How To Implement Changeable-Type Datapoints With III

Revision 2, October 7, 2015, Bernd Gauweiler

Abstract

This memo introduces the problem of changeable-type datapoints and discusses the implementation of changeable-type datapoints with the IzoT Interface Interpreter (III) using the IzoT Interface Interpreter Meta Language (IML).

Introduction

A datapoint typically implements a member of a profile and serves a particular, well-defined purposed within that profile. For example, the *SFPTcalendar* profile dictates that its *nvoDateEvent* output datapoint member implements a *SNVT_date_event* data type. Implementing this profile's *nvoDateEvent* member using any other data type would render the implementation of the profile invalid.

However, some profiles define generic functionality. For example, the SFPTdataLogger profile can be implemented and configured to log data from a range of datatypes. Such profiles have two degrees of freedom, which can be used separately or in combination:

A profile can define the datatype of a member datapoint as *SNVT_xxx.* This is a special placeholder type. Defining a profile's data member as *SNVT_xxx* means that this member's profile may be implemented using any datapoint type within scope[1]. The implementation type is chosen at the time the profile is implemented ("compile-time").

A profile defining multiple member datapoints with the SNVT_xxx placeholder generally requires that all SNVT_xxx references are implemented using one and the same datatype. A specific profile may define different rules and support, or even require, different datatypes.

A profile can indicate that the datatype of a datapoint is not fixed at compile-time. Instead, the datatype is initialized at compile-time, but may be changed at network configuration time. A datapoint whose datatype is not fixed at compile-time is known as a *changeable-type datapoint.* To indicate that the implementation of a given profile member datapoint as a changeable-type datapoint is permitted, the profile supports a *SCPTnvType* property. This property must apply to the datapoint in question. The SCPTnvType property can also apply to the profile. In this case, the property determines the type of the profile's principal datapoint member.

[1] That is, an implementation of a standard profile can only use standard datapoint types. An implementation of a user-defined profile applicable to a particular program ID mask and scope selector value can only use standard datapoint types or any user-defined datapoint types which apply to the same program ID mask and the same or a numerically lower scope selector value.

The initial datatype is determined by the profile, but can also be defined as *SNVT_xxx.* The implemented initial type also defines the maximum size for the type of this datapoint. The actual datatype is determined by the *SCPTnvType* property which applies to the datapoint in question. This type is change at runtime, typically during network configuration and installation. To change the datatype, the network integrator assigns the description of the new datatype to this property.

The application validates the assignment and ensures that only datatypes compatible with the profile and the application are used. For example, a generic PID controller could support any floating-point datatypes (e.g. SNVT_volt_f, SNVT_temp_f, etc), but no non-numeric, non-float types such as SNVT_switch or SNVT_alarm.

Rejection of the desired type is indicated with a raised *invalid_request* flag in the Node Object's status output. This must occur within 30s from the type change request. The refusing application resets the property and all related algorithms to the last known good type in this case.

What III Does For You

To implement a profile's datapoint member as a changeable-type datapoint, implement the SCPTnvType property which governs the datapoint's datatype.

For example, the SFPTdataLogger profile defines a *nviDataValue* as *SNVT_xxx*, and allows for the implementation of a *SCPTnvType* with its optional *cpNVType* property member. The following IML construct implements the SFPTdataLogger profile in block *dl*, and implements this property member:

```
SFPTdataLogger(1, SNVT_volt) dl; //@Izot block implement(nviDataValue.cpNVType)
```

When III implements the SCPTnvType, a number of things happen:

1. III makes sure that the program ID includes the *changeable-type* flag (bit 0x80 in the 6th byte, also known as the *usage* field).

   III will raise this flag and will warn if the flag wasn't originally set (this warning becomes an error if the IML option 'strict' is enabled). To avoid this warning (or error), define the correct program ID with the IML *programId* option.

   For example:

   //@IzoT Option programId("9F:FF:FE:01:54:80:1A:00")

2. III generates self-documentation data to indicate that this datapoint implements a changeable type (a single question mark appears in the datapoint's self-documentation string).

3. For the CPM 4200 SDK, III generates a synchronous event handler within IzotDev.c to supply the IzoT Device Stack DX with the current size of the changeable-type datapoint. This size is obtained from the SCPTnvType property using this algorithm:

   If SCPTnvType.type_category indicates NVT_CAT_INITIAL (the default value), use the initial datatype size.
   Otherwise, if SCPTnvType.type_length is zero, or larger than the initial datatype size, use the initial datatype size.
   Otherwise, use SCPT_nvType.type_length.

   For IzoT ShortStack SDK, III implements the LonGetCurrentNvSize() handler in the generated ShortStackDev.c file, using the same logic as with the CPM 4200 SDK described above.

What You Must Do For III

To assist III, your implementation of the changeable type protocol must examine a new value assignment to SCPTnvType. You must accept or reject this type assignment within 30s.

To reject the type assignment, raise the *invalid_request* flag in your node object's status output and revert to the last known good type. When this information is not available, set the *type_category* field to *NVT_CAT_INITIAL* to return to the initial datatype.

To accept the type assignment, ensure that the type_length field is updated with the correct *network data size*.

For users of the CPM 4200 SDK, some datatypes' presentation differs slightly from their presentation on the network. The *sizeof()* operator always returns the true host size of the type, but may not yield the correct network size[2]. You can use the *IzotGetDeclaredNetworkSizeByIndex()* utility to obtain the correct network size of a type.

To handle values of the changeable-type datapoint, you may need to cast the static (initial) data type to the current type according to SCPTnvType.

[2] A difference between network and host datatype size occurs when the datatype includes one or more *union* members.