

# ObjectFlow for Microcontrollers

Simple and Reliable Data  
Flow Graph programming  
for embedded applications

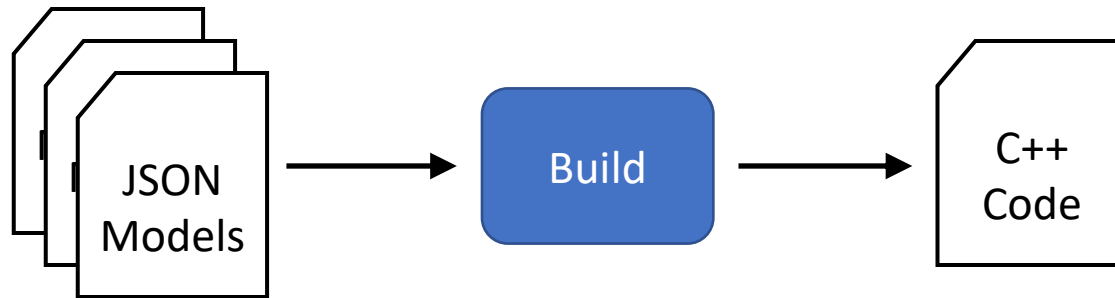
# ObjectFlow

- Data-driven programming for tiny microcontrollers
- Arduino Uno class (2KB/32KB) and larger
- Small library, no central executive, timer- and communication event-driven
- Similar to IEC61499, Node-RED, etc. based on Data Flow Graphs (DFG)
- Uses the LWM2M data model and semantics with an event-driven communication protocol
- Communication is implemented in the application layer using a set of well-known LWM2M types

# ObjectFlow - Digital Twin for Embedded Code

- On-device code is simple, minimal, and reliable
- A small set of primitives implemented as static C++ wrapper classes for Object and Resource
- Graphs are build from a standardized JSON format that models Object and Resource Instances on the embedded device – a digital twin of the device code
- Tools construct a C++ header file template that is built with the device code application handler bundle using the standard IDE (e.g. Arduino IDE), and downloaded to the device in the usual way.

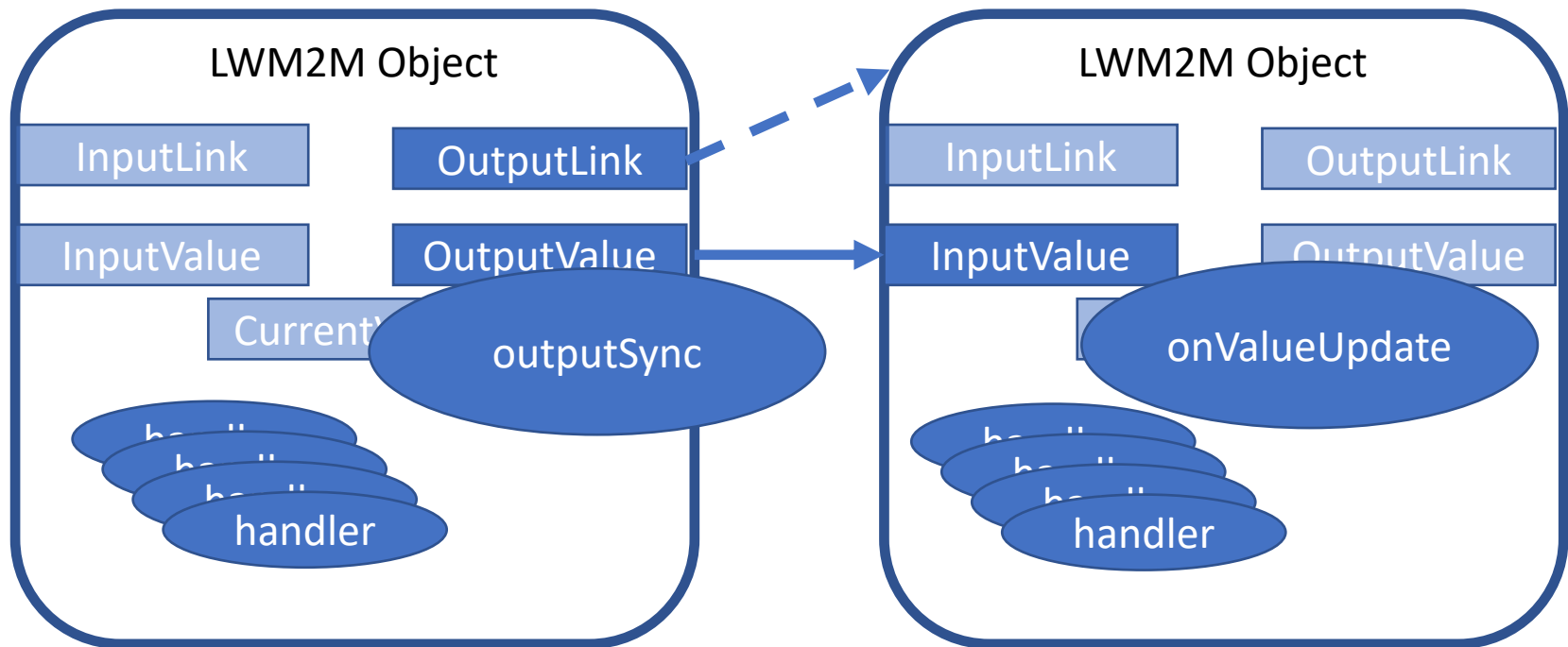
# High Level Process



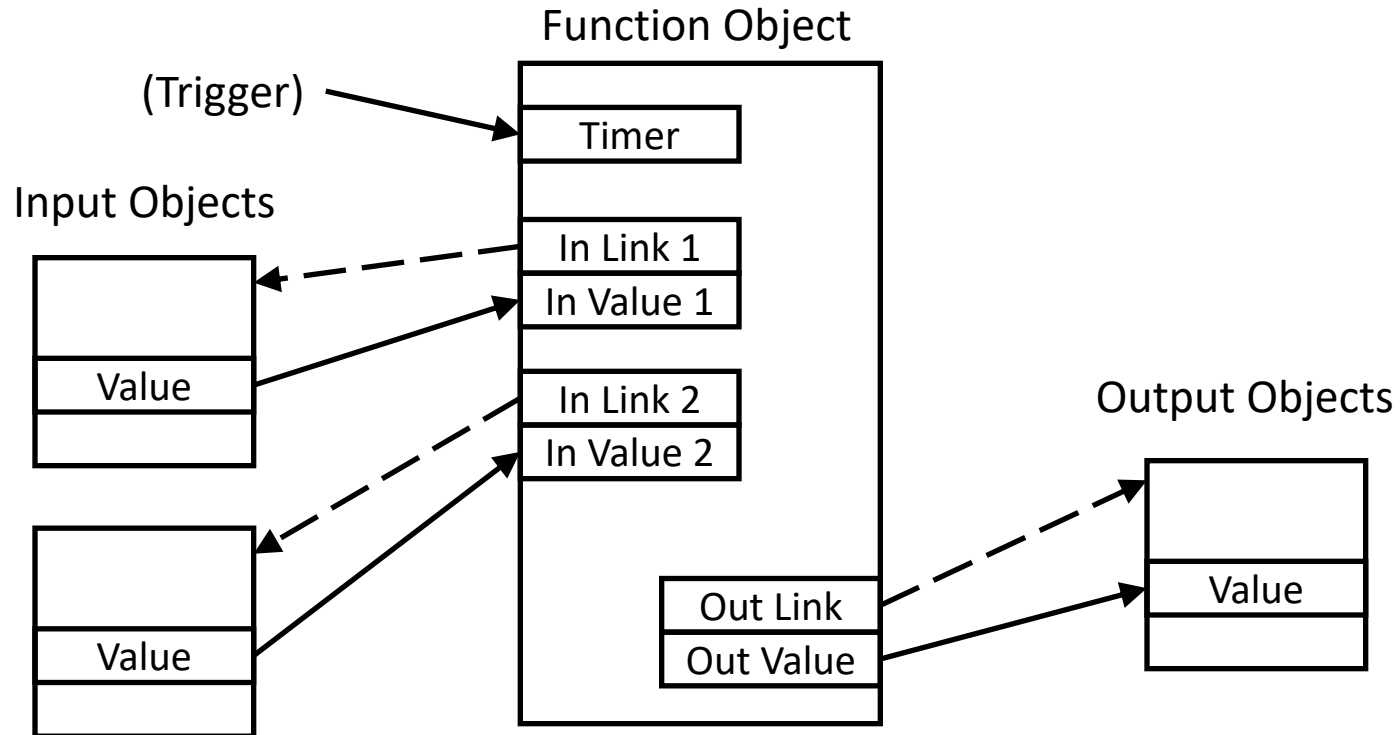
# ObjectFlow – DFG wrapper

- A small set of Resource types for communication
  - Time types for Current, Interval, and LastActivation
  - Input and Output link types based on LWM2M Objlink
  - Value subtypes for Input, Output, and Current Value
- DFG - communication is event driven and initiated according to application logic
  - Bound methods implement application handlers
  - InputSync (data pull) and OutputSync (data push)
  - onValueUpdate, onValueSync, onInterval handlers

# ObjectFlow – DFG and Reactive Communication



# Complex Functions



# ObjectFlow – DFG architecture

- An ObjectFlow application is a Data Flow Graph
- DFG Nodes are implemented as LWM2M Objects
- A DFG Node is a collection of one or more Objects
- LWM2M Object Links (Objlink) are used to group Objects into a DFG Node and to implement the data flow connections between Nodes (arcs, edges)



# Example graph

TimeSource:

Type: TimeSource  
IntervalTime: 1000  
TimerOutputLink: AnalogInput

AnalogInput:

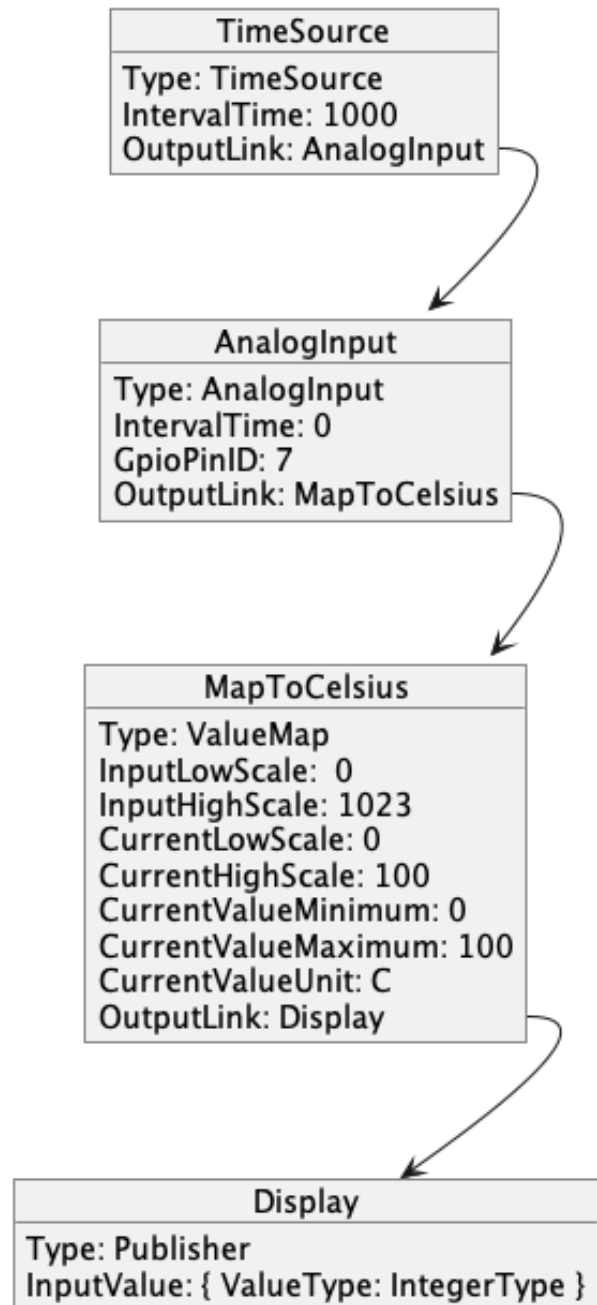
Type: AnalogInput  
GpioPinID: 7  
IntervalTime: 0  
OutputLink: MapToCelsius

MapToCelsius:

Type: ValueMap  
InputLowScale: 0  
InputHighScale: 1023  
CurrentLowScale: 0  
CurrentHighScale: 100  
CurrentValueMinimum: 0  
CurrentValueMaximum: 100  
CurrentValueUnit: C  
OutputLink: Display

Display:

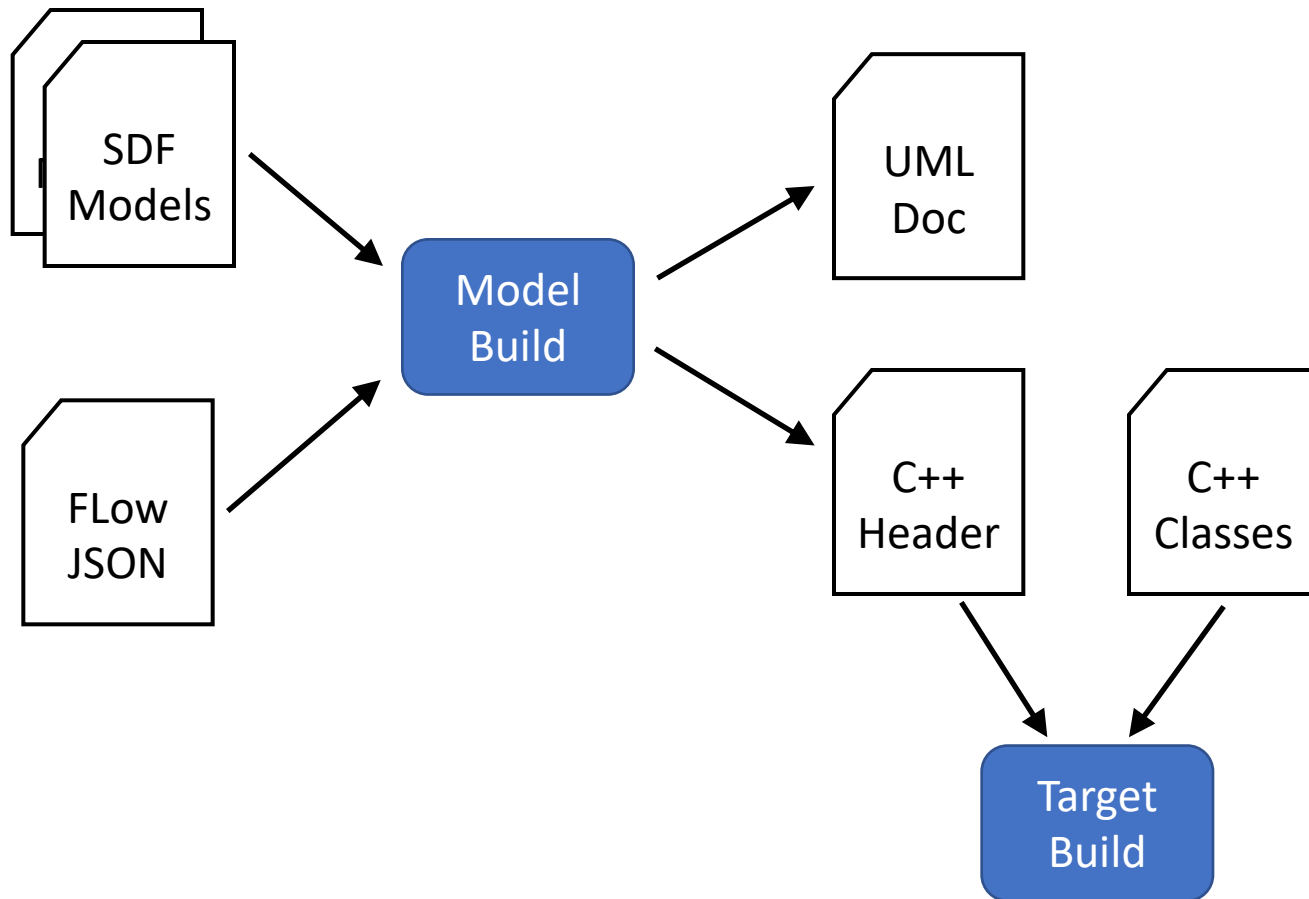
Type: Publisher  
InputValue: { ValueType: IntegerType }



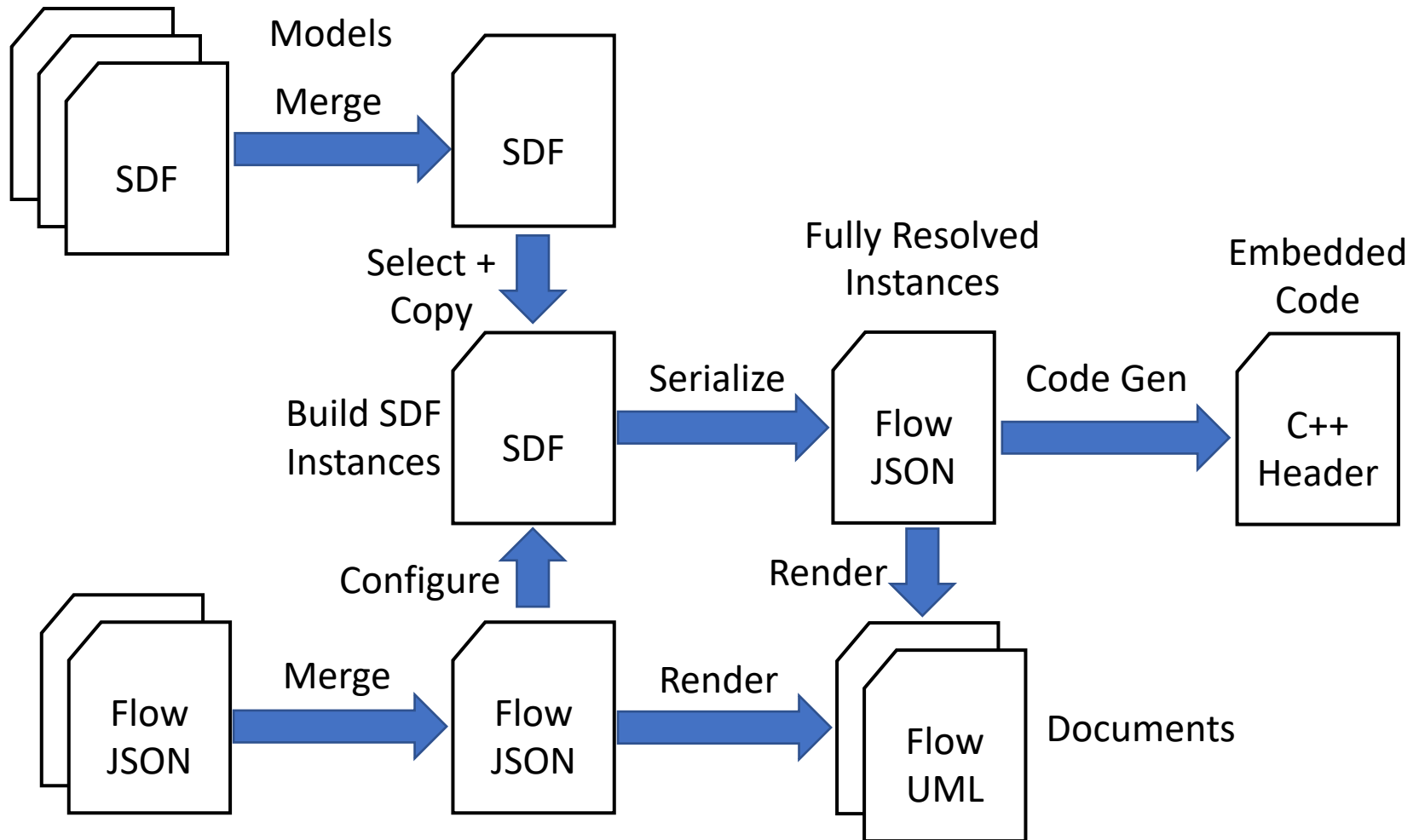
# ObjectFlow - Tools

- The DFG model tools do all of the heavy lifting so the embedded code can be extremely simple and lightweight, e.g. type checking is done in the model
- Code generation involves serialization of the objects and resources into a C++ header file, and packaging of the implementation code for the application objects (time and data event handlers)
- The result is a standard C++ package that can be built by any C++ toolchain (Arduino, etc.)

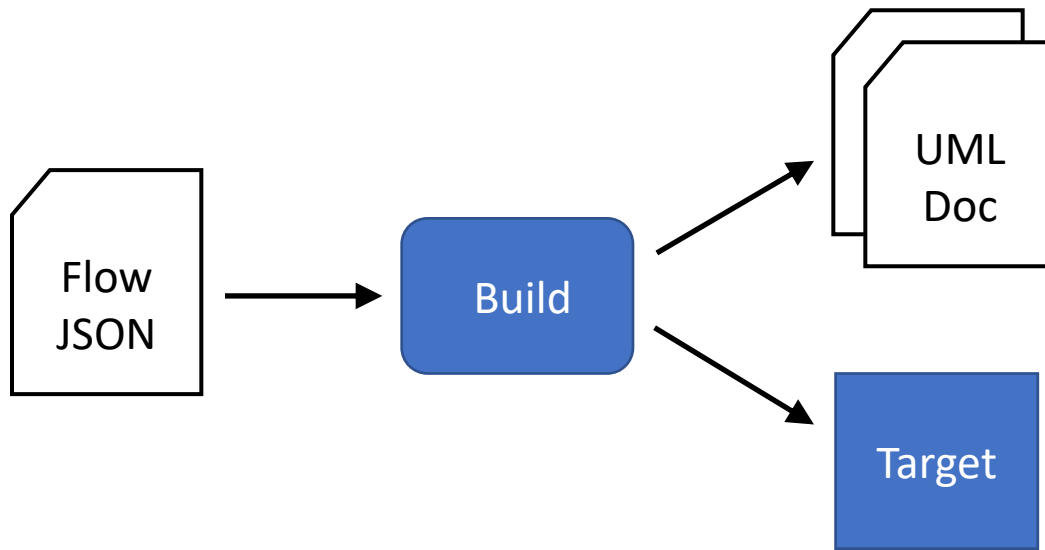
# High Level Process



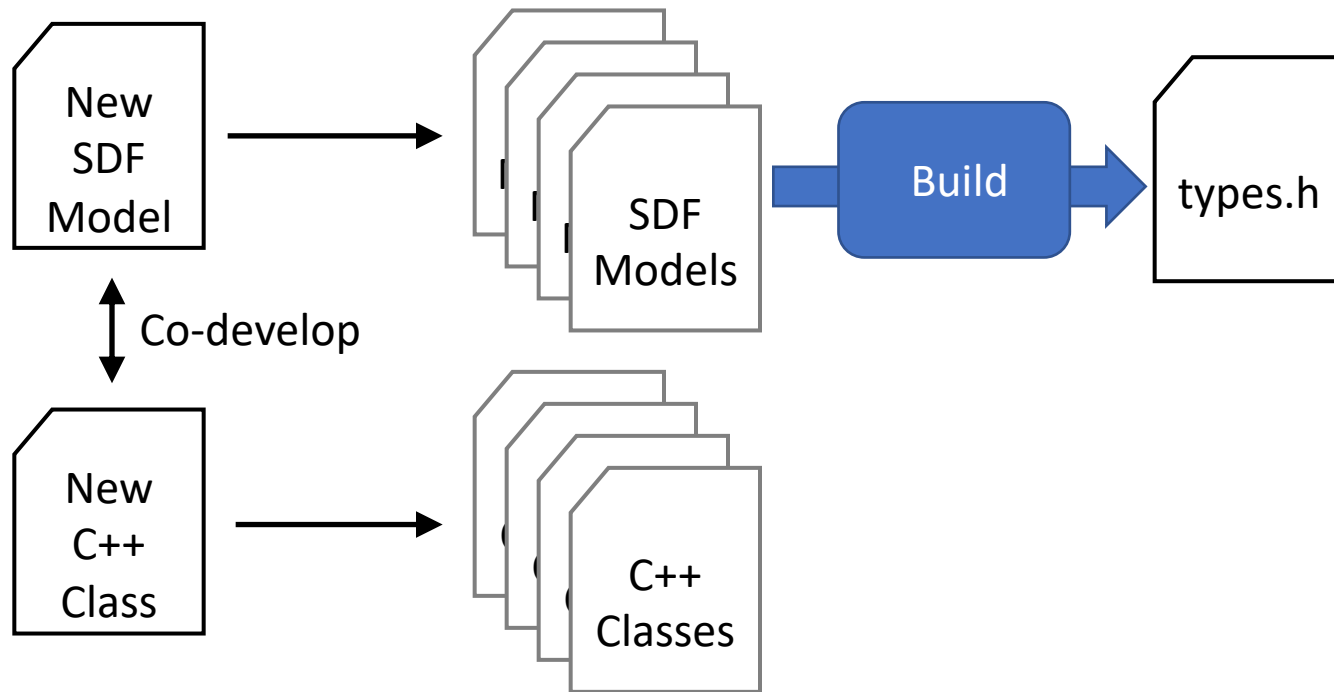
# Build Process



# Making a New Flow Graph



# Making a New Object



# Example SDF Model

```
defaultnamespace: objectflow
```

```
sdfData:
```

```
  # add this ObjectType ID to the TypeID registry
```

```
  TypeID:
```

```
    ObjectType:
```

```
      Publisher: { const: 43008 }
```

```
sdfThing:
```

```
  # Publisher Object
```

```
  Publisher:
```

```
    sdfRef: /#/sdfThing/ObjectFlowObject
```

```
    sdfData:
```

```
      TypeID:
```

```
        sdfRef: /#/sdfData/TypeID/ObjectType/Publisher => TypeID: { const: 43008 }
```

```
  # Publisher Object Resources
```

```
  sdfRequired:
```

```
    - /#/sdfThing/Publisher/sdfThing/InputValue
```

```
  sdfThing:
```

```
    InputValue:
```

```
      sdfRef: /#/sdfThing/ObjectFlowObject/sdfThing/InputValue
```

```
      ValueType:
```

```
        sdfChoice:
```

```
          default: { sdfRef: /#/sdfData/ValueType/sdfChoice/IntegerType }
```

```
sdfAction:
```

```
  OnDefaultValueUpdate: Publish the data to the endpoint
```

# Example SDF Instance

```
defaultnamespace: objectflow

sdfThing:
  # Instance Graph
  InstanceGraph:
    sdfRef: /#/sdfThing/ObjectList
    # Objects in the graph
    sdfThing:

      TimeSource:
        sdfRef: /#/sdfThing/TimeSource
        sdfThing:
          IntervalTime:
            sdfProperty:
              Value:
                sdfChoice:
                  IntegerType: { const: 1000 }
          OutputLink:
            sdfData:
              SdfLink:
{ InstancePointer: /#/sdfThing/InstanceGraph/sdfThing/AnalogInput }

      AnalogInput:
        sdfRef: /#/sdfThing/AnalogInput
        sdfThing:
          GpioPinID:
            sdfProperty:
              Value:
                sdfChoice:
                  IntegerType: { const: 7 }
          OutputLink:
            sdfData:
              SdfLink:
{ InstancePointer: /#/sdfThing/InstanceGraph/sdfThing/ScaleMapper }
```



```
defaultnamespace: objectflow
```

```
sdfThing:
```

```
# Instance Graph
```

```
InstanceGraph:
```

```
sdfRef: /#/sdfThing/ObjectList
```

```
# Objects in the graph
```

```
sdfThing:
```

```
TimeSource:
```

```
sdfRef: /#/sdfThing/TimeSource
```

```
sdfThing:
```

```
IntervalTime:
```

```
sdfProperty:
```

```
Value:
```

```
sdfChoice:
```

```
IntegerType: { const: 1000 }
```

```
TimerOutputLink:
```

```
sdfData:
```

```
SdfLink:
```

```
{ InstancePointer: /#/sdfThing/InstanceGraph/sdfThing/AnalogInput }
```

```
AnalogInput:
```

```
sdfRef: /#/sdfThing/AnalogInput
```

```
sdfThing:
```

```
GpioPinID:
```

```
sdfProperty:
```

```
Value:
```

```
sdfChoice:
```

```
IntegerType: { const: 7 }
```

```
OutputLink:
```

```
sdfData:
```

```
SdfLink:
```

```
{ InstancePointer: /#/sdfThing/InstanceGraph/sdfThing/MapToCelsius }
```

# Simplified DSL

```
TimeSource:
```

```
Type: TimeSource
```

```
IntervalTime: 1000
```

```
TimerOutputLink: AnalogInput
```

```
AnalogInput:
```

```
Type: AnalogInput
```

```
GpioPinID: 7
```

```
OutputLink: MapToCelsius
```

# Reserved LWM2M Types

TypeID:

ResourceType:

```
ValueType: { const: 27000 }  
InputLinkType: { const: 27001 }  
OutputLinkType: { const: 27002 }  
InputValueType: { const: 27003 }  
CurrentValueType: { const: 27004 }  
OutputValueType: { const: 27005 }  
CurrentTimeType: { const: 27006 }  
IntervalTimeType: { const: 27007 }  
LastActivationTimeType: { const: 27008 }
```

SDF Definition

Generated  
C++ Header

```
#define InputLinkType 27000  
#define InputLinkType 27001  
#define OutputLinkType 27002  
#define InputValueType 27003  
#define CurrentValueType 27004  
#define OutputValueType 27005  
#define CurrentTimeType 27006  
#define IntervalTimeType 27007  
#define LastActivationTimeType 27008
```

# Resolved Instance Graph

