ADDENDA

ANSI/ASHRAE Addendum bp to
ANSI/ASHRAE Standard 135-2016

A Data Communication Protocol for Building Automation and Control Networks

Approved by ASHRAE on June 15, 2018, and by the American National Standards Institute on June 15, 2018.

This addendum was approved by a Standing Standard Project Committee (SSPC) for which the Standards Committee has established a documented program for regular publication of addenda or revisions, including procedures for timely, documented, consensus action on requests for change to any part of the standard. The change submittal form, instructions, and deadlines may be obtained in electronic form from the ASHRAE® website (www.ashrae.org) or in paper form from the Senior Manager of Standards.

The latest edition of an ASHRAE Standard may be purchased on the ASHRAE website (www.ashrae.org) or from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: orders@ashrae.org. Fax: 678-539-2129. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada). For reprint permission, go to www.ashrae.org/permissions.

 ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus Standard developed under the auspices of ASHRAE. *Consensus* is defined by the American National Standards Institute (ANSI), of which ASHRAE is a member and which has approved this Standard as an ANS, as "substantial agreement reached by directly and materially affected interest categories. This signifies the concurrence of more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that an effort be made toward their resolution." Compliance with this Standard is voluntary until and unless a legal jurisdiction makes compliance mandatory through legislation.

ASHRAE obtains consensus through participation of its national and international members, associated societies, and public review.

ASHRAE Standards are prepared by a Project Committee appointed specifically for the purpose of writing the Standard. The Project Committee Chair and Vice-Chair must be members of ASHRAE; while other committee members may or may not be ASHRAE members, all must be technically qualified in the subject area of the Standard. Every effort is made to balance the concerned interests on all Project Committees.

The Senior Manager of Standards of ASHRAE should be contacted for

a. interpretation of the contents of this Standard,
b. participation in the next review of the Standard,
c. offering constructive criticism for improving the Standard, or
d. permission to reprint portions of the Standard.

[This foreword and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present changes to ANSI/ASHRAE Standard 135-2016. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The changes are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE Standard 135-2016 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are added, plain type is used throughout.

The use of placeholders like X, Y, Z, X1, X2, etc., should not be interpreted as literal values of the final standard. These placeholders will be assigned actual numbers/letters only with incorporation of this addendum into the standard for republication.

135-2016*bp*-1. Make rules for POST consistent with rules for PUT.

Rationale

W.29.1 lists the "computed metadata" that can't be PUT to the server, but W.28 does not make a similar prohibition against POSTing computed metadata.

[Change **Clause W.28**, p. 1228]

W.28 Creating Data

If the server allows it, new members of the collection types Collection, List, SequenceOf, and Array shall be creatable by POSTing a fully formed data to the path of the collection. *The server shall first check that the type of the provided data matches the target's member type and that none of the server-computed metadata, 'count', 'children', 'descendants', 'truncated', 'history', 'valueAge', 'etag', 'next', 'self', 'edit', 'failures', or 'id', are present. If these basic conditions are not met, a WS_ERR_VALUE_FORMAT error shall be indicated.*

. . .

135-2016*bp*-2. Make 'type' consistent at all levels and introduce 'effectiveType'.

Rationale

The requirement in W.15.4 that the 'type' change meaning at the top level does not fully support data that is defined to be an "Any". Instances that replace a defined "Any" need two "types". Their "effective type" or "defined type" is specified by Clause W.15.4 to be the location in a named definition where the Any is declared. But when that Any is replaced at run-time, it also has a "given type" also known as "actual type" or "runtime type." Both the "effective type" and the "given type" are valid and the total set of metadata for the data item is the combination of the metadata from the two types.

The problem is that, as currently required by W.15.4, the protocol only returns one 'type', and that is specified to be the "effective type". So the "given type" for an Any is masked and the client cannot determine how to properly consume or understand the instance data.

Note that this problem applies only to 'Any' members of a Sequence, Composition, or Object. It is not a problem for collections (Collection/Array/List/SequenceOf) of 'Anys'. This is because there is no "effective type" in that case. A definition for a collection of 'Any' can't specify any metadata for its members; therefore, it is not augmenting the member's given type and there is no conflict (the effective type is the same as the given type in this case, and there is no problem with the published standard).

It is critical for clients to be able to decode data, especially JSON encodings where the $base is not always included and therefore the client needs to know the type definition in order to know the base types.

For example, given the following definition,

```
<Definitions>
   <Composition name="555-Outer">
      <String name="innerString" displayName="InnerString"/>
      <Any   name="innerAny" displayName="InnerAny"/>
   </Composition>
</Definitions>
```

If the client asks for an instance of "555-Outer", there is no problem because the "given type" of the inner is conveyed by the 'type' attribute and the "effective type" of "555-Outer/innerAny" is implied by its containment.

```
GET /path/to/outer

<Composition name="outer" type="555-outer">
   <String      name="innerString" value="…"/>
   <Sequence name="innerAny" type="0-BACnetAddress">…</Sequence>
</Composition>
```

However, if the client drills down to the Any, there is a problem since the standard currently requires the 'type' on the top level item be the "effective type", which masks the runtime type of the data and the client no longer knows that the data is an instance of "0-BACnetAddress".

```
GET /path/to/inner

<Sequence name="inner" type="555-Outer/inner">…</Sequence>
```

The problem is solved by providing the given 'type' and the effective type as defined by Clause W.15.4 in a separate 'effectiveType' metadata. This makes 'type' consistent at all levels, as its meaning does not change for the top level only.

```
GET /path/to/inner
```

```
<Sequence name="inner" effectiveType="555-Outer/inner" type="0-BACnetAddress" >…</Sequence>
```

[Insert new **Clause W.15.X**, p. 1219]

W.15.X The 'type' Metadata

If the data is an instance of a named definition, then the 'type' metadata shall be the name of that definition. This shall be present at the top level in response or anywhere that it cannot be derived by context. For example, if the defined type of a data item is 'Any' then an instance of that will need a 'type' metadata to indicate the actual instance type.

[Change **Clause W.15.4**, p. 1219]

W.15.4 The ~~'type'~~ *'effectiveType'* Metadata

Since the server returns only the metadata that is different from its definition, the client needs to know that definition in order to infer the elided metadata values. If the top level response data is a direct instance of a named type, then that is easy and the 'type' metadata directly specifies that named type. But if the client has "drilled down" into an instance using a URI path that references data that is only a part of a larger named type, then the server needs to reflect that information in the ~~'type'~~ *'effectiveType'* metadata so the client can know what higher level named definition the referenced data was instantiated from. Therefore, for the top level data item in a response:

(a) If the data item is a direct instance of a defined type, then the 'type' metadata on the data item shall simply be the name of that type definition. This includes members of collections that use 'memberType' since no other metadata can be defined for the members.
(b) If the data item is a part of an instance of larger data definition (it has an anonymous type), then the ~~'type'~~ *'effectiveType'* metadata shall be the name of the outermost applicable definition plus a slash-separated path down to that data item. This includes members of collections that use 'memberTypeDefinition', since additional metadata can be defined by that construct.

Therefore, given a definition of:

. . .

For an instance of "B" at /b, the server shall return the ~~'type'~~ *'effectiveType'* metadata, and the client shall infer the effective value of the 'comment' metadata, for the following paths:

/b/$*effectiveType*~~type~~ = "B"	/b/$comment = "This is a B"
/b/a1/$*effectiveType*~~type~~ = "B/a1"	/b/a1/$comment = "comment on B/a1"
/b/a2/$*effectiveType*~~type~~ = "B/a2"	/b/a2/$comment = ""
/b/a3/$*effectiveType*~~type~~ = "B/a3"	/b/a3/$comment = "comment on A/a3"
/b/b1/$*effectiveType*~~type~~ = "B/b1"	/b/b1/$comment = ""
/b/b1/a1/$*effectiveType*~~type~~= "B/b1/a1"	/b/b1/a1/$comment = "comment on B/a1"
/b/b1/a2/$*effectiveType*~~type~~= "B/b1/a2"	/b/b1/a2/$comment = "comment on B/b1/a2"
/b/b1/a3/$*effectiveType*~~type~~= "B/b1/a3"	/b/b1/a3/$comment = "comment on A/a3"
/b/b2/$*effectiveType*~~type~~ = "B/b2"	/b/b2/$comment = ""
/b/b2/{n}/$*effectiveType*~~type~~= "A"	/b/b2/{n}/$comment = "This is an A"
/b/b2/{n}/a1/$*effectiveType*~~type~~ = "A/a1"	/b/b2/{n}/a1/$comment = ""
/b/b2/{n}/a2/$*effectiveType*~~type~~ = "A/a2"	/b/b2/{n}/a2/$comment = ""
/b/b2/{n}/a3/$*effectiveType*~~type~~ = "A/a3"	/b/b2/{n}/a3/$comment = "comment on A/a3"
/b/b3/$*effectiveType*~~type~~ = "B/b3"	/b/b3/$comment = ""
/b/b3/{n}/$*effectiveType*~~type~~ = "B/b3/$memberTypeDefinition/1"	/b/b3/{n}/$comment = "an anon extension of A"
/b/b3/{n}/a1/$*effectiveType*~~type~~ = "B/b3/$memberTypeDefinition/1/a1"	/b/b2/{n}/a1/$comment = ""
/b/b3/{n}/a2/$*effectiveType*~~type~~ = "B/b3/$memberTypeDefinition/1/a2"	/b/b3/{n}/a2/$comment = "comment on B/b3/?/a2"
/b/b3/{n}/a3/$*effectiveType*~~type~~ = "B/b3/$memberTypeDefinition/1/a3"	/b/b2/{n}/a3/$comment = "comment on A/a3"
/b/b4/$*effectiveType*~~type~~ = "B/b4"	/b/b4/$comment = ""
/b/b4/r/$*effectiveType*~~type~~ = "B/b4/$choices/r"	/b/b4/r/$comment = "comment on B/b4/r"

[Change **Clause Y.4.3**, p. 1290]

Y.4.3 'type'

This optional metadata item, of type String, indicates the type of a data item when that item is an instance of a previously defined type. If the data item has a defined type, then the type of that data can only be changed to a type that is an extension of the defined type, unless the defined type is Any, in which case the new type is limited to the types, or extensions thereof, allowed by the definition of the Any. See Clause ~~W.15.4~~ *W.15.X* for more information on the type metadata item.

[Insert new **Clause Y.4.X**, p. 1290]

Y.4.X 'effectiveType'

This optional metadata item, of type String, indicates the type of a data item when that item is only a portion of a defined type. This can be used in contexts where the outer container for the data is missing and thus the context of the defining type name. See Clause W.15.4 for more information on the 'effectiveType' metadata item.

[Change **Clause Q.2.1**, p. 1145]

Q.2.1 <CSML>

When used in a file context, the XML syntax defined by this annex is enclosed in the element <CSML> ("Control Systems Modeling Language") that has an xml namespace of ~~"http://www.bacnet.org/CSML/1.2"~~ *"http://www.bacnet.org/CSML/1.3"*.

135-2016*bp*-3. Fully specify the behavior of "includes".

Rationale

It is not fully specified how <Includes> and $$includes work.

[Change **Clause Q.2.1.4**, p. 1146]

Q.2.1.4 <Includes>

This optional child element of <CSML> provides one or more "include directives" instructing the consumer to include all of the information contained in other CSML file(s). The valid child elements of <Includes> are <Link> elements. There can be multiple <Includes> elements under a <CSML> element and these may appear in any position. *The <Includes> element is structurally equivalent to a <List>. The rules for evaluating Link paths are the same as for the JSON $$includes member defined in Z.2.4.*

Data item(s) in the target file are included as if they existed in the place of the <includes> directive, so the directive can appear anywhere in a file. However, the included items are unordered because <CSML> is defined to be a Collection, which is unordered, and the <Includes> is defined to be a List of Links, which is also unordered. Therefore, <Includes> cannot be used inside a Sequence base type or in situations with implied ordering behavior like the auto-assignment of Enumerated 'namedValues' metadata.

If the included file has a <CSML> wrapper, the menbers of that collection are included separately.

For example if the included file has a CSML wrapper, then

```
<Collection>
  <String name="bob" value="Bob"/>
  <String name="carol" value="Carol"/>
  <String name="ted" value="Ted"/>
  <String name="alice" value="Alice"/>
</Collection>
```

and

```
<Collection>
  <String name="alice" value="Alice"/>
  <Includes><Link value="http://host/include-some.xml"/></Includes>
  <String name="bob" value="Bob"/>
</Collection>
```

produce the same logical result if the included file is either:

```
<?xml...>
<CSML...>
  <String name="carol" value="Carol"/>
  <String name="ted" value="Ted"/>
</CSML>
```

or

```
<?xml...>
<CSML...>
  <String name="ted" value="Ted"/>
  <String name="carol" value="Carol"/>
</CSML>
```

If the included file does not have a CSML wrapper, then

```
<Collection>
  <String name="bob" value="Bob"/>
  <String name="carol" value="Carol"/>
  <String name="ted" value="Ted"/>
```

<String name="alice" value="Alice"/>
</Collection>
and
<Collection>
<String name="alice" value="Alice"/>
<Includes><Link value="http://host/include-some.xml"/></Includes>
<String name="ted" value="Ted"/>
<String name="bob" value="Bob"/>
</Collection>
produce the same logical result if the included file is:
<?xml...>
<String name="carol" value="Carol" …>

[Change **Clause Z.2.4**, p. 1343]

Z.2.4 "$$includes"
This optional member of a CSML object, of base type "List of Link", provides a list of directives to include other CSML documents. The value restrictions for the Links follow the same rules defined in Clause Q.2.1.4 for XML includes. Consumers of JSON documents are only required to include other JSON documents. *The rules for including data are the same as for the XML <Includes> element defined in Clause Q.2.1.4*

135-2016*bp*-4. Remove the path syntax from the 'select' query parameter.

Rationale

The standard grammar for the 'select' query parameter allows for "paths" but it did not also allow for wildcards on those paths. Allowing paths without wildcards proves to be impractical and not useful. Rather than introduce wildcards, this addendum simply removes the path option since the complexity of implementation was not outweighed by the benefit, given that the standard allows the use of /.multi for retrieving multiple non-contiguous data items.

[Change **Clause W.14**, p. 1217]

W.14 Selecting Children

The selection of children to be returned for constructed base types can be controlled with the query parameter 'select'. Selecting children that do not exist is not considered an error and shall be ignored.

The syntax for the selection expression is defined by the following grammar. Whitespace (space or tab in any combination) is not shown in this grammar definition. Whitespace is allowed between any of the items of the grammar and shall be ignored by the server device.

selectQuery = "select=" selectClause

selectClause = ~~selectPath~~ *selectName* [";" selectClause]

~~selectPath = selectName / ".required" / ".optional" ["/" selectPath]~~

selectName = *".required" / ".optional"* / {string matching the name of the data item}

135-2016*bp*-5. Resolve conflicting statements about configuring external authorization servers.

Rationale

The opening sentence of Clause W.3.3.1 conflicts with item (d) in that same Clause.

[Change **Clause W.3.3.1**, p. 1193]

W.3.3.1 Factory Default Condition

. . .

While in the default condition, the configuration of other usernames or passwords ~~or the writing of any external authorization server information~~ shall be forbidden. Therefore, the only security configuration operations that are allowed in this condition are:

(a) to set the base username and password at "/.auth/int/user" and "/.auth/int/pass" to non-default values,
(b) to set the base client's id and secret at "/.auth/int/id" and "/.auth/int/secret" to non-default values,
(c) to set the server UUID at "/.auth/dev-uuid" if not factory preset and fixed, or
(d) to configure the external authorization server information and then disable the internal server by writing "false" to "/.auth/int/enable".

135-2016*bp*-6. Remove incorrect table for callback formats.

Rationale

Table W-11 is a vestige of earlier drafts of the standard. It is no longer valid in the published standard and does not match the textual description of callbacks or the examples.

[Delete **Table W-11**, p. 1235]

~~**Table W-11.** Callback Format~~

~~Path~~	~~Type~~	~~Description~~
~~/~~	~~Composition~~	~~The POSTed callback data container~~
~~/$subscription~~	~~Link~~	~~A Link to the subscription~~
~~/covs~~	~~List~~	~~A list of COV values~~
~~/covs/{n}~~	~~Composition~~	~~A single COV value~~
~~/covs/{n}/path~~	~~String~~	~~The path to the data~~
~~/covs/{n}/value~~	~~Any~~	~~The value of the data~~
~~/logs~~	~~List~~	~~The list of log specifications~~
~~/logs/{n}~~	~~Composition~~	~~A single log specification~~
~~/logs/{n}/path~~	~~String~~	~~The path to the log buffer~~
~~/logs/{n}/records~~	~~List~~	~~The list of trend records~~
~~/logs/{n}/records/{x}~~	~~Sequence~~	~~a record of type "0-BACnetLogRecord"~~

135-2016*bp*-7. Allow plain text POSTs for primitive data.

Rationale

Plain text POSTs should not be forbidden.

Clause W.3.1.4 says:

> While in the factory defaults mode, the authority certificate(s) shall be writable in plain text, each using an HTTP POST to the List at "/.auth/ca-certs-pend"

But W.9 says:

> The alt=plain format applies to GET and PUT operations only. Other methods shall generate a WS_ERR_BAD_METHOD error response.

There is no reason to forbid POSTs for collections that are defined to be of primitive data types

[Change **Clause W.9**, p. 1209]

W.9 Representation of Data

. . .

For the format alt=plain, the representation of primitive value data is to return or accept the value in plain text. The representation of other base types in plain text is not defined and shall generate a WS_ERR_NOT_REPRESENTABLE error response. The HTTP Content-Type shall be set to "text/plain". The text is placed directly in the body of the HTTP request or response, and, since escaping and quoting are not necessary, they shall not be used. The textual format shall be the same as the 'value' attribute of the XML format, with the exception that for Date, Time, and DateTime base types, the textual format when 'unspecifiedValue' is true shall be "----/--/--", "--:--:--" and "----/--/--T--:--:--Z", respectively. The alt=plain format applies to GET and PUT operations *on primitive base types and for POST operations where the 'memberType' is defined to be a primitive.* ~~only.~~ Other methods shall generate a WS_ERR_BAD_METHOD error response, *and other base types shall generate a WS_ERR_VALUE_FORMAT error response.*

. . .

135-2016*bp*-8. Allow extended error numbers.

Rationale

There is no provision for proprietary error numbers. It is not desirable to foresee every error condition to use WS_ERR_OTHER.

[Change **Clause W.40**, p. 1240]

W.40 Errors

To report errors to the client, the server shall use an appropriate HTTP status code with a Content-Type of "text/plain" and a response body containing text that is both machine readable and human readable.

The body of an error response shall consist of at least one line of text. Other lines of text can be provided at the server's option. The format of the first line of text shall normally consist of a question mark followed by a space followed by a decimal error number, defined by the table in this clause *or vendor-defined*, followed by a human readable text to provide details. The content of the human readable text is a local matter; however, it is recommended that it be appropriate to the locale into which the server device is deployed.

. . .

Vendor-defined error numbers for conditions beyond those specified by this standard shall be greater than or equal to 1000 and the selection of the corresponding HTTP status codes is a local matter but shall be appropriate to the kind of error detected.

135-2016*bp*-9. Add new error numbers.

Rationale

Several error conditions have been identified that would benefit from standard error numbers.

[Add to **Table W-14**, p. 1241]

Table W-14. Error Numbers

Error Name	Error Number	HTTP Status Code	Example Error Text
WS_ERR_OTHER	0	500	"Other error"
. . .	. . .	. . .	. . .
WS_ERR_CANNOT_FOLLOW	46	404	"Cannot follow reference to data"
WS_ERR_FUNCTION_NAME	*47*	*403*	*"Invalid Function Name"*
WS_ERR_FUNCTION_TARGET	*48*	*403*	*"Invalid target data for function"*
WS_ERR_ ARG_BAD_SYNTAX	*49*	*400*	*"Bad syntax for function argument"*
WS_ERR_ARG_NOT_SUPPORTED	*50*	*403*	*"Function argument not supported"*
WS_ERR_ARG_VALUE_FORMAT	*51*	*403*	*"Function argument value format"*
WS_ERR_ARG_OUT_OF_RANGE	*52*	*403*	*"Function argument out of range"*
WS_ERR_TARGET_DATATYPE	*53*	*403*	*"Incompatible target datatype"*
WS_ERR_CANNOT_HAVE_CHILDREN	*54*	*403*	*"Target cannot have children"*
WS_ERR_CANNOT_HAVE_VALUE	*55*	*403*	*"Target cannot have value"*
WS_ERR_OAUTH_INVALID_REQUEST	*56*	*403*	*"Invalid OAuth request"*
WS_ERR_OAUTH_INVALID_TOKEN	*57*	*401*	*"Invalid OAuth token"*
WS_ERR_INSUFFICIENT_SCOPE	*58*	*401*	*"Insufficient authorization scope"*
WS_ERR_CALLBACK_FAILED	*59*	*500*	*"Callback failed"*
WS_ERR_CLIENT_ACTION_FAILED	*60*	*500*	*"Client action failed"*

135-2016*bp*-10. Add formal definition for JSON equivalent to XML's <CSML>.

Rationale

XML files have a capability that JSON files lack: a <CSML> wrapper. With this wrapper absent, an XML file represents a single data item, but with the wrapper present, an XML file represents zero or more data items and can contain only a <Definitions> section with no data items at all. But a JSON file is assumed only to be a single data item, with no capability to define a "wrapper," therefore, it can't contain only a $$definitions section. Also, when including a JSON file, the include processor doesn't know when to include the single top level object or include the members individually the way it would when including the <CSML> Collection.

Therefore, this addendum defines the reserved name ".csml" to indicate such a wrapper. If the outer JSON object has "$name":".csml", then it behaves exactly as its <CSML> counterpart - it is assumed to be a Collection of zero or more data items, and therefore can have only a $$definitions and/or $$tagDefinitions section(s) and no other data items, and the members are included individually and the outer JSON object disappears the same way the <CSML> wrapper Collection does.

[Change **Clause Z.2**, p. 1341]

Z.2 JSON Document Structure
In JSON, the top level object is anonymous. If required by context, e.g., in the body of a POST where a new name is being proposed, an explicit "$name" can be included to provide a name for the top level data item. This top level object can have an optional member named "$$defaultLocale" (see Clause Z.2.1).

When used in a file context, the top level object shall be an Annex Y data item of base type Collection *and shall have a "$name" of ".csml"*. This top level object is referred to as the "CSML object" (Control Systems Modeling Language) *and equivalent to the <CSML> Collection defined in Clause Q.2.1*. This object can have an optional "$$definitions" member (Clause Z.2.2), an optional "$$tagDefinitions" member (Clause Z.2.3), an optional "$$includes" member (Clause Z.2.4), and any number and combination of members allowed by the Collection base type. Since this top level object in a file is a Collection, the metadata "published", "author", "description", "dataRev", etc., can be used to provide helpful information to consumers..

. . .

135-2016*bp*-11. Specify 'name' safety check for setting data.

Rationale

There is no explicit restriction in the 'name' of data that is PUT. This can be dangerous if the client is putting data that has a different name from the target item since this is likely an error and needs to be rejected.

[Change **Clause W.29**, p. 1229]

W.29 Setting Data

. . .

Upon receipt of a properly authorized PUT, the server shall recursively update the values, metadata, and children in the target data with the values, metadata, and children in the provided data, according to the rules in the following clause. When a rule says to "recurse on" an item, that item becomes the new "target" and the rules are followed again, descending as needed to process all the provided data.

If the provided data has a name and that name is not equal to ".anonymous", then that name shall match the target data item's name. If it does not match, the server shall return a WS_ERR_INCONSISTENT_VALUES error response.

. . .

135-2016*bp*-12. Specify how to evaluate relative paths for collections of links.

Rationale

The $descendants and $failures metadata are defined to return "relative links". It is clear from the standard textual description what the links are relative to, but it is not possible for an automated processor for the $target alias to know what the links are to be relative to without hardcoded rules.

Therefore, the standard needs to state that relative paths in Links are evaluated relative to the parent of the Link, unless the parent is a collection (Array/List/Collection/SequenceOf) of Links, in which case, the paths are evaluated relative to the grandparent of the Link. This works because $descendants and $failure are defined to have a $memberType of Link, so that that layer of "collection of links" gets skipped over when evaluating relative links. However, if Links are placed on a collection (Array / List / Collection / SequenceOf) that does not have a $memberType of Link, then the relative links are evaluated relative to the collection itself. This would be expected behavior, since putting links on the collection base types is a legitimate thing to do - it's only a "collections of links" layer that gets skipped. Other kinds of lists of pointers like /.data/objects or /.data/nodes/point don't have this problem because they contain absolute links.

[Change **Clause Y.4.46**, p. 1301]

Y.4.46 'target'

The 'target' metadata is an alias for the data that is being pointed at by the values of a Link. This metadata is normally not present in serialized contexts. If applied to a non-Link base type, the 'target' metadata simply refers to the data item itself, i.e., every data item implicitly "targets" itself if it is not actually a link to some other data. This metadata only has practical use in contexts and operations (such as web services) where it is desirable to "traverse" through links to the referent data, e.g., GET /path/to/link/$target/child/of/target.

When applied to a Link base and the value contains a relative path, the relative path is evaluated relative to the parent of the Link, unless that parent has a 'memberType' of "Link", in which case the relative path is evaluated relative to the grandparent of the Link. Thus, the evaluation of relative paths "skips over" collections of links, such as 'descendants' or 'failures' to find the basis for evaluation of the relative path.

135-2016*bp*-13. Allow proprietary categories for the 'metadata' query.

Rationale

There is an implied prohibition against proprietary categories for the 'metadata' query parameter.

[Change **Clause W.15.2**, p. 1218]

W.15.2 Enhanced Content

. . .

The *standard* category identifiers ~~allowed~~ for the 'metadata' query parameter are defined by the following table:

. . .

[Add a new entry to **History of Revisions**, p. 1364]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

…	…	**…**
1	20	**Addendum *bp* to ANSI/ASHRAE 135-2016** Approved by ASHRAE on June 15, 2018; and by the American National Standards Institute on June 15, 2018. 1. Make rules for POST consistent with rules for PUT 2. Make 'type' consistent at all levels and introduce 'effectiveType' 3. Fully specify the behavior of "includes" 4. Remove the path syntax from the 'select' query parameter 5. Resolve conflicting statements about configuring external authorization servers 6. Remove incorrect table for callback formats 7. Allow plain text POSTs for primitive data 8. Allow extended error numbers 9. Add new error numbers 10. Add formal definition for JSON equivalent to XML's <CSML> 11. Specify 'name' safety check for setting data 12. Specify how to evaluate relative paths for collections of links 13. Allow proprietary categories for the 'metadata' query parameter

POLICY STATEMENT DEFINING ASHRAE'S CONCERN FOR THE ENVIRONMENTAL IMPACT OF ITS ACTIVITIES

ASHRAE is concerned with the impact of its members' activities on both the indoor and outdoor environment. ASHRAE's members will strive to minimize any possible deleterious effect on the indoor and outdoor environment of the systems and components in their responsibility while maximizing the beneficial effects these systems provide, consistent with accepted Standards and the practical state of the art.

ASHRAE's short-range goal is to ensure that the systems and components within its scope do not impact the indoor and outdoor environment to a greater extent than specified by the Standards and Guidelines as established by itself and other responsible bodies.

As an ongoing goal, ASHRAE will, through its Standards Committee and extensive Technical Committee structure, continue to generate up-to-date Standards and Guidelines where appropriate and adopt, recommend, and promote those new and revised Standards developed by other responsible organizations.

Through its *Handbook*, appropriate chapters will contain up-to-date Standards and design considerations as the material is systematically revised.

ASHRAE will take the lead with respect to dissemination of environmental information of its primary interest and will seek out and disseminate information from other responsible organizations that is pertinent, as guides to updating Standards and Guidelines.

The effects of the design and selection of equipment and systems will be considered within the scope of the system's intended use and expected misuse. The disposal of hazardous materials, if any, will also be considered.

ASHRAE's primary concern for environmental impact will be at the site where equipment within ASHRAE's scope operates. However, energy source selection and the possible environmental impact due to the energy source and energy transportation will be considered where possible. Recommendations concerning energy source selection should be made by its members.

ASHRAE · 1791 Tullie Circle NE · Atlanta, GA 30329 · www.ashrae.org

About ASHRAE

ASHRAE, founded in 1894, is a global society advancing human well-being through sustainable technology for the built environment. The Society and its members focus on building systems, energy efficiency, indoor air quality, refrigeration, and sustainability. Through research, Standards writing, publishing, certification and continuing education, ASHRAE shapes tomorrow's built environment today.

For more information or to become a member of ASHRAE, visit www.ashrae.org.

To stay current with this and other ASHRAE Standards and Guidelines, visit www.ashrae.org/standards.

Visit the ASHRAE Bookstore

ASHRAE offers its Standards and Guidelines in print, as immediately downloadable PDFs, on CD-ROM, and via ASHRAE Digital Collections, which provides online access with automatic updates as well as historical versions of publications. Selected Standards and Guidelines are also offered in redline versions that indicate the changes made between the active Standard or Guideline and its previous version. For more information, visit the Standards and Guidelines section of the ASHRAE Bookstore at www.ashrae.org/bookstore.

IMPORTANT NOTICES ABOUT THIS STANDARD

To ensure that you have all of the approved addenda, errata, and interpretations for this Standard, visit www.ashrae.org/standards to download them free of charge.

Addenda, errata, and interpretations for ASHRAE Standards and Guidelines are no longer distributed with copies of the Standards and Guidelines. ASHRAE provides these addenda, errata, and interpretations only in electronic form to promote more sustainable use of resources.

6/18