. In case if the service is not yet started, then it returns a NULL value.

Signature of Function:

```
std::string GetAgentServiceVersion(void);
```

Return Value:

Type	Description
std::string	<ul style="list-style-type: none">Returns version of the Gateway Agent Service if the service is started.NULL if service is not started.

2.1.1.5 IsHdmPaired– API Function

It checks whether HDM is already paired with Gateway. The status is returned through the event “HDM is paired with Gateway” or “HDM is not paired with Gateway”. This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t IsHdmPaired(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 1 - Already a request is in progress -3 - Agent Service not started yet



2.1.1.6 SetDeviceDiscoverable – API Function

This API sets the BT of HDM in discoverable mode. Once the HDM device is discoverable, pairing can be initiated from Gateway. Either “Start Discovery Successful” or “Start Discovery Failed” event is sent to User Application based on the status.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t SetDeviceDiscoverable(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request -3 - Agent Service not started

2.1.1.7 DisableDeviceDiscoverable – API Function

This API disables the discovery mode of HDM's BT. Either “Stop Discovery Successful” or “Stop Discovery Failed” event is sent to User Application based on the status.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t DisableDeviceDiscoverable(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request -3 - Agent Service not started

2.1.1.8 GetCurrentUTC – API Function

GetCurrentUTC API is responsible to get the current time of Reciprocity Gateway in UTC format. The user application receives the current time through the “Current Time” event. In case if the request cannot be executed due to any network failure, then user application will receive “Get Current Request Failed” event. In case if Gateway's time stamp is older than the manufacturing time, then User Application will receive event “Ntp time not synced”.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
void GetCurrentUTC (void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 1 - Already Get request in progress -3 - Agent Service not started



2.1.1.9 GetBpmMeasurement – API Function

GetBpmMeasurement API is responsible to get BPM measurement reading from Reciprocity Gateway. This API is a non-blocking call. On calling this API:

1. If Reciprocity Gateway already has the data (which was received from peripheral within the configured time, say less than 10 mins ago), Gateway will send this data immediately.
2. If Reciprocity Gateway do not have the data, it will return with no measurement available event.

Note: GetBPMMeasurement API will only request for measurement of a particular vital parameter. The process / logic of connecting and communicating with the Peripheral based on the device specific protocol is handled by the Reciprocity Gateway.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t GetBpmMeasurement(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 1 - Already Get request in progress -3 - Agent Service not started

2.1.1.10 GetWsMeasurement – API Function

GetWSMeasurement API is responsible to get WS measurement reading from the Reciprocity Gateway. This API is a non-blocking call. On calling this API:

1. If Reciprocity Gateway already has the data (which was received from peripheral within the configured time, say less than 10 mins ago), Gateway will send this data immediately.
2. If Reciprocity Gateway do not have the data, it will return with no measurement available event.

Note: GetWSMeasurement API will only request for measurement of a particular vital parameter. The process / logic of connecting and communicating with the Peripheral based on the device specific protocol is handled by the Reciprocity Gateway.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t GetWsMeasurement(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 1 - Already Get request in progress -3 - Agent Service not started



2.1.1.11 DelHdmPairing – API Function

Deletes BT pairing of HDM with Gateway. User application will receive “Delete Pairing Failed” or “Delete Pairing Successful” event based on the status.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t DelHdmPairing(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request 1 - Already a request is in progress -3 - Agent Service not started

2.1.1.12 StopAgentService – API Function

It will stops / kills the Device Gateway Agent Service demon process. User application will receive “Agent Service Stop Successful” or “Agent Service Stop Failed” event based on the status.

This API can be called only if Device Agent Client Proxy was Started (i.e. “StartAgentService” API was called earlier).

Signature of Function:

```
int32_t StopAgentService(void);
```

Return Value:

Type	Description
int32_t	3 - Acknowledge the request -3 - Not yet started

2.1.1.13 Delnit – API Function

Delnit API releases all the resources that are created and used by the Device Agent Client Proxy. Once a Delnit function is called, no other functionality of these APIs can be availed until Init is called again. Exception to this is GetAgentProxyVersion API. Before calling Delnit, the StopAgentService must be called.

Signature of Function:

```
int32_t DeInit(void);
```

Return Value:

Type	Description
int32_t	0 - Success -2 - Not Initiated -4 - Not stopped.

2.1.1.14 PairingResponse – API Function



PairingResponse API will be called to acknowledge the pairing request of Gateway. The pairing request can be either accepted or rejected for the corresponding BT address.

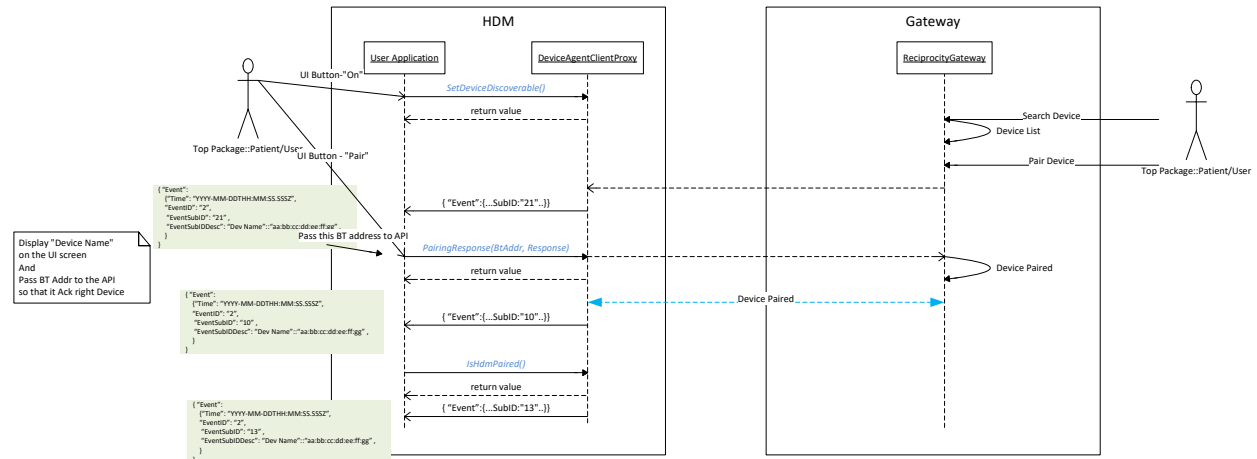


Figure 2: Sequence of interaction for Pairing HDM from Gateway

Above diagram depict API and related event usage. The diagram provides high level sequence only.

Signature of Function:

`int32_t PairingResponse(std::string sBtAddress, int32_t iUserResponse);`

Input Parameters:

Parameter Name	Type	Description
sBtAddress	string	"00:17:E9:D1:75:5D"
iUserResponse	int32_t	1 – Pair -1 – Don't Pair

Return Value:

Type	Description
int32_t	3 – Acknowledge the request -3 – Not yet started

2.1.1.15 GetInstance – API Function

GetInstance API will return pointer to the class object. It will create object if object is not created so far. Using this pointer, user application will call the APIs. This API makes CDeviceAgentClientProxy a singleton class.

Signature of Function:

`CDeviceAgentClientProxy * GetInstance(void);`

Return Value:

Type	Description
CDeviceAgentClientProxy *	0 – Success NULL – failed to create object.



2.1.1.16 ~CDeviceAgentClientProxy – API Function

It is a destructor function. It will be automatically get called on destruction. The destructor function will release resources and initialize the values.

Signature of Function:

```
void ~CDeviceAgentClientProxy (void);
```

Return Value:

Type	Description
void	no return value



3 Annex – I

3.1 Configuration file and argument to AgentService

Device Gateway Agent Service and Proxy will need certain configuration parameters. These parameters will be stored on HDM in the AgentService.config file:

```
### AgentService.conf ###

[General]
# Device Gateway Agent will communicate with Reciprocity Gateway
# to get the Blood Pressure and Weight measurements. Hence Agent
# need to know the IP address of Reciprocity Gateway to communicate
ReciprocityGatewayIp = 192.168.1.100

# Reciprocity Gateway will run a server which will listen to
# connection and request from Device Gateway Agent
# Agent need to know the port of server running on Reciprocity Gateway.
ReciprocityGatewayPort = 1234

# Device Gateway Agent Proxy will communicate with
# AgentService over AF_UNIX socket. The IPC is
# established through PATH/Port. A server will run on
# AgentService whose port/path need to be shared
# with both Agent Proxy and AgentService
GatewayAgentServicePort = "/tmp/GatewayAgent"

# Device Gateway Agent Proxy can have different mode of operation.
# 0 -> Normal; 1 -> Simulator (Stub)
ReciprocityGatewayMode = 1
```

Note: This config file is common to IoT agent

The following arguments will be passed to GatewayAgentService when it is started:

```
/data/agents/gateway/GatewayAgentService --KeepAlive=true --
KeepAliveInterval=60 --
ReciprocityGatewayClientCerti="/data/agents/gateway/CliCerti" --
ReciprocityGatewayServCerti="/data/agents/gateway/ServCerti" --
ReciprocityGatewayTransport=bluetooth --
ReciprocityGatewayConfig="/data/agents/DeviceAgent.conf"
```

Note: The parameters that need input from HDM is part of command line arguments and rest are part of configuration file.

Path of certificates required for establishing TLS1.2, dual authentication ^[2] between Agent Service and Gateway Service will be provided by HDM.

Interface file in HDM should have following in order to get dynamic IP while setting BNEP



File name: /etc/network/interfaces

```
auto bnep0  
iface bnep0 inet dhcp
```




4 Annex – II

4.1 Interface with User Application

4.1.1 CDeviceAgentClientProxy.h

The Device Agent Client Proxy will support JSON and C++ object format. There will be separate library for JSON interface and C++ interface.

The API interface header file will be:

```
class CDeviceAgentClientProxy
{
    CDeviceAgentClientProxy(void);

    public:

    ~CDeviceAgentClientProxy(void);
    CDeviceAgentClientProxy * CGetInstance(void);
    static CDeviceAgentClientProxy * GetInstance(void);
#ifdef JSON_INTERFACE
    int32_t Init(CHdmCbImpl *cHdm,
                void (CHdmCbImpl::*pFnBpm) ( CJson *pBpm),
                void (CHdmCbImpl::*pFnWs) ( CJson *pWs),
                void (CHdmCbImpl::*PFnEvt) ( CJson *pEvt),
                std::string sConfigFilePath);
    int32_t StartAgentService(std::string sCommands);
#else
    int32_t Init(CHdmCbImpl *cHdm,
                void (CHdmCbImpl::*pFnBpm) (CBloodPressureMtr *pBpm),
                void (CHdmCbImpl::*pFnWs) (CWeighingScale *pWs),
                void (CHdmCbImpl::*PFnEvt) (CEvent *pEvt),
                std::string sConfigFilePath);
#endif
    int32_t StopAgentService(void);
    int32_t DeInit(void);
    int32_t GetBpmMeasurement(void);
    int32_t GetWsMeasurement(void);
    int32_t GetCurrentUTC(void);
    std::string GetAgentProxyVersion(void);
    std::string GetAgentServiceVersion(void);
    int32_t IsHdmPaired(void);
    int32_t DelHdmPairing(void);
    int32_t SetDeviceDiscoverable(void);
    int32_t DisableDeviceDiscoverable(void);
    int32_t PairingResponse(std::string sBtAddress, int32_t iUserResponse);
};
```



4.1.2 CGatewayServiceCbImpl.h

```
class CGatewayServiceCbImpl{

    public:
        CGatewayServiceCbImpl();
        ~CGatewayServiceCbImpl();
#ifdef JSON_INTERFACE
        void BpmHandler(CBloodPressureMtr *BpmMtr);
        void WsHandler(CWeighingScale *WsMtr);
        void EventHandler(CEvent *Evt);
#else
        void BpmHandler(Json *BpmMtr);
        void WsHandler(Json *WsMtr);
        void EventHandler(Json *Evt);
#endif
};
```

4.2 Deliverables for HDM

Following diagram displays the list of files required for HDM to interface with Gateway Agent.

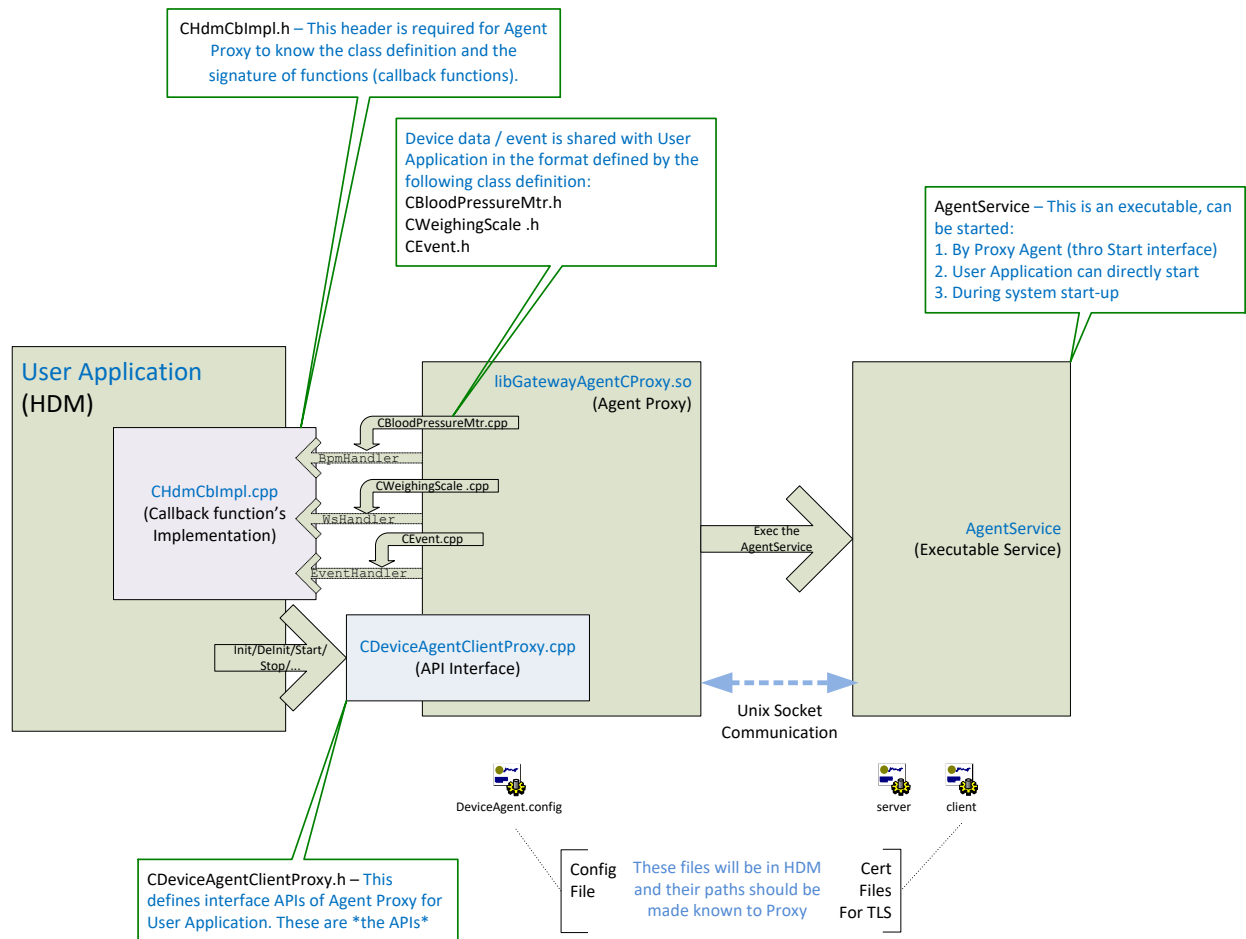


Figure 3: Agent Proxy – Agent Service -HDM interaction diagram w.r.t. files

library	Details
libGatewayAgentCProxy.so	It is the proxy .so file.
bin	Details
GatewayAgentService	It is an executable that takes runtime parameter and starts a demon process.
header	Details
CDeviceAgentClientProxy.h	This file defines the API interface of Device Agent Client Proxy (Gateway Agent).
CHdmCbImpl.h	This file contains the definition of callback function and its class. The implementation of this class will be done by User Application (HDM). However its class definition is required for Proxy to register and invoke callback methods.
CEvent.h	This file defines the Event class. Event will be passed to callback method as an object of this class. <i>It has getter – setter methods.</i>



CBloodPressureMtr.h	This file defines the class for Blood Pressure Measurement. BPM data will be passed to callback method as an object of this class. <i>It has getter – setter methods.</i>
CWeighingScale.h	This file defines the class for Weighing Scale Measurement. WS data will be passed to callback method as an object of this class. <i>It has getter – setter methods.</i>
CPeripheralDevice.h	This file is a base class for CBloodPressureMtr.h and CWeighingScale.h. It has some common structure.