# Allowing multiple encodings for TTCN-3 types

Today the type of encoding of the test data is not fixed in the specification in all cases. The simplest example is that IoT common service level data is specified at the abstract level and can be encoded by XML or by JSON, in the discretion of the vendor. The abstract data structure can well be defined in TTCN-3, however the different possible types of encoding is not supported by the language.

Therefore TTCN-3 shall, as well, be able to encode and decode the same abstract data structure in different encodings.

## Assumptions

The actual encoding and decoding can be controlled in a session- static way, i.e. by runtime parameterization, causing that within a test session only one type of encoding is used; or dynamically, when the type of encoding can be selected at each encoding/decoding event, i.e. communication events, encvalue/decvalue etc. individually.

As dynamic control would affect the language at many places, it is proposed to separate the introduction to two phases:  
- the first phase could be session-static control+: for sending and receiving operations the actual encoding type is controlled by a runtime parameter (it’s implementation is tool-specific), but in addition it would also allow to override the default encoding for individual encvalue, decvalue, encvalue\_unichar, decvalue\_unichar and decmatch calls (as @decoded is related to communication operations, it need not support this feature and also would be difficult syntactically). This will allow workarounds in the TTCN-3 code, when session-static encode selection is not sufficient.  
- if required by a use case, add dynamic control later.

The two phases should also be separated to different CRs, as we believe, it is easier to agree and more quick to implement.

This CR deals with static control+ only.

## Needed new features

Session-static control+ of multiple encodings will require the following extensions of the core language:

* Allow attaching multiple encode with attributes to TTCN-3 types.
* Allow explicit scoping of variant attributes.
* Extend the above predefined functions to enable control of codec selection.

### 2.1 Attaching several encodings to types and type fields

It requires minimal changes only; the BNF allows adding multiple encode attributes to a language element, only text of clause 27.4 need to be amended to allow multiple encode attributes.

Encoding attribute overwriting rules need not be changed, it need just be clarified that one or more overriding encode attribute overwrites all encodings associated to the external scope.

NOTE: it could also be possible to allow a syntax, where one encode attribute lists more encodings, e.g. something like: **encode** “XML”, “JSON”; While the advantage of this is would be minimal, this would open the door to constructs like “several strings following an attribute”, which is in general could be a dangerous direction, and hence discouraged.

### 2.2 Explicit scoping of variant attributes

Variant attributes are controlling the options related to a given encoding. Therefore it is necessary to unanimously identify, to which encoding a given variant belongs to.

Note: In principle, a technical solution could be found to associate a variant to more than one encodings implicitly, but this could lead to ambiguities, hence strongly recommended to be avoided (i.e. when another encode is added to a module, suddenly a variant, originally meant to the first encode only, becomes associated to several encode-s).

Proposed rules:

* It is proposed to use an explicit scope:value syntax, i.e.  
  **variant** [“<encode>”:]”<variant value>”;  
  to establish the scope of the variant attribute value.
* If a type/field has a single encode associated to it, the   
  “<encode>”:  
  part is optional; this ensures backward compatibility with existing TTCN-3 codebase. The “:” identifies the border between the scoping and the value parts (i.e. **variant** "Foo" "Bar"; shall give an error, as is it unclear if Foo meant to be the encode scope or by error several variant strings are present).
* If a type/field has multiple encodes associated to it, the   
  “<encode>”:  
  part is mandatory in the variant-s; i.e. a missing encode scope shall cause an error.
* Variant attribute (i.e. encoding instruction) inheritance and accumulation rules are encode-specific; the TE has the responsibility to collect them and send to the codec in the form, expected by the codec.

EXAMPLES

**Example 1**: Typical XML encoding today == single encode case in the future.

**module** Types {

**type** **record** AE

{

NCName resourceName **optional**,

ResourceType resourceType **optional**,

ID resourceID **optional**,

NhURI parentID **optional**,

Timestamp creationTime **optional**,

Timestamp lastModifiedTime **optional**,

**union** {

**record** **length**(1 .. **infinity**) of ChildResourceRef childResource\_list,

**record** **length**(1 .. **infinity**) of **union** {

Container container,

Group group\_,

AccessControlPolicy accessControlPolicy,

Subscription subscription,

PollingChannel pollingChannel,

Schedule schedule

} choice\_list

} choice **optional**

}

**with** {

**variant** "element";

**variant** (resourceName) "attribute";

**variant** (choice) "untagged";

**variant** (choice.childResource\_list) "untagged";

**variant** (choice.choice\_list) "untagged";

**variant** (choice.choice\_list[-]) "untagged";

/\* long -short name conversion XML\*/

**variant** "name as 'ae'";

**variant** (resourceName) "name as 'rn'";

**variant** (resourceType) "name as 'ty'";

**variant** (resourceID) "name as 'ri'";

**variant** (parentID) "name as 'pi'";

**variant** (creationTime) "name as 'ct'";

**variant** (lastModifiedTime) "name as 'lt'";

**variant** (choice.childResource\_list[-]) "name as 'cr'";

**variant** (choice.choice\_list[-].container) "name as 'cnt'";

**variant** (choice.choice\_list[-].group\_) "name as 'grp'";

**variant** (choice.choice\_list[-].accessControlPolicy) "name as 'acp'";

**variant** (choice.choice\_list[-].subscription) "name as 'sub'";

**variant** (choice.choice\_list[-].pollingChannel) "name as 'pch'";

**variant** (choice.choice\_list[-].schedule) "name as 'sch'";

};

}

**with** {

**encode** "XML";

**variant** "namespace as 'http://www.onem2m.org/xml/protocols' prefix 'm2m'";

**variant** "controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

}

**Example 2**: XML & JSON encodings.

**module** Types {

**type** **record** AE

{

NCName resourceName **optional**,

ResourceType resourceType **optional**,

ID resourceID **optional**,

NhURI parentID **optional**,

Timestamp creationTime **optional**,

Timestamp lastModifiedTime **optional**,

**union** {

**record** **length**(1 .. **infinity**) of ChildResourceRef childResource\_list,

**record** **length**(1 .. **infinity**) of **union** {

Container container,

Group group\_,

AccessControlPolicy accessControlPolicy,

Subscription subscription,

PollingChannel pollingChannel,

Schedule schedule

} choice\_list

} choice **optional**

}

**with** {

**variant** "XML":"element";

**variant** (resourceName) "XML":"attribute";

**variant** (choice) "XML":"untagged";

**variant** (choice.childResource\_list) "XML":"untagged";

**variant** (choice.choice\_list) "XML":"untagged";

**variant** (choice.choice\_list[-])"XML":"untagged";

**variant** (resourceName) "JSON":"escape as short" //just an example for a JSON  
 //-specific encoding instr.

/\* long -short name conversion XML\*/

**variant** "XML":"name as 'ae'";

**variant** (resourceName) "XML":"name as 'rn'";

**variant** (resourceType) "XML":"name as 'ty'";

**variant** (resourceID) "XML":"name as 'ri'";

**variant** (parentID) "XML":"name as 'pi'";

**variant** (creationTime) "XML":"name as 'ct'";

**variant** (lastModifiedTime) "XML":"name as 'lt'";

**variant** (choice.childResource\_list[-])"XML":"name as 'cr'";

**variant** (choice.choice\_list[-].container) "XML":"name as 'cnt'";

**variant** (choice.choice\_list[-].group\_) "XML":"name as 'grp'";

**variant** (choice.choice\_list[-].accessControlPolicy) "XML":"name as 'acp'";

**variant** (choice.choice\_list[-].subscription) "XML":"name as 'sub'";

**variant** (choice.choice\_list[-].pollingChannel) "XML":"name as 'pch'";

**variant** (choice.choice\_list[-].schedule) "XML":"name as 'sch'";

/\* long -short name conversion JSON\*/

**variant** "JSON":"name as 'ae'";

**variant** (resourceName) "JSON":"name as 'rn'";

**variant** (resourceType) "JSON":"name as 'ty'";

**variant** (resourceID) "JSON":"name as 'ri'";

**variant** (parentID) "JSON":"name as 'pi'";

**variant** (creationTime) "JSON":"name as 'ct'";

**variant** (lastModifiedTime) "JSON":"name as 'lt'";

**variant** (choice.childResource\_list[-])"JSON":"name as 'cr'";

**variant** (choice.choice\_list[-].container) "JSON":"name as 'cnt'";

**variant** (choice.choice\_list[-].group\_) "JSON":"name as 'grp'";

**variant** (choice.choice\_list[-].accessControlPolicy) "JSON":"name as 'acp'";

**variant** (choice.choice\_list[-].subscription) "JSON":"name as 'sub'";

**variant** (choice.choice\_list[-].pollingChannel) "JSON":"name as 'pch'";

**variant** (choice.choice\_list[-].schedule) "JSON":"name as 'sch'";

};

}

**with** {

**encode** "XML";

**variant** "XML":"namespace as 'http://www.onem2m.org/xml/protocols' prefix 'm2m'";

**variant** "XML""controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

**encode** "JSON"

}

### 2.3 Explicit encoding control for predefined functions

Add the optional charstring parameter encode\_type to the predefined functions encvalue, decvalue, encvalue\_unichar, decvalue\_unichar and decmatch. When present it shall override the session-default encoding type for the given function call. The string shall be one of the encode attribute strings associated with the type of the instance to be encoded/decoded.

NOTE: encvalue, decvalue, encvalue\_unichar & decvalue\_unichar have the optional encoding\_info/ decoding\_info parameters already. In principle, these could be used to select the actual encoding type. However, to my understanding these strings were meant to be passed to the codec, while the parameter encode\_type is used by the TE to invoke the right codec.

EXAMPLES

**Example**: Overriding the default encoding.

**module** Types {

**type** **record** AE

{

NCName resourceName **optional**,

ResourceType resourceType **optional**,

ID resourceID **optional**,

NhURI parentID **optional**,

Timestamp creationTime **optional**,

Timestamp lastModifiedTime **optional**,

**union** {

**record** **length**(1 .. **infinity**) of ChildResourceRef childResource\_list,

**record** **length**(1 .. **infinity**) of **union** {

Container container,

Group group\_,

AccessControlPolicy accessControlPolicy,

Subscription subscription,

PollingChannel pollingChannel,

Schedule schedule

} choice\_list

} choice **optional**

}

**with** {

**variant** "element";

**variant** (resourceName) "attribute";

**variant** (choice) "untagged";

**variant** (choice.childResource\_list) "untagged";

**variant** (choice.choice\_list) "untagged";

**variant** (choice.choice\_list[-]) "untagged";

};

}

**with** {

**encode** "XML";

**encode** "JSON";

**encode** "MyEncoding";

**variant** "XML":"namespace as 'http://www.onem2m.org/xml/protocols' prefix 'm2m'";

**variant** "XML":"controlNamespace 'http://www.w3.org/2001/XMLSchema-instance' prefix 'xsi'";

}

…

**template** AE t\_AE\_invoke := { “myResource”, int1, “1”, <https://www.example.org/parent>, “20160708T154559”}

**var** **bitstring** v\_encoded := **encvalue**(t\_AE\_invoke, -, “MyEncoding”);