IoT Hub Schema Client Kit APIs v3

# Overview

The IoT Hub Schema APIs allows developers to quickly and easily define models for their devices directly as code, while supporting the required features for modeling devices (including multiple models and multiple devices within the same application).

# IOT model description macro

## Example

In the header file:

#include "iotagent.h"

BEGIN\_IOT\_DECLARATIONS(MyFunkyTV)

DECLARE\_IOT\_STRUCT(MenuType,

int, source,

double, brightness

);

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

WITH\_DATA(MenuType, tvMenu),

WITH\_ACTION(LostSignal, int, source, int, resolution)

);

DECLARE\_IOT\_MODEL(AnotherDevice,

…

);

END\_IOT\_DECLARATIONS(MyFunkyTV);

## BEGIN\_IOT\_DECLARATIONS(schemaNamespace)

This macro marks the start of a section that declares IOT model elements (like complex types, etc.). Declarations are typically placed in header files, so that they can be shared between translation units.

## END\_IOT\_DECLARATIONS(schemaNamespace)

This macro marks the end of a section that declares IOT model elements

## DECLARE\_IOT\_STRUCT(structTypeName, field1Type, field1Name, …)

This macro allows declaring a struct (complex) type for an IOT model.

Arguments:

* structTypeName – specifies the struct type name
* (fieldXType, fieldXName) – The type and the name for the field X of the struct type. A struct type can have any number of fields from 1 to 61 (inclusive). At least one field must be defined.

Example:

DECLARE\_IOT\_STRUCT(MenuType,

int, source,

double, brightness

);

## DECLARE\_IOT\_MODEL(modelName, element1, …)

This macro allows declaring a model that can be later used to instantiate a device.

Arguments:

* *modelName* – specifies the model name
* element1, element2, … – a model element (can be a property and action).
  + A property is described in a model by using the WITH\_DATA.
  + An action is described in a model by using the WITH\_ACTION macro.

Example:

DECLARE\_IOT\_MODEL(FunkyTV,

…

);

## WITH\_DATA(propertyType, propertyName)

The WITH\_DATA macro allows declaring a model property in a model. A property can be published by using the SEND macro.

Arguments:

* *propertyType* – specifies the property type. Can be any of the following types:
  + int
  + double
  + float
  + long
  + int8\_t
  + uint8\_t
  + int16\_t
  + int32\_t
  + int64\_t
  + bool
  + ascii\_char\_ptr
  + EDM\_DATE\_TIME\_OFFSET
  + EDM\_GUID
  + EDM\_BINARY
  + Any struct type previously introduced by another DECLARE\_IOT\_STRUCT.
* *propertyName* – specifies the property name

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

## WITH\_ACTION(actionName, arg1Type, arg1Name, …)

The WITH\_ACTION macro allows declaring a model action.

Arguments:

* *actionName* – specifies the action name.
* *argXtype, argXName* – defines the type and name for the Xth argument of the action. The type can be any of the primitive types or a struct type.

DECLARE\_IOT\_MODEL(FunkyTV,

…

WITH\_ACTION(channelChange, ascii\_char\_ptr, Property1)

);

## GET\_MODEL\_HANDLE(schemaNamespace, modelName)

The GET\_MODEL\_HANDLE macro returns a model handle that can be used in subsequent operations like generating the CSDL schema for the model, uploading the schema, creating a device, etc.

# Acting on models

## SERIALIZE(destination, destinationSize, property1, …)

This macro produces JSON serialized representation of the properties.

Arguments:

* destination – pointer to an unsigned char\* that will receive the serialized data.
* destinationSize – Pointer to a size\_t that gets written with the size in bytes of the serialized data
* property1, property2, … - a list of property values to send. The order in which the properties appear in the list does not matter, all values will be sent together.

Returns:

* CODEFIRST\_ OK on success
* Any other value on failure

…

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

…

int main(int argc, char\*\* argv)

{

…

FunkyTV\* funkyTV =

CREATE\_DEVICE(MyFunkyTV, FunkyTV);

unsigned char\* destination; size\_t destinationSize;

funkyTV->hasEthernet = false;

funkyTV->screenSize = 42;

SERIALIZE(&destination, &destinationSize, funkyTV->hasEthernet, funkyTV->screenSize);

printf("serialized data is %\*.\*s\r\n",(int)destinationSize, (int)destinationSize, (char\*)destination);

…

}

## SERIALIZE\_DELAYED(destination, destinationSize, property1, property2, …)

This macro allows batching data indicated by property1, property2... The parameters destination, destinationSize are not used, but they must point to an unsigned char\* and to a size\_t (they cannot be NULL).

Arguments:

* destination – points to an unsigned char\*. It is not used, but must be non-NULL.
* destinationSize - points to a size\_t. It is not used, but must be non-NULL.
* property1, property2, … - a list of property values to send. The order in which the properties appear in the list does not matter, all values will be sent together.

Returns:

* CODEFIRST\_OK on success
* Any other value on failure

## SERIALIZE\_DELAYED\_DATA(device, destination, destinationSize)

BEGIN\_IOT\_DECLARATIONS(MyFunkyTV)

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

int main(int argc, char\*\* argv)

{

…

FunkyTV\* funkyTV =

CREATE\_DEVICE(iotHubClientHandle, TRANSPORT\_THREADED, MyFunkyTV, FunkyTV);

unsigned char\* destination; size\_t destinationSize;

funkyTV->hasEthernet = false;

funkyTV->screenSize = 42;

SERIALIZE\_DELAYED(&destination, &destinationSize, funkyTV->hasEthernet, funkyTV->screenSize);

…

}

This macro will serialize all delayed batched data for a specific device. If the device does not have data to be sent, it will ignore the command.

Arguments:

* *device* – A device pointer that identifies the device to send batched data.
* *Destination* - pointer to an unsigned char\* that recieves the batched data
* *destinationSize* - pointer to a size\_t that received the size of the batched data in bytes.

## EXECUTE\_COMMAND

BEGIN\_IOT\_DECLARATIONS(MyFunkyTV)

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

…

void handleIoTHubCallBack(IOTHUB\_CLIENT\_CONFIRMATION\_RESULT result, void\* userContextCallback)

{

<YOUR HANDLE CODE HERE>

}

…

int main(int argc, char\*\* argv)

{

…

FunkyTV\* funkyTV =

CREATE\_DEVICE(iotHubClientHandle, TRANSPORT\_THREADED, MyFunkyTV, FunkyTV);

unsigned char \*destination; size\_t destinationSize;

funkyTV->hasEthernet = false;

funkyTV->screenSize = 42;

SERIALIZE\_DELAYED(&destination, &destinationSize, funkyTV->hasEthernet, funkyTV->screenSize); /\*here destination, destinationSize play no role)

…

SERIALIZE\_DELAYED\_DATA(device, &destination, &destinationSize);

printf("serialized data is %\*.\*s", (int)destinationSize, (int)destinationSize, (char\*) destination);

…

}

Any action that is declared in a model must also have an implementation as a C function.

The C function arguments must be:

* First argument must be of the type pointer to device data (i.e. FunkyTV\*).
* Following arguments must match the arguments declared in the model action.

The macro EXECUTE\_COMMAND(device, commandBuffer) shall execute the command indicated in the commandBuffer for the device.

# IOTHub\_Schema Client APIs

DECLARE\_IOT\_MODEL(MyFunkyTV,

…

WITH\_ACTION(changeChannel, ascii\_char\_ptr, Property1)

);

…

void changeChannel(FunkyTV\* device, ascii\_char\_ptr Property1)

{

printf(“Changing Channel to channel %s\r\n”, Property1);

}

#define COMMAND\_TEXT "\"Name\":\"changeChannel\",\"Parameters\":\"FabrikamTV\""

int main(void)

{

...

EXECUTE\_COMMAND(funkyTv, COMMAND\_TEXT);

...

}

## IOTHUB\_SCHEMA\_CLIENT\_RESULT iothub\_schema\_client\_init(const char\* overrideSchemaNamespace)

Initializes the library.

Arguments:

* *overrideSchemaNamespace* – An override schema namespace to use for all models. Optional, can be NULL.

If *schemaNamespace* is not NULL, its value shall be used instead of the namespace defined for each model by using the DECLARE\_XXX macro.

Returns:

* IOTHUB\_SCHEMA\_CLIENT\_OK on success
* Any other error on failure

Example:

## void iothub\_schema\_client\_deinit()

int main(int argc, char\*\* argv)

{

…

if (iothub\_schema\_client\_init(NULL) != IOTHUB\_SCHEMA\_CLIENT\_OK)

{

/\* error \*/

}

else

{

…

}

}

Shuts down the IOT library. The library will track all created devices and upon a call to iothub\_schema\_client\_deinit it will de-initialize all devices.

Example:

int main(int argc, char\*\* argv)

{

…

(void)iothub\_schema\_client\_init(NULL);

…

iothub\_schema\_client\_deinit();

}

## void\* CREATE\_DEVICE(schemaNamespace, modelName, serializerIncludePropertyPath)

Initializes a device that has the model identified by the schema Namespace and Model Name.

Iothub\_schema lib will track all created devices and upon a call to iothub\_schema\_client\_deinit it will de-initialize all devices.

Arguments:

* *schemaNamespace* – Theschema namespace as specified in BEGIN\_IOT\_DECLARATION macro.
* *modelName* – The model name, as defined with the DEFINE\_IOT\_MODEL macro.
* *serializerIncludePropertyPath* – an optional bool argument. Default value: false. If set to true it instructs the serializer to include the full property path (including the property name) in the resulting JSON. If set to false, the property path (and name) will not appear in the resulting JSON that is sent to the service.

Returns:

* A pointer to a structure of type *modelName* that can be used in further calls to send data, etc.
* NULL in case of an error.

Example:

DECLARE\_IOT\_MODEL(MyFunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

int main(int argc, char\*\* argv)

{

…

(int)iothub\_schema\_client\_init(NULL);

FunkyTV\* funkyTV =

CREATE\_DEVICE(MyFunkyTV, FunkyTV);

…

funkyTV->hasEthernet = false;

funkyTV->screenSize = 42;

unsigned char\* destination; size\_t destinationSize;

SERIALIZE(&destination, &destinationSize, funkyTV->hasEthernet, funkyTV->screenSize);

printf("serialization = %\*.\*s", (int)destinationSize, (int)destinationSize, destination);

}

## DESTROY\_DEVICE(device)

Frees any resources associated with the device.

Arguments:

* *device* – A device pointer that identifies the device to de-initialize.

BEGIN\_IOT\_DECLARATIONS(MyFunkyTV);

DECLARE\_IOT\_MODEL(FunkyTV,

WITH\_DATA(int, screenSize),

WITH\_DATA(bool, hasEthernet),

…

);

END\_IOT\_DECLARATIONS(MyFunkyTV);

…

int main(int argc, char\*\* argv)

{

…

FunkyTV\* funkyTV =

CREATE\_DEVICE(MyFunkyTV, FunkyTV);

…

DESTROY\_DEVICE(device);

…

}