Document Identifier: DSP0271

Date: 2016-10-13

Version: 0.5.6

YANG to Redfish Mapping Specification

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of the DMTF or its members. Because this document is a Work in Progress, this document may still change, perhaps profoundly and without notice. This document is available for public review and comment until superseded.

Provide any comments through the DMTF Feedback Portal:
http://www.dmtf.org/standards/feedback

Supersedes: None

Document Class: Normative

Document Status: Work in Progress

Document Language: en-US

CONTENTS

116

117 # Tables

Foreword

The *YANG to Redfish Mapping Specification* was prepared by the Chinook Technical Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Mrittika Ganguli – Intel Corporation
- Ajay Gummadi - Microsoft
- Mike Lazar – Dell, Inc
- John Leung – Intel Corporation
- Peter Mellquist – Hewlett Packard Enterprise
- Michael Pizzo - Microsoft
- Rohan Sen - VMware
- Joseph White – Dell, Inc

Introduction

The information in this specification should be sufficient to convert a YANG model to a file which adheres to the Common Schema Data Language (CSDL) format. CSDL is one of the formats that Redfish's uses to describe schema and is described in OASIS OData specification (odata.org). The conversion can be done manually or programmatically.

YANG to Redfish Mapping Specification

1 Scope

The *YANG to Redfish Mapping Specification* describes how to map a YANG model to a Redfish model, specifically, the mapping to YANG RFCs to Redfish CSDLs.

The mapping should be universal enough to convert any YANG model. This will allow network devices to be managed via the Redfish RESTful interface, regardless of the YANG model they support.

The specification uses IETF RFC 6020 as the description of the YANG model elements. The specification uses examples from DHCP for usages of the YANG model elements.

This document describes a mapping translation. The goal is for completeness. However, there may be YANG model elements and constructs beyond RFC 6020 which may need to be added.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

DMTF DSP0266, "Redfish Scalable Platforms Management API Specification", http://www.dmtf.org/standards/redfish

IETF RFC 6020, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)" https://tools.ietf.org/html/rfc6020

ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

OASIS OData v4.0, https://www.oasis-open.org/standards#odatav4.0

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Annex H. The terms in parentheses are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The terms defined in DSP0004, DSP0223, and DSP1001 apply to this document. The following additional terms are used in this document.

4 Symbols and abbreviated terms

The abbreviations defined in DSP0004, DSP0223, and DSP1001 apply to this document. The following additional abbreviations are used in this document.

5 Description

This YANG to Redfish Mapping Guidelines document describes how to map YANG statements into Redfish OData CSDL constructs.

5.1 YANG

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications. YANG is used to model the operations and content layers of NETCONF.

Various SDO have YANG RFCs for various network capabilities.

A YANG RFC includes a YANG depiction of the model (tree diagram) and YANG code (or schema). The YANG code is consider more normative than the YANG depiction.

The YANG depiction gives a high level view of the model's construct. Below is a fragment of the depiction from the DHCP Draft.

```
      +--rw interfaces
      |  +--rw interface* [name]
      |     +--rw name                        string
      |     +--rw description?                string
      |     +--rw type                        identityref
      |     +--rw enabled?                    boolean
  . . .
```

The YANG code specifies the schema associated with the YANG depiction. The YANG code is bracketed by <CODE BEGINS> and <CODE ENDS) delimiters. Below is a fragment of the YANG code.

```
<CODE BEGINS>
module ietf-interfaces {

     namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
     prefix if;

     import ietf-yang-types {
       prefix yang;
     }

     organization
       "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

     Contact "…";
```

```
     container interfaces {
        description "Interface configuration parameters.";
        . . .
<CODE ENDS>
```

5.2 Redfish

The Redfish Scalable Platforms Management API ("Redfish") is a new specification that uses RESTful interface semantics to access data defined in model format to perform systems management. It is suitable for a wide range of servers, from stand-alone servers to rack mount and bladed environments but scales equally well for large scale cloud environments.

RESTful interface specified by:

- A URI path to resource
- The content of the resource are described in an OData schema (CSDL) and json-schema

5.3 Differences between YANG and Redfish CSDL

There are basic differences between YANG and Redfish CSDL which are evident throughout. Table 1 contains systemic differences between YANG RFCs and Redfish CSDL. The table includes the decision made for mapping purposes.

Table 1 – Differences between YANG and Redfish

YANG	Redfish JSON and CSDL	Mapping Decision
Names contain "-" (dashes)	OData does not allow dashes	Convert dashes to "_" underscore, when used in an identifier
Names contain ":" (colons)	OData does not allow colons	Convert colon to "." (period), when used in an identifier
Names are generally Camel case, but exceptions exist	Names are Pascal case	Use YANG naming
Some names are concatenations (e.g. dhcp/relay/dhcpRelayServerGroups)	Prefers shorter names (e.g. dhcp/relay/ServerGroups)	Use YANG naming
Container names are plural	URI uses plural forms (Systems), but CSDL use singular form ("SystemCollection")	Use YANG naming
YANG has implicit scoping based on containment	CSDL has explicit scoping based on namespaces	Synthesize names to retain YANG scoping
Containers may contain no leafs/properties	"Resources should contain properties (otherwise, consider eliminating resource)"	Include resources without properties
List nodes may have leafs/properties	Resource.Collections don't have properties	Place properties in a subordinate-resource
Reference statements are not normative	LongDescription properties contain normative text	Place LongDescription at the module level which normatively refers to the RFC

5.3.1 Other mapping decisions

These general decisions were also followed for the mapping the YANG models:

- Map RFCs as-is. Suppress the desire to optimize for CSDL
- Define everything in the schema and don't worry about feature exposure exclusion
- A YANG module will correspond to an entity type at the top level
- Treat YANG some statements as a pre-processor style directive (e.g., uses, grouping)

5.3.2 YANG namespace

Preserve the YANG naming, including case and spelling (e.g., module, node structure).

The above rule strays from the Redfish's Pascal-case capitalization convention, since most YANG RFCs use camel-case. The deviation is necessary to allow the YANG community to understand the resultant mapping collateral.

5.3.3 Synthesized names for CSDL

Some model translations will require synthesized names for intermediate objects in the CSDL version. The intent is to create a translated mapping such that the resulting derived schema and JSON message match what would be expected from reading the YANG model directly.

5.3.4 OData annotations

Liberal use of CSDL Annotations to encapsulate YANG model information.

For each YANG statement, an annotation shall exist which retains the value from the original YANG statement. For example, the *default* statement results in an annotation of Term="Redfish.Yang.default" and whose String attribute have the value of the <default value>, "enable".

```
default: "enable"
```

```
<Annotation Term="Redfish.Yang.default" String="enable"/>
```

If a YANG statement is specifies a YANG node, an annotation is added which specifies the type of node which the YANG statement specifies. YANG nodes exist for *module*, *submodule, container*, and *list*. For example, the following module statement results in the following annotation in the CSDL

```
module: ietf-system
```

```
<Annotation Term="Redfish.Yang.NodeName" String="ietf_system" >
     <Annotation Term="Redfish.Yang.NodeType"
        EnumMember ="Redfish.Yang.NodeTypes/module"/>
/Annotation>
```

If the value of YANG statement has double quotes, then the CSDL escaping rules should be follow in creating the annotation string.

5.4 Redfish resource URI

The resource which represents the YANG model is attached to the instance of the NetworkDevice. Because of the abundance of YANG definitions, the resource name is constructed from the organization and the module name.

```
./NetworkDevices/{id}/<org>_<module-name>
```

The resource name is "ietf_interface" for IETF RFC 7317 (System) as shown below.

```
./NetworkDevices/{id}/ietf_system
```

An example mockup of the NetworkDevice singleton resource is shown below. The properties for DHCP, DNS and interfaces are shown.

```
{
    "@Redfish.Copyright": "",

    "@odata.context": "/redfish/v1/$metadata#NetworkDevices/Members/$entity",
    "@odata.type": "#NetworkDevice.v1_0_0.NetworkDevice",
    "@odata.id": "/redfish/v1/NetworkDevices/SW_15",
    "Id": "SW_15",

    "Name": "Ethernet Switch",
    "Manufacturer": "Manufacturer Name",
    "Model": "Model Name",
    "SKU": "67B",
    "SerialNumber": "2M220100SL",
    "PartNumber": "76-88883",

    "Dhcp": { "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp" },
    "Dns":  { "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dns" },
    "Interfaces": {"@odata.id":"/redfish/v1/NetworkDevices/SW_15/ietf_interfaces"},

    "Links": {
     "Chassis": [{
        "@odata.id": "/redfish/v1/Chassis/NetworkDeviceChassis_1"
     }],
     "ManagedBy": [{
        "@odata.id": "/redfish/v1/Managers/NetworkDeviceManager_1"
     }]
    },

 "Actions": {
    "#NetworkDevice.Reset": {
        "target": "/redfish/v1/NetworkDevices/SW_15/Actions/NetworkDevice.Reset",
        "ResetType@Redfish.AllowableValues": [
           "On",
           "ForceOff",
           "GracefulShutdown",
           "ForceRestart"
        ]
    }
  }
}
```

6 YANG statement mapping format

This clause describes how the mapping is formatted which the remainder of this document.

The clauses follow the ordering from RFC6020. For each YANG statement, the clause will contain the three sub-clauses

- Mapping YANG Depiction to Redfish Mockup
- Mapping YANG code to Redfish CSDL
- Statement Mapping Table

The "Mapping YANG Depiction to Redfish Mockup" clause shows an example of how the YANG depiction would look as a Redfish mockup, if the mapping rules are followed. The Redfish mockup shows what the end-user will see, without looking at the schema. If a statement does not have a depiction, then this section may not exist.

The "Mapping YANG code to Redfish CSDL" specifies a mapping ruleset to convert YANG code to a model with adheres to the Redfish specification.

The Statement Mapping table contains the mapping rules for the statement and each allowable sub-statement. These tables are heavily cross-referenced. There are sub-sections for sub-statements for which additional text is beneficial to understanding the mapping.

Table 2 shows the set of YANG statements that will to be mapped in Redfish CSDL. The ordering of these statements mirrored the ordering in RFC6020.

Note: Uses and grouping statement should be resolved during the translation. Annotations as still added to retain the notion of uses/grouping relationship. The text in the Description column are taken from RFC6020.

Table 2 - YANG statements

YANG	Description	Details
module	The "module" statement defines the module's name, and groups all statements that belong to the module together.	Clause 6.1
submodule	The "submodule" statement defines the submodule's name, and groups all statements that belong to the submodule together.	Clause 6.2
typedef	The "typedef" statement defines a new type that may be used locally in the module, in modules or submodules which include it, and by other modules that import from it.	Clause 6.3
type	The "type" statement takes as an argument a string that is the name of a YANG built-in type or a derived type, followed by an optional block of sub-statements that are used to put further restrictions on the type.	Clause 6.4
container	The "container" statement is used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).	Clause 6.5
leaf	The "leaf" statement contains simple data like an integer or a string. It has exactly one value of a particular type and no child nodes.	Clause 6.6
leaf-list	The "leaf-list" is a sequence of leaf nodes with exactly one value of a particular type per leaf.	Clause 6.7
list	The "list" statement defines a sequence of list entries.	Clause 6.8

YANG	Description	Details
choice	The "choice" statement defines a set of alternatives, only one of which may exist at any one time.	Clause 6.9
anyxml	The "anyxml" statement defines an interior node in the schema tree. The "anyxml" statement is used to represent an unknown chunk of XML.	Clause 6.10
grouping	The "grouping" statement is used to define a reusable block of nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it.	Clause 6.11
uses	The "uses" statement is used to reference a "grouping" definition. It takes one argument, which is the name of the grouping.	Clause 6.12
rpc	The "rpc" statement is used to define a NETCONF RPC operation.	Clause 6.13
notification	The "notification" statement is used to define a NETCONF notification.	Clause 6.14
augment	The "augment" statement allows a module or submodule to add to the schema tree defined in an external module, or the current module and its submodules, and to add to the nodes from a grouping in a "uses" statement.	Clause 6.15
identity	The "identity" statement is used to define a new globally unique, abstract, and untyped identity.	Clause 6.16
extension	The "extension" statement allows the definition of new statements within the YANG language. This new statement definition can be imported and used by other modules.	Clause 6.17
argument	The "argument" statement, which is optional, takes as an argument a string that is the name of the argument to the keyword. If no argument statement is present, the keyword expects no argument when it is used.	Clause 6.18
feature	The "feature" statement is used to define a mechanism by which portions of the schema are marked as conditional. A feature name is defined that can later be referenced using the "if-feature" statement.	Clause 6.19
if-feature	The "if-feature" statement makes its parent statement conditional.	Clause 6.20
deviation	The "deviation" statement defines a hierarchy of a module that the device does not implement faithfully.	Clause 6.21
config	The "config" statement takes as an argument the string "true" or "false". If "config" is "true", the definition represents configuration.	Clause 6.22
status	The "status" statement takes as an argument one of the strings "current", "deprecated", or "obsolete".	Clause 6.24
description	The "description" statement takes as an argument a string that contains a human-readable textual description of this definition. The text is provided in a language (or languages) chosen by the module developer;	Clause 6.25
reference	The "reference" statement takes as an argument a string that is used to specify a textual cross-reference to an external document, either another module that defines related management information, or a document that provides additional information relevant to this definition.	Clause 6.26
when	The "when" statement makes its parent data definition statement conditional. The node defined by the parent data definition statement is only valid when the condition specified by the "when" statement is satisfied.	Clause 6.27

6.1 Module statement

From RFC6020, the "module" statement defines the module's name, and groups all statements that belong to the module together. The "module" statement's argument is the name of the module, followed by a block of sub-statements that hold detailed module information.

6.1.1 Mapping YANG depiction to Redfish mockup

The *module* statement is depicted as follows:

```
module: [module-name]

module: ietf-system                    (System example)
```

The resultant URI for the module resource is shown below. The module resource is a subordinate resource to the NetworkDevice resource.

In which, [modified-module-name] is synthesized by changing the dashes "-" to underscores "_" in the module-name.

```
./NetworkDevices/{id}/[modified-module-name]

./NetworkDevices/{id}/ietf_system      (System example)
```

A mockup of the ietf_system resource is shown below.

```
{
    "@Redfish.Copyright": "",

    "@odata.context": "/redfish/v1/$metadata#NetworkDevices/Members/ietf dhcp/$entity",
    "@odata.type": "#ietf_dhcp.1.0.0.ietf_dhcp",
    "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_system",

    "Id": "ietf system",
    "Name": "System",

    "system": {
        "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/system"
    }
    "system state": {
        "@odata.id": "/redfish/v1/NetworkDevices/SW 15/ietf dhcp/system state"
    }
}
```

6.1.2 Mapping YANG code to Redfish CSDL

The YANG code for a *module* statement is shown below.

```
<CODE BEGINS> file "ietf-system@2014-08-06.yang"

  module ietf-system {
     namespace "urn:ietf:params:xml:ns:yang:ietf-system";
     prefix "sys";

     import ietf-yang-types {
         prefix yang;
     }

     organization "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
     contact "...";
     description "..."
     revision "2014-12-18";
     . . .
}
```

The resultant CSDL fragment is shown below. Note the following items in the mapping:

- The <edmx:Reference> tag is constructed from the *import* statements. The Uri and Namespace attributes are constructed from the *import* statement. The Alias attribute is constructed from the *prefix* statement.
- The <Schema> tag is constructed from the *namespace* and *prefi*x statements. The un-versioned <Schema> tag uses the *prefix* statement.
- There is an annotation for Redfish.Yang.NodeType
- Three annotation are added to the *contac*t, *description* and *revision* statements
- The annotations Redfish.Yang.description and Odata.Description are both present

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:Reference Uri=" http://redfish.dmtf.org/schemas/v1/ietf-inet-types.xml">
      <edmx:Include Namespace="ietf-inet-types.v1_0_0" Alias="inet" />
  </edmx:Reference>
  . . .

   <edmx:DataServices>

     <Schema Namespace="ietf_system" xmlns="urn.ietf.params.xml.ns.yang.ietf_system"
             Alias="sys">
          <Annotation Term="OData.LongDescription" String="[normative statement about RFC"/>

          <EntityType Name="ietf_system" BaseType="Resource.v1_0_0.Resource">
              <Annotation Term="Redfish.Yang.NodeType"
                  EnumMember ="Redfish.Yang.NodeTypes/module"/>
              <Annotation Term="Redfish.Yang.contact" String="…" />
              <Annotation Term="Redfish.Yang.description"
                  String="[text from description statement]" />
              <Annotation Term="Redfish.Yang.revision" String="2014-12-18" />
              <Annotation Term="OData.Description"
                  String="[text from description statement]"/>
          </EntityType>

     </Schema>

     <Schema Namespace="ietf_system.v1_0_0" xmlns="urn.ietf.params.xml.ns.yang.ietf_system"
             Alias="sys">

       <EntityType Name="ietf_system" BaseType="ietf_dhcp.ietf_system">
         <NavigationProperty Name="system" Type="system.system">
           <Annotation Term="OData.Permissions" EnumMember="OData.Permission/ReadWrite"/>
           <Annotation Term="OData.Description" String=""/>
           <Annotation Term="OData.LongDescription" String=""/>
           <Annotation Term="OData.AutoExpand"/>
         </NavigationProperty>
         <NavigationProperty Name="system_state" Type="system_state.system_state">
           <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
           <Annotation Term="OData.Description" String=""/>
           <Annotation Term="OData.LongDescription" String=""/>
           <Annotation Term="OData.AutoExpand"/>
         </NavigationProperty>
       </EntityType>

     </Schema>

   </edmx:DataServices>
</edmx:Edmx>
```

Table 3 shows the mapping of the *module* statement's sub-statements.

Table 3 – Module statement mapping

Statement	Mapping
anyxml	See clause 6.10
augment	See clause 6.15
choice	See clause 6.9
contact	<Annotation Term="Redfish.Yang.contact" String="[text from *contact* statement]"/>
container	See clause 6.4.4
description	See clause 6.25
deviation	See clause 6.21
extension	See clause 6.16.2
feature	See clause 6.19
grouping	See clause 6.11
identity	See clause 6.16
import	See clause 6.1.3
include	See clause 6.1.4
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
namespace	See clause 6.1.5
notification	See clause 6.1.3
organization	<Annotation Term="Redfish.Yang.organization" String="[text from *organization* statement"/>
prefix	See clause 6.1.6.
reference	See clause 6.26
revision	<Annotation Term="Redfish.Yang.revision" String="[text from r*evision* statement"> <Annotation Term="Redfish.Yang.description" String="[text from *description* statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from *reference* statement]"/> </Annotation>
rpc	See clause 6.12.1
typedef	See clause 6.2.1
uses	See clause 6.12
yang-version	<Annotation Term="Redfish.Yang.yang_version" String="[Text from yang-*version* statement]"/>

6.1.3 Import statement

The *import* statement is mapped to a <edmx:Reference> tag. The *import* statement text is used to synthesize the value of the Uri and Namespace attributes. The *prefix* statement is mapped to value of the tag's Alias attribute.

Open the import target and read the YANG module's namespace to fill in the Namespace attribute of the Edmx:Include statement.

The YANG import statement is shown below.

```
    prefix "dhcp";
        import <import_value {
        prefix <prefix value>;
    }
```

The resultant Redfish CSDL is shown below.

```
<edmx:Reference Uri="<uri value>">
    <edmx:Include Namespace="<namespace value>" Alias="<alias value>" />
</edmx:Reference>
```

In which

- <uri value> = http://redfish.dmtf.org/schemas/v1/<import value>.xml
- <namespace value> = <import value>.v1_0_0
- <alias value> = <prefix value>

The YANG import statement from DHCP is shown below.

```
    prefix "dhcp";
        import ietf-inet-types {
        prefix "inet";
    }
```

The resultant Redfish CSDL is shown below.

```
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ietf-inet-types.xml">
    <edmx:Include Namespace="ietf-inet-types.v1_0_0" Alias="inet" />
</edmx:Reference>
```

Table 4 shows the mapping of the *submodule* statement's sub-statements.

Table 4 – Import statement mapping

Statement	Mapping
prefix	<edmx:Include Alias="[text of prefix statement]"
revision-date	<Annotation Term="Redfish.Yang.revision_date" String="[text from revision-date statement"/>

6.1.4 Include statement

From RFC6020, the "include" statement is used to make content from a submodule available to that submodule's parent module, or to another submodule of that parent module. The argument is an identifier that is the name of the submodule to include.

Modules are only allowed to include submodules that belong to that module, as defined by the "belongs-to" statement. Submodules are only allowed to include other submodules belonging to the same module.

Open the include target and read the YANG module's namespace to fill in the Namespace attribute of the Edmx:Include statement.

6.1.5 Namespace statement

The namespace statement is mapped to the OData <schema> tag.

The YANG *namespac*e statement is shown below.

```
module <module value> {
     namespace <namespace value>;
     . . .
}
```

The resultant CSDL is shown below.

```
<schema Namespace="<Namespace value>" xmlns="<xmlns value>"
```

In which

- <Namespace value> = <module name>.v1_0_0
- <xmlns value> = <namespace value>"

The YANG code from DHCP is shown below.

```
module huawei-dhcp {
     namespace "urn:ietf:params:xml:ns:yang:ietf-dhcp";
     . . .
}
```

The resultant CSDL is shown below. In manual mapping, mapped to <schema xmlns value>

```
e.g. <schema Namespace="ietf_dhcp.v1_0_0" xmlns="urn:ietf:params:xml:ns:yang:ietf-dhcp">
```

6.1.6 Prefix statement

See clause 6.1.3, which also describes the *prefix* statement.

6.2 Submodule statement

While the primary unit in YANG is a module, a YANG module can itself be constructed out of several submodules. The "submodule" statement defines the submodule's name, and groups all statements that belong to the submodule together. The "submodule" statement's argument is the name of the submodule, followed by a block of sub-statements that hold detailed submodule information.

The YANG *submodule* is depicted as follows:

```
submodule: [submodule-name]
```

The resultant Redfish construct is a singleton resource. The modified-submodule-name is created by changing the dashes "-" to underscores "_" in the submodule-name.

```
./NetworkDevices/{id}/[modified-name]
```

The following is example YANG code for a *submodule* statement.

```
submodule acme-types {
     belongs-to "acme-system" {
         prefix "acme";
     }
     import ietf-yang-types {
         prefix "yang";
     }
     organization "ACME Inc.";
     contact
         "Joe L. User
         ACME, Inc. . . . ";
```

```
    description "This submodule defines common ACME types.";
    revision "2007-06-09" {
        description "Initial revision.";
    }
    ...
}
```

Table 5 shows the mapping of the *submodule* statement's sub-statements.

Table 5 – Submodule statement mapping

Statement	Mapping
anyxml	See clause 6.10
augment	See clause 6.15
belongs-to	See clause 6.2.1
choice	See clause 6.9
contact	<Annotation Term="Redfish.Yang.contact" String="[text from *contact* statement]"/>
container	See clause 6.4.4
description	See clause 6.25
deviation	See clause 6.21
extension	See clause 6.16.2
feature	See clause 6.19
grouping	See clause 6.11
identity	See clause 6.16
import	See clause 6.1.3
include	See clause 6.1.4
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
namespace	See clause 6.1.5
notification	See clause 6.1.3.
organization	<Annotation Term="Redfish.Yang.organization" String="[text from *organization* statement"/>
reference	See clause 6.26
revision	<Annotation Term="Redfish.Yang.revision" String="[text from r*evision* statement"> <Annotation Term="Redfish.Yang.description" String="[text from *description* statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from *reference* statement]"/> </Annotation>
rpc	See clause 6.12.1
typedef	See clause 6.2.1
uses	See clause 6.12
yang-version	<Annotation Term="Redfish.Yang.yang_version" String="[Text from *version* statement]"/>

6.2.1 Belongs-to statement

From RFC602, the "belongs-to" statement specifies the module to which the submodule belongs. The argument is an identifier that is the name of the module. A submodule MUST only be included by the module to which it belongs, or by another submodule that belongs to that module.

The mandatory "prefix" substatement assigns a prefix for the module to which the submodule belongs.

The CSDL for the *belongs-to* statement is shown below.

```
<Annotation Term="Redfish.Yang.belongs to" String="[text from belongs-to statement">
    <Annotation Term="Redfish.Yang.prefix" String="[text from prefix statement]"/>
</Annotation>
```

6.3 Typedef statement

The "typedef" statement defines a new type that may be used locally in the module, in modules or submodules which include it, and by other modules that import from it. The new type is called the "derived type", and the type from which it was derived is called the "base type". All derived types can be traced back to a YANG built-in type.

There is no YANG depiction of a YANG *typedef* statement.

6.3.1 Mapping YANG code to Redfish CSDL

An example of the *typedef* statement from RFC 6991 (Common YANG data types) is shown below.

```
typedef gauge32 {
       type uint32;
       description "...";
       reference "…";
}
```

The resultant Redfish construct is a TypeDefinition shown below.

```
<TypeDefinition Name="gauge32" UnderlyingType="Edm.Redfish.Yang.uint32">
     <Annotation Term="Redfish.Yang.description" String="…"/>
     <Annotation Term="Redfish.Yang.reference" String="…"/>
</TypeDefinition>
```

Another example of the *typedef* statement from RFC 6991 (Common YANG data types) is shown below. This one with a non-built-in type. Instead, *listen-ipv4*-address is derived from the exist type *inet:ipv4-address.*

```
typedef listen-ipv4-address {
         type inet:ipv4-address;
         default "0.0.0.0";
}
```

The resultant Redfish construct is a TypeDefinition declaration in the CSDL

```
<TypeDefinition Name="listen_ipv4_address" UnderlyingType="Edm.String">
     <Annotation Term="Validation.Pattern" String="^?:[0-9]{1,3}\\.){3}[0-9]{1,3}$"/>
     <Annotation Term="Redfish.Yang.default" String="0.0.0.0"/>
</TypeDefinition>
```

Table 6 shows the mapping of the *typedef* statement's sub-statements.

Table 6 – Typedef statement mapping

Statements	Mapping
default	See clause 6.3.2
description	See clause 6.25

Statements	Mapping
reference	See clause 6.26
status	See clause 6.24
type	**UnderlyingType = <type_name>**
units	<Annotation Term="Redfish.Yang.units" String="Text from units statement"/>

6.3.2 Default statement

The default value from the typedef statement is used, if the leaf or leaf-list statements does not have a default sub-statement present, use the default value from the typedef of the leaf or leaf-list type sub-statement to set the CSDL DefaultValue of the leaf or leaf-list corresponding property.

The default statement shall be mapped to an annotation in the CSDL and the value of the DefaultValue attribute of the Property property. The annotation shall be of the form shown below.

```
<Annotation Term="Redfish.Yang.default" String="Text from default statement"/>
```

The resultant Redfish CSDL for the example above is shown below.

```
<Property Name="listen_ipv4_address", Type="inet.ipv4_address",
       DefaultValue="0.0.0.0" >
   <Annotation Term="Redfish.Yang.YangType" String="inet.ipv4address"/>
   <Annotation Term="Redfish.Yang.default" String="0.0.0.0"/>
   . . .
</Property>
```

6.4 Type statement

From RFC6020, the "type" statement takes as an argument a string that is the name of a YANG built-in type or a derived type, followed by an optional block of sub-statements that are used to put further restrictions on the type. The restrictions that can be applied depend on the type being restricted.

Table 7 shows the list of YANG built-in types.

Table 7 – Built in YANG types

Name	Description	CSDL Mapping
binary	Any binary data	Edm.Binary
bits	A set of bits or flags	Edm.Binary
boolean	"true" or "false"	Edm.Boolean
date-and-time	Date and time	Edm.DateTimeOffset
decimal64	64-bit signed decimal number	Edm.Decimal
empty	A leaf that does not have any value	See clause 6.4.4.1
enumeration	Enumerated strings	See clause 6.4.4.2
identityref	A reference to an abstract identity	See clause 6.4.4.2
instance-identifier	References a data tree node	Redfish.Yang.instance_identifier
int8	A 8-bit signed integer	Edm.Sbyte
int16	A 16-bit signed integer	Edm.Int16

Name	Description	CSDL Mapping
int32	A 32-bit signed integer	Edm.Int32
int64	A 64-bit signed integer	Edm.Int64
leafref	A reference to a leaf reference	See clause 6.4.4.3
string	A human readable string	Edm.String
uint8	A 8-bit unsigned integer	Edm.Byte
uint16	A 16-bit unsigned integer	Redfish.Yang.uint16
uint32	A 32-bit unsigned integer	Redfish.Yang.uint32
uint64	A 64-bit unsigned integer	Redfish.Yang.uint64
union	A choice of member types	See clause 6.4.4.2

In Redfish.Yang.Types, there are TypeDefintion's that reflect the above table.

```
<TypeDefinition Name="uint16" UnderlyingType="Edm.Int32" />
<TypeDefinition Name="uint32" UnderlyingType="Edm.Int64" />
<TypeDefinition Name="uint64" UnderlyingType="Edm.Decimal" />
```

The type statement is mapped to following annotation

```
<Annotation Term="Redfish.Yang.YangType" String="[value of type statement]"/>
```

There is no YANG depiction of a YANG *type* statement.

6.4.1 Mapping YANG code to Redfish CSDL

A *type* statement from DHCP is shown below.

```
leaf enable {
     description "Enable or disable dhcp relay function";
     type "boolean";
     default "false";
     config "true";
}
```

The resultant Redfish is shown below. The value of the type statement is mapped to the Type value in the Property definition. The annotation is also added to preserve the original YANG type.

```
<Property Name="enable", Type="edm:Boolean" >
   <Annotation Term="Redfish.Yang.YangType" String="boolean"/>
   . . .
</Property>
```

Table 8 shows the mapping of the *type* statement's sub-statements.

Table 8 – Type statement mapping

Statements	Mapping
base	<Annotation Term="Redfish.Yang.base" String="the_yang_statement_base string"/>
bit	<Annotation Term="Redfish.Yang.bit" String="bit_name"/> <Annotation Term="Redfish.Yang.position" Redfish.Yang.uint32=bit_position"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus"/ </Annotation>

Statements	Mapping
enum	**Instance of edm.Member where Name = "enum_name"**
	<Annotation> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus"/ </Annotation>
length	<Annotation Term="Redfish.Yang.length" String="the length sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>
path	See clause 6.4.2
pattern	<Annotation Term="Redfish.Yang.pattern" String="[text from the pattern statement]"> <Annotation Term="Redfish.Yang.error_message" String="[text from error-message statement]"/> <Annotation Term="Redfish.Yang.error_app_tag" String="[text from error-app-tag statement]"/> <Annotation Term="Redfish.Yang.description" String="[text from description statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from reference statement]"/> </Annotation>
range	<Annotation Term="Redfish.Yang.range" String="the range sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="[text from error-message statement]"/> <Annotation Term="Redfish.Yang.error_app_tag" String="[text from error-app-tag statement]"/> <Annotation Term="Redfish.Yang.description" String="[text from description statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from reference statement"/> </Annotation>
required-instance	See clause 6.4.3
type	Ignore. The *type* sub-statement is not supported.

6.4.2 Path statement

The "path" statement, takes as an argument a string that MUST refer to a leaf or leaf-list node. The syntax for a path argument is a subset of the XPath abbreviated syntax. Predicates are used only for constraining the values for the key nodes for list entries. Each predicate consists of exactly one equality test per key, and multiple adjacent predicates MAY be present if a list has multiple keys.

6.4.3 require-instance statement

The "require-instance" statement MAY be present if the type is "instance-identifier". It takes as an argument the string "true" or "false".

If "require-instance" is "true", it means that the instance being referred MUST exist for the data to be valid.

If "require-instance" is "false", it means that the instance being referred MAY exist in valid data.

The CSDL annotation is show below.

```
<Annotation Term="Redfish.Yang.require_instance"
         String="[text from require-instance statement]"/>
```

6.4.4 Mapping special types

Returning to Table 7, some of the built-in YANG types are mapped to something more complex than a simple annotation. The following clause specifies that mapping of each of these special built-in YANG types.

6.4.4.1 Empty type

From RFC6020, the empty built-in type represents a leaf that does not have any value, it conveys information by its presence or absence.

Neither CSDL nor json-schema support this semantic.

The *empty* statement is mapped to a read-only string that only returns the name of the leaf.

The *empty* statement shall be mapped to an annotation in the CSDL and a Property that only contains the value of the empty statement.

The YANG depiction is shown below.

```
     +--ro is-router?             empty
```

The resultant Redfish CSDL for the example above is shown below.

```
  <Property Name="is router", Type="Redfish.Yang.empty", DefaultValue="is router" >
     <Annotation Term="Redfish.Yang.YangType" String="empty"/>
  </Property>
```

6.4.4.2 Enumeration type

From RFC 6020, the enumeration built-in type represents values from a set of assigned names.

The enumeration type will be mapped to Odata EnumType.

The YANG code for the enumeration type from RFC 6991 (Common YANG Types) is shown below.

```
typedef ip-version {
     type enumeration {
         enum unknown {
           value "0";
           description
            "An unknown or unspecified version of the Internet
             protocol.";
         }
         enum ipv4 {
           value "1";
           description
            "The IPv4 protocol as defined in RFC 791.";
         }
         enum ipv6 {
           value "2";
           description
            "The IPv6 protocol as defined in RFC 2460.";
         }
     )
}
```

The resultant Redfish CSDL is shown below. (system example)

```
<EnumType Name="association_typeEnumeration">
         <Member Name="server">
```

```
     <Annotation Term="Redfish.Yang.enum" String="server"/>
         <Annotation Term="OData.Description"
             String="Use client association mode.[...]"/>
         </Member>
         <Member Name="peer">
     <Annotation Term="Redfish.Yang.enum" String="peer"/>
     <Annotation Term="OData.Description"
             String="Use symmetric active association mode.[...]"/>
         </Member>
         <Member Name="pool">
     <Annotation Term="Redfish.Yang.enum" String="pool"/>
         <Annotation Term="OData.Description"
             String="Use client association mode with one or more of the NTP
servers.[...]"/>
          </Member>
      </EnumType>
Identifyref Type
```

From RFC6020, the identityref type is used to reference an existing identity.

The "base" statement, which is a substatement to the "type" statement, MUST be present if the type is "identityref". The argument is the name of an identity, as defined by an "identity" statement.

The YANG code from RFC7223

```
        leaf type {
           type identityref {
             base interface-type;
           }
           mandatory true;
           description "…";
           reference
             "RFC 2863: The Interfaces Group MIB - ifType";
         }
```

6.4.4.3 Leafref type

From RFC6020, the leafref built-in type is used to reference a particular leaf instance in the data tree. The "path" sub-statement selects a set of leaf instances, and the leafref value space is the set of values of these leaf instances. The "path" statement MUST be present if the type is "leafref".

The value of Leaftype is set to the type of the Edm.Property for the leaf is the type of the leafref's target leaf node. Returns the value of another leaf.

The YANG code from RFC7223

```
typedef interface-state-ref {
       type leafref {
         path "/if:interfaces-state/if:interface/if:name";
       }
       description
         "This type is used by data models that need to reference
          the operationally present interfaces.";
     }


leaf-list higher-layer-if {
           type interface-state-ref;
           description
             "A list of references to interfaces layered on top of this
              interface.";
           reference
             "RFC 2863: The Interfaces Group MIB - ifStackTable";
         }
```

The resultant CSDL is shown below (path value is considered a opaque string, therefore the colons remain.

```
<Property Name="<name of the leaf with has the type specified by the leafref",
Type="(derived by derferencing the path and using the type of dereferenced target" >
     <Annotation Term="Redfish.Yang.YangType" String="leafref">
         <Annotation Term="Redfish.Yang.path" String="if:interfaces/if:interface/if:name
"/>
     </Annotation>
  </Property>
```

An example

```
<Property Name="higher-layer-if", Type="string" >
     <Annotation Term="Redfish.Yang.YangType" String="leafref">
         <Annotation Term="Redfish.Yang.path" String="if:interfaces/if:interface/if:name
"/>
     </Annotation>
</Property>
```

6.4.4.4 Union type

From RFC6020, the union built-in type represents a value that corresponds to one of its member types.

A member type can be of any built-in or derived type, except it MUST NOT be one of the built-in types "empty" or "leafref".

For example:

```
type union {
     type int32;
     type enumeration {
         enum "unbounded";
     }
}
```

6.4.4.4.1 Mockup

The JSON payload would include an @odata.type annotation to specify the type of the actual IPAddress:

```
{
     …
     "IPAddress":"…",
     "IPAddress@odata.type":"#IP.IPV4_no_zone"
}
```

6.4.4.4.2 Mapping YANG code to Redfish CSDL

The union statement can be mapped two ways in CSDL.

One option is that the IPAddress property can be annotated with a Redfish.Yang.Union annotation, which specifies the possible values within a collection.

```
  <Property Name="IPAddress" Type="Edm.Primitive">
    <Annotation Term="Redfish.Yang.Union">
      <Collection>
        <String>"IPV4_no_zone"</String>
        <String>"IPV6 no zone"</String>
      </Collection>
    </Annotation>
  </Property>
```

The Redfish.Yang.Union annotation is specified elsewhere, as a collection type.

```
  <Term Name="Union" Type="Collection(String)">
    <Annotation Term="OData.Description" String=""/>
  </Term>
```

Another options is that the IPAddress property specifies a property type definition for the union.

```
   <Property Name="IPAddress" Type="IP.ip_no_zone"/>
```

The type definition declares that ip_no_zone has an underlying type of "Edm.Primitive and specifies the possible types.

```
  <TypeDefinition Name="ip_no_zone" UnderlyingType="Edm.Primitive">
    <Annotation Term="Redfish.Yang.Union">
      <Collection>
        <String>"IP.IPV4 no zone"</String>
        <String>"IP.IPV6_no_zone"</String>
      </Collection>
    </Annotation>
  </TypeDefinition>
```

6.5 Container statement

From RFC6020, the "container" statement is used to define an interior data node in the schema tree. It takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed container information.

A container node does not have a value, but it has a list of child nodes in the data tree. The child nodes are defined in the container's sub-statements.

YANG supports two styles of containers, those that exist only for organizing the hierarchy of data nodes, and those whose presence in the configuration has an explicit meaning.

6.5.1 Mapping the YANG depiction to Redfish mockup

The YANG *container* is depicted is show below.

```
+--[container-name]

+--relay            (DHCP example)
  +--rw dhcpRelayIfCfgs
  +--rw dhcpRelayServerGroups
  +--r dhcpRelayStatistics
```

The resultant Redfish construct is a singleton resource

```
./NetworkDevices/{id}/[module-name]/[container-name]

./NetworkDevices/{id}/ietf_dhcp/relay          (DHCP example)
```

A mockup of the "relay" resource is shown below. It contains navigation links for the containers contained by "relay".

```
{
    "@Redfish.Copyright": "",
    "@odata.context":
"/redfish/v1/$metadata#NetworkDevices/Member/ietf dhcp/relay/$entity",
    "@odata.type": "#relay.1.0.0.relay",
    "@odata.id": "/redfish/v1/NetworkDevices/SW 15/ietf dhcp/relay",

    "Id": "relay",
    "Name": "DHCP Relay Service",
```

```
    "dhcpRelayIfCfgs": {
        "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs"
    },
    "dhcpRelayServerGroups": {
        "@odata.id":
"/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayServerGroups"
    },
    "dhcpRelayStatistics": {
        "@odata.id": "/redfish/v1/NetworkDevices/SW 15/ietf dhcp/relay/dhcpRelayStatistics"
    }
}
```

6.5.2 Mapping YANG code to Redfish CSDL

The YANG code for the "relay" container statement from DHCP is shown below.

```
container relay {
     container dhcpRelayIfCfgs {
         . . .
     }
     Container dhcpRelayServerGroups {
         . . .
     }
     Container dhcpRelayStatistics {
         . . .
     }
}
```

The resultant CSDL fragment for relay container statement is shown below. There is a Navigation property for each sub-container.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright 2014-2015 Distributed Management Task Force, Inc. (DMTF). All rights
reserved.-->
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">

  <edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/cs01/vocabularies/Org.OData.Core.V1.xml">
    <edmx:Include Namespace="Org.OData.Core.V1" Alias="OData" />
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Resource.xml">
    <edmx:Include Namespace="Resource.1.0.0"  />
  </edmx:Reference>

  <edmx:DataServices>

    <Schema Namespace="relay" xmlns="http://docs.oasis-open.org/odata/ns/edm">
      <EntityType Name="relay" BaseType="Resource.1.0.0.Resource">
        <Annotation Term="OData.Description" String=""/>
        <Annotation Term="OData.AdditionalProperties" Bool="false"/>
      </EntityType>
    </Schema>

    <Schema Namespace="relay.1.0.0" xmlns="http://docs.oasis-open.org/odata/ns/edm">
      <EntityType Name="relay" BaseType="relay.relay">
         Annotation Term="Redfish.Yang.NodeType"
             EnumMember ="Redfish.Yand.NodeTypes/container"/>
         <Annotation Term="OData.Description" String=""/>
         <Annotation Term="OData.AdditionalProperties" Bool="false"/>
         <NavigationProperty Name="dhcpRelayIfCfgs"
                Type="dhcpRelayIfCfgsCollection.dhcpRelayIfCfgsCollection"
                ContainsTarget="true">
            <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
            <Annotation Term="OData.Description" String=""/>
            <Annotation Term="OData.LongDescription" String=""/>
            Annotation Term="OData.AutoExpandReferences"/>
         </NavigationProperty>
         <NavigationProperty Name="dhcpRelayServerGroups"
                Type="dhcpRelayServerGroupsCollection.dhcpRelayServerGroupsCollection"
```

```
                ContainsTarget="true">
            <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
            <Annotation Term="OData.Description" String=""/>
            <Annotation Term="OData.LongDescription" String=""/>
            <Annotation Term="OData.AutoExpandReferences"/>
        </NavigationProperty>
        <NavigationProperty Name="dhcpRelayStatistics"
                Type="dhcpRelayStatistics.dhcpRelayStatistics"
                ContainsTarget="true">
            <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
            <Annotation Term="OData.Description" String=""/>
            <Annotation Term="OData.LongDescription" String=""/>
            <Annotation Term="OData.AutoExpandReferences"/>
        </NavigationProperty>
     </EntityType>
    </Schema>

  </edmx:DataServices>
</edmx:Edmx>
```

Table 9 shows the mapping of the *container* statement's sub-statements.

Table 9 – Container statement mapping

Statement	Mapping
container	Recursion. See this clause.
list	See clause 6.8
leaf	See clause 6.6
leaf-list	See clause 6.7
presence	<Annotation Term="Redfish.Yang.presence" String="text from presence statement"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>
when	See clause 6.27
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
choice	See clause 6.9
grouping	See clause 6.11
uses	See clause 6.12
anyxml	See clause 6.10

6.6 Leaf statement

From RFC6020, the "leaf" statement is used to define a leaf node in the schema tree. It takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed leaf information.

The *leaf* statement is mapped to a JSON property.

6.6.1 Mapping YANG depiction to Redfish mockup

The YANG depiction of the *leaf* statement is shown below.

```
+--[permission] [leaf-name] [leaf-type]

+--rw serverGroupName string                    (DHCP example)
```

The resultant Redfish is a JSON property within a resource mockup.

```
[leaf-name]: "[value]"

"serverGroupName": "webservers"                 (DHCP example)
```

6.6.2 Mapping YANG code to Redfish CSDL

The YANG code for a *leaf* statement is shown below.

```
leaf clientRequestCount {
     description "Client Request Count";
     type uint32;
     config "false";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="clientRequestCount" Type="Redfish.Yang.uint32">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
     <Annotation Term="Redfish.Yang.YangType" String="uint32"/>
     . . .
</Property>
```

Table 10 shows the mapping of the *leaf* statement's sub-statements.

Table 10 – Leaf statement mapping

Statement	Mapping
type	See clause 6.3.2
units	<Annotation Term="Redfish.Yang.units" String="[text from units statement]"/>
default	See clause 6.3.2
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>

Statement	Mapping
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

6.7 Leaf-list statement

The "leaf-list" statement is used to define an array of a particular type. The "leaf-list" statement takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed leaf-list information.

The leaf-list statement is mapped to JSON property array which the mockup.

6.7.1 Mapping YANG depiction to Redfish mockup

The YANG *leaf-list* statement is depicted is shown below. The depiction is identical to the depiction of a *leaf* statement. One needs to consult the YANG code to view the statement.

```
+--[permission] [leaf-list-name] [leaf-type]

+--rw serverAddress inet:ipv4-address          (DHCP example)
```

The resultant Redfish construct is a JSON array property within the resource mockup.

```
"[leaf-list-name]": [
     "[value 1]",
     "[value 2]"
     . . .
}

"serverAddress": [                             (DHCP example)
     "[ip address 1]",
     "[ip address 2]"
]
```

6.7.2 Mapping YANG code to Redfish CSDL

The YANG code from *leaf-list* statement of DHCP is shown below. (with

```
leaf-list serverAddress {
     description "DHCP relay destination server IP address";
     type inet:ipv4-address;
     config "true";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="serverAddress" Type="Collection(Yang.inet:ipv4-address)">
     <Annotation Term="Redfish.Yang.NodeType"
         EnumMember="Redfish.Yang.NodeTypes/leaf_list" />
     <Annotation Term="Redfish.Yang.YangType" String="inet:ipv4-address" />
<Property>
```

1011 **Table 11 – Leaf-list statement mapping**

Statement	Mapping
type	See clause 6.5
units	<Annotation Term="Redfish.Yang.units" String="Text from units statement"/>
max-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
ordered-by	<Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/false"/> <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>
config	See clause 6.22
If-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1012 ## 6.8 List statement

1013 A list defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely
1014 identified by the values of its key leafs. A list can define multiple key leafs and may contain any number
1015 of child nodes of any type (including leafs, lists, containers, etc.)

1016 The *list* statement is mapped to a Redfish collection resource and its member resource.

1017 ### 6.8.1 Mapping YANG depiction to Redfish mockup

1018 The YANG depiction of the *list* statement is shown below.

1019
1020
1021
1022
1023
1024
1025
1026
1027
1028

```
+--rw [list-name]
| +--rw [list-member-name]* [[name-of-member]]
| +--rw [name-of-member] string
. . .

+--rw dhcpRelayIfCfgs                          (DHCP example)
| +--rw dhcpRelayIfCfg* [ifName]
| +--rw ifName string
. . .
```

The resultant Redfish is a collection resource and singleton resources. The value of the "ifName" *leaf* statement is used at the name of the member of the collection.

```
./NetworkDevices/{id}/ietf_dhcp/relay/dhcpRelayIfCfgs
./NetworkDevices/{id}/ietf_dhcp/relay/dhcpRelayIfCfgs/[Text of ifName leaf statement]
```

A mockup of the "dhcpRelayIfCfgs" collection resource is shown below.

```
{
     . . .
     "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs",

     "Name": "Collection of interface configurations for DHCP relay service",
     "Members@odata.count": 1,
     "Members": [
         {"@odata.id":
"/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs/IF_foo"
        }
     ]
}
```

A mockup of the 'IF_foo' singleton dhcpRelayIfCfg resource is show below.

```
{
     . . .
     "Id": "IF_foo",
     "Name": "Interface configuration for a DHCP relay service",

     "ifName": "IF foo",
     "enable": "TRUE",
     "serverAddress": "192.168.1.10",
     . . .
    }
}
```

6.8.2 Mapping YANG code to Redfish CSDL

The YANG code for the *list* statement from the DHCP is shown below.

```
list dhcpRelayIfCfg {
     key "ifName";
     leaf ifName {
         description "Specify the interface name that dhcp relay configured on";
         type "string";
         config "true";
     }
     . . .
}
```

The CSDL for the collection resource is shown below.

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">

  . . .
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Namespace="dhcpRelayIfCfg.xml">
    <edmx:Include Namespace="dhcpRelayIfCfg"/>
  </edmx:Reference>

  <edmx:DataServices>

    <Schema Namespace="Namespace="dhcpRelayIfCfgsCollection"
         xmlns="http://docs.oasis-open.org/odata/ns/edm" >
      <EntityType Name="dhcpRelayIfCfgsCollection"
            BaseType="Resource.1.0.0.ResourceCollection">
        <NavigationProperty Name="Members"
                Type="Collection(dhcpRelayIfCfg.dhcpRelayIfCfg)">
```

```
            <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
            <Annotation Term="OData.Description" String=""/>
            <Annotation Term="OData.AutoExpandReferences"/>
          </NavigationProperty>
        </EntityType>
      </Schema>

  </edmx:DataServices>
</edmx:Edmx>
```

Table 12 shows the mapping of the *list* statement.

Table 12 – List statement mapping

Statement	Mapping
container	See clause 6.4.4
list	See clause 6.8
leaf	See clause 6.6
leaf-list	See clause 6.7
key	See clause 6.8.3
max-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
ordered-by	One of <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/false"/> <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/true"/>
must	<Annotation Term="Redfish.Yang.must" String="text from the XPath statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
choice	See clause 6.9
grouping	See clause 6.11
uses	See clause 6.12
anyxml	See clause 6.10
unique	<Annotation Term="Redfish.Yang.unique" String="text from unique statement"/>
when	See clause 6.27

6.8.3 Key statement

From RFC6020, the "key" statement, which MUST be present if the list represents configuration, and MAY be present otherwise, takes as an argument a string that specifies a space-separated list of leaf identifiers of this list. A leaf identifier MUST NOT appear more than once in the key. Each such leaf identifier MUST refer to a child leaf of the list. The leafs can be defined directly in sub-statements to the list, or in groupings used in the list.

The combined values of all the leafs specified in the key are used to uniquely identify a list entry.

The *key* statement value is a space separated sting of leaf names. Typically there will be only one key token in the key string but there are a couple cases of 2 or more keys.

The following is a *key* sub-statement from DHCP

```
<EntityType Name=”dhcpRelayIfCfgs”>
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/list"/>
     . . .
     <Annotation Term="Redfish.Yang.key" String=" the yang key string"/>
     <Key>
         <PropertyRef Name="ifName" />
     </Key>
     . . .
}
```

The set of names constitutes the key for this list. The ithKeyName corresponds to the ith string token in the key string. We add annotations containing the original yang information in addition to the actual translation to make understanding the translated schema clearer.

```
<EntityType Name=”theListName”>
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/list"/>
     . . .
     <Annotation Term="Redfish.Yang.key" String=" the yang key string"/>
     <Key>
         <PropertyRef Name=”firstKeyName” />
         …
         <PropertyRef Name=”ithKeyName” />
         …
         <PropertyRef Name=”lastKeyName” />
     </Key>
     . . .
     <Property Name=”firstlLeafName” Type=”translatedLeafType”>
         <Annotation Term="Redfish.Yang.NodeType" EnumMember
="Redfish.Yang.NodeTypes/leaf"/>
         <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
     </Property>
     <Property Name=”nthLeafName” Type=”translatedLeafType”>
         <Annotation Term="Redfish.Yang.NodeType" EnumMember
="Redfish.Yang.NodeTypes/leaf"/>
         <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
     </Property>
     <Property Name=”lastLeafName” Type=”translatedLeafType”>
         <Annotation Term="Redfish.Yang.NodeType" EnumMember
="Redfish.Yang.NodeTypes/leaf"/>
         <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
     </Property>
     . . .
</EntityType>
```

6.9 Choice statement

From RFC6020, the "choice" statement defines a set of alternatives, only one of which may exist at any one time. The argument is an identifier, followed by a block of sub-statements that holds detailed choice

information. The identifier is used to identify the choice node in the schema tree. A choice node does not exist in the data tree.

A choice consists of a number of branches, defined with the "case" sub-statement. Each branch contains a number of child nodes. The nodes from at most one of the choice's branches exist at the same time.).

The choice statement maps to a Redfish collection resource and the key maps to members of the collection

The choice statement maps to annotations for the choice and each of the cases with the containing resource. The positioning shall correspond to the position of the element within the case statement.

The choice annotation will have as children the translated annotations for the directly dependent non-node YANG statements of the choice plus case statements of the choice plus a <Redfish.Yang.NodeName < Redfish.Yang.NodeType > />statement hierarchy for each node.

Put all node elements from all cases directly in the parent node and annotate each one individually with a "choice annotation and a case annotation" s.

Create Annotations to represent the choice/case structure. Create an Annotation for the choice itself in the context of its parent container. The choice annotation will have as children the translated annotations for the directly dependent non-node yang statements of the choice plus case statements of the choice plus a <Redfish.Yang.NodeName < Redfish.Yang.NodeType > />statement hierarchy for each node

The nodes themselves will otherwise be translated in the context of the parent node of the choice statement as direct properties of the parent node plus EntityType objects as needed to translate list and container.

6.9.1 Mapping the YANG depiction to Redfish mockup

The YANG *choice* depiction from RFC7317 is shown below. For "timezone", there is choice between "timezone-name" and "timezone-utc-offset".

```
           +--ro (timezone)?
              +--:(timezone-name)
              |  +--ro timezone-name?   string
              +--:(timezone-utc-offset)
                 +--ro timezone-utc-offset?  int16
```

The possible resultant Redfish mockups are the shown below.

```
{
     "timezone-name": "Europe/Stockholm",
}
```

```
{
     "timezone-utc-offset": "3",
}
```

6.9.2 Translating the YANG depiction to Redfish mockup

The YANG code from RFC7317 is shown below.

```
    container clock {
     description "Configuration of the system date and time properties.";

     choice timezone {
       description "The system time zone information.";

       case timezone-name {
         if-feature timezone-name;
         leaf timezone-name {
           type timezone-name;
```

```
            description "The TZ database name to use for the system, such as
               'Europe/Stockholm'.";
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {
            type int16 {
              range "-1500 .. 1500";
            }
            units "minutes";
            description "The number of minutes to add to UTC ...";
          }
        }
      }
    }
```

The resultant CSDL fragment for the DHCP service is shown below.

```
<EntityType Name = “clock” >
    <Annotation Term="Redfish.Yang.description" String="Configuration of the system date
         and time properties."/>
    <Annotation Term="Redfish.Yang.choice" String="timezone">
        <Annotation Term="Redfish.Yang.description" String="The system time zone
information."/>
        <Annotation Term="Redfish.Yang.case" String="timezone-name">
             <Annotation Term="Redfish.Yang.if feature" String="timezone-name"/>
             <Annotation Term="Redfish.Yang.NodeName" String="timezone-name" >
                 <Annotation Term="Redfish.Yang.NodeType"
                    EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
             </Annotation>
        </Annotation>
        <Annotation Term="Redfish.Yang.case" String="timezone-utc-offset">
             <Annotation Term="Redfish.Yang.NodeName" String="timezone-utc-offset" >
                 <Annotation Term="Redfish.Yang.NodeType"
                    EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
             </Annotation>
        </Annotation>
  </Annotation>

  <Property Name = “timezone_name”  Type = “timezone_name”>
      <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
      <Annotation Term="Redfish.Yang.YangType" String="timezone-name"/>
      <Annotation Term="Redfish.Yang.description" String="The TZ database name to….”/>
      <Annotation Term="Redfish.Yang.choice" String="timezone"/>
      <Annotation Term="Redfish.Yang.case" String="timezone-name"/>
  </Property>

  <Property Name = “timezone_utc_offset”  Type = “int16”>
      <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
      <Annotation Term="Redfish.Yang.YangType" String="int16">
         <Annotation Term="Redfish.Yang.range" String="-1500 .. 1500"/>
      </Annotation>
      <Annotation Term="Redfish.Yang.units" String=" minutes "/>
      <Annotation Term="Redfish.Yang.description"
            String="The number of minutes to add to UTC time to..."/>
      <Annotation Term="Redfish.Yang.choice" String="timezone"/>
      <Annotation Term="Redfish.Yang.case" String="timezone-utc-offset"/>
    </Property>

</EntityType>
```

Table 13 shows the mapping of the *choic*e statement's sub-statements.

1258 **Table 13 – Choice statement mapping**

Statements	Mapping
anyxml	See clause 6.10
case	See clause 6.9.3
choice	See clause 6.9
container	See clause 6.4.4
default	<Annotation Term="Redfish.Yang.default" String="the_yang_default_string"/>
description	See clause 6.25
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1259 ### 6.9.3 Case

1260 From RFC6020, the "case" statement is used to define branches of the choice. It takes as an argument
1261 an identifier, followed by a block of sub-statements that holds detailed case information.

1262 The identifier is used to identify the case node in the schema tree. A case node does not exist in the data
1263 tree.

1264 See clause 6.9 for the mapping details.

1265 ## 6.10 Anyxml statement

1266 From RFC6020, The "anyxml" statement defines an interior node in the schema tree. It takes one
1267 argument, which is an identifier, followed by a block of sub-statements that holds detailed anyxml
1268 information.

1269 The "anyxml" statement is used to represent an unknown chunk of XML. No restrictions are placed on the
1270 XML.

1271 The *anyxml* statement is mapped to an annotation within its parent container and parent annotation.

1272 ### 6.10.1 Mapping YANG depiction to Redfish mockup

1273 An example of a YANG depiction of the *anyxml* statement has not been found.

6.10.2 Mapping YANG code to Redfish CSDL

The YANG code from *anyxml* statement from RFC6020 is shown below.

```
anyxml data;
```

The resultant CSDL is shown below.

```
<Term Name="IsXml" Type="Edm.Boolean" Default="True">
    <Annotation Term="OData.Description" String="The string type contains XML"/>
</Term>

<TypeDefinition Name="XmlBlock" UnderlyingType="Edm.String">
    <Annotation Term="Redfish.Yang.IsXml"/>
</TypeDefinition>

<Property Name="myProperty" Type="Redfish.Yang.XmlBlock">
     [text from anyxml statement]
</Property>
```

Where "myProperty" is a unique name synthesized by appending a number to the string "Anyxml_".

Table 14 shows the mapping of the *anyxml* statement's sub-statements.

Table 14 – Anyxml statement mapping

Statement	Mapping
config	See clause 6.22
description	See clause 6.25
if-feature	See clause 6.20
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

6.11 Grouping statement

From RFC6020, the "grouping" statement is used to define a reusable block of nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it. It takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed grouping information.

The "grouping" statement is not a data definition statement and, as such, does not define any nodes in the schema tree. A grouping is like a "structure" or a "record" in conventional programming languages.

The *grouping* and *uses* statement should be handled and resolved prior to mapping the YANG to CSDL. Since the *grouping* statement does not define a node in the schema tree, there is no YANG depiction.

The YANG code for the grouping statement from inet-types is shown below.

```
import ietf-inet-types {
     prefix "inet";
}

     grouping endpoint {
         description "A reusable endpoint group.";
         leaf ip {
             type inet:ip-address;
         }
         leaf port {
             type inet:port-number;
         }
     }
```

Table 15 shows the mapping of the *grouping* statement's sub-statements.

Table 15 – Grouping statement mapping

Statement	Mapping
choice	See clause 6.9
container	See clause 6.4.4
description	See clause 6.25
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
uses	See clause 6.12

6.12 Uses statement

From RFC6020, the "uses" statement is used to reference a "grouping" definition. It takes one argument, which is the name of the grouping.

The *grouping* and *uses* statement should be handled and resolved prior to mapping the YANG to CSDL. Since the *grouping* statement does not define a node in the schema tree, there is no YANG depiction.

The YANG code shown below, uses the "endpoint" grouping defined in clause 6.11 in a definition of an HTTP server in some other module.

```
     import acme-system {
         prefix "acme";
     }

     container http-server {
         leaf name {
             type string;
         }
         uses acme:endpoint;
     }
```

Table 16 shows the mapping of the *uses* statement's sub-statements.

Table 16 – Uses statement mapping

Statement	Mapping
augment	See clause 6.15
description	See clause 6.25
if-feature	See clause 6.20
refine	See clause 6.12.1
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

6.12.1 Refine statement

From RFC6020, some of the properties of each node in the grouping can be refined with the "refine" statement. The argument is a string that identifies a node in the grouping. This node is called the refine's target node.

The preprocessor should which resolves to uses statement should also resolve the refine statement.

In the above example, if port 80 should be the default for the HTTP server, default can be added as a refinement.

```
     container http-server {
        leaf name {
            type string;
        }
        uses acme:endpoint {
            refine port {
                default 80;
            }
        }
     }
```

6.13 Rpc statement

From RFC6020, the "rpc" statement is used to define a NETCONF RPC operation. It takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed rpc information.

The *rpc* statement is mapped to a CSDL Action. The NETCONF RPC semantics are replaced by the Redfish action semantics. Note, parameters can be complex

6.13.1 Mapping YANG code to Redfish CSDL

From the purpose of illustration, an XML instance example for rpc is shown below, from RFC6020.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <rock-the-house xmlns="http://example.net/rock">
         <zip-code>27606-0100</zip-code>
     </rock-the-house>
</rpc>
```

The YANG code for the above rpc example is shown below.

```
rpc rock-the-house {
      input {
          leaf zip-code {
              type string;
          }
      }
}
```

The resultant CSDL fragment is shown below.

```
<Action Name="rock-the-house" IsBound="true">
   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/rpc"/>
   <Parameter Name="rock_the_houseInput" Type="rock_the_houseInputType">
      <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/input"/>
   </Parameter>
</Action>

<ComplexType Name="rock_the_houseInputType" >
     <Property Name="zip code" Type="Edm.String"/>
</ComplexType>
```

Table 17 shows the mapping of the *rpc* statement's sub-statements.

Table 17 – Rpc statement mapping

YANG	Redfish JSON and CSDL
description	See clause 6.25
grouping	See clause 6.11
if-feature	See clause 6.20
input	See clause 6.13.2
	<Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/input"/>
output	See clause 6.13.3
	<Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/output"/>
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1

6.13.2 Input statement

See clause 6.13.1, which includes the *input* statement in the discussion.

The value of Name attribute is synthesized by appending the string "Input" to the value of the *rpc* statement. The value of the Type attribute is synthesized by appending the string "InputType" to the value of the *rpc* statement.

The "input type" shall be declared as a ComplexType.

6.13.3 Output statement

See clause 6.13.1, which includes the *output* statement in the discussion.

The value of Name attribute is synthesized by appending the string "Output" to the value of the *rpc* statement. The value of the Type attribute is synthesized by appending the string "OutputType" to the value of the *rpc* statement.

The "output type" shall be declared as a ComplexType.

```
rpc rock-the-house {
     input {
         leaf zip-code {
             type string;
         }
     }
     output {
         leaf volume {
             type int16;
         }
     }
}
```

The value of Name attribute is synthesized by appending the string "Output" to the value of the *output* statement.

```
<Action Name="rock-the-house" IsBound="true">
   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/rpc"/>
  . . .
   <ReturnType Name="rock_the_houseOutput" Type="rock_the_houseOutputType">
      <Annotation Term="Redfish.Yang.NodeType"
            EnumMember ="Redfish.Yang.NodeTypes/output"/>
   </ReturnType>
</Action>

<ComplexType Name="rock_the_houseOutputType" >
     <Property Name="volume" Type="Edm.int16"/>
</ComplexType>
```

6.14 Notification statement

From RFC6020, the "notification" statement is used to define a NETCONF notification. It takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed notification information.

The *notification* statement is mapped to an EntityType.

6.14.1 Mapping YANG code to Redfish CSDL

From the purpose of illustration, an XML instance example of a notification is shown below, from RFC6020.

```
     <notification
       xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
       <eventTime>2008-07-08T00:01:00Z</eventTime>
       <event xmlns="http://example.com/event">
         <event-class>fault</event-class>
         <reporting-entity>
           <card>Ethernet0</card>
         </reporting-entity>
         <severity>major</severity>
       </event>
     </notification>
```

The YANG code for the above notification example is shown below.

```
notification event {
     leaf event-class {
         type string;
     }
     anyxml reporting-entity;
     leaf severity {
         type string;
     }
}
```

The resultant CSDL fragment is shown below.

```
<EntityType Name="event" BaseType="Resource.1.0.0.Resource">
  <Annotation Term="Redfish.Yang.NodeType"
    EnumMember ="Redfish.Yang.NodeTypes/notification"/>
  <Annotation Term="OData.Description" String=""/>
  <Annotation Term="OData.AdditionalProperties" Bool="false"/>
  <Property Name="event-class", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
  <Property Name="severity", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
  <Property Name="reporting-entity", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/anyxml"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
</EntityType>
```

Table 18 shows the mapping of the *notification* statement's sub-statements.

Table 18 – Notification statement mapping

YANG	Redfish JSON and CSDL
anyxml	See clause 6.10
choice	See clause 6.9
description	See clause 6.25
grouping	See clause 6.11
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
uses	See clause 6.12

6.15 Augment statement

From RFC6020, The "augment" statement allows a module or submodule to add to the schema tree defined in an external module, or the current module and its submodules, and to add to the nodes from a grouping in a "uses" statement. The argument is a string that identifies a node in the schema tree.

The augment statement is treated as a pre-processor directive. The resulting CSDL contains the superset of augmentations, and also an annotation which indications what was augmented.

For example, the following examples show the *augment* statement, which augments the interfaces container.

The following is an *augment* statement for an "interfaces" container.

```
     container interfaces {
        list ifEntry {
            key "ifIndex";

            leaf ifIndex {
                type uint32;
            }
            leaf ifDescr {
                type string;
            }
            leaf ifType {
                type iana:IfType;
            }
            leaf ifMtu {
                type int32;
            }
        }
    }
```

The following is an *augment* statement that augments the ifEntry *list* statement. In example, there is a conditional *when* statement associated with the augment.

```
     import interface-module {
        prefix "if";
    }
    augment "/if:interfaces/if:ifEntry" {
        when "if:ifType='ds0'";
        leaf ds0ChannelNumber {
            type ChannelNumber;
        }
    }
```

The resultant CSDL is shown below. Note if the augment statement adds more than one entry, then the CSDL for each entry contains the augment annotation and conditional annotation.

```
<Property Name="ifMtu" Type="Redfish.Yang.int32">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
     <Annotation Term="Redfish.Yang.YangType" String="int32"/>
     . . .
</Property>
<Property Name="ds0ChannelNumber" Type="Redfish.Yang.int32">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
     <Annotation Term="Redfish.Yang.augment" String="if:interfaces/if:ifEntry">
         <Annotation Term="Redfish.Yang.when" String=" if:ifType='ds0'"/>
     </Annotation>
     <Annotation Term="Redfish.Yang.YangType" String="int32"/>
     . . .
</Property>
```

Table 19 shows the mapping of the *augment* statement and its sub-statements.

1534 **Table 19 – Augment statement mapping**

YANG	Redfish JSON and CSDL
case	See clause 6.9.3
choice	See clause 6.9
description	See clause 6.25
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
uses	See clause 6.12
when	See clause 6.27

1535 ## 6.16 Identity statement

1536 From RFC6020, the "identity" statement is used to define a new globally unique, abstract, and untyped
1537 identity. Its only purpose is to denote its name, semantics, and existence. An identity can either be
1538 defined from scratch or derived from a base identity.

1539 There is no YANG depiction of an *identity* statement.

1540 The *identity* statement is mapped to a complex annotation.

1541 Identity results in ComplexType (see system RFC)

1542 ### 6.16.1 Mapping YANG code to Redfish CSDL

1543 The general *identity* statement is shown below.

1544
```
identity [identity value]
```

1545 The resultant CSDL is shown below.

1546
```
<ComplexType Name="[identity value]">
     <Annotation Term="Redfish.Yang.identity" String="[identity value]"/>
     <Annotation Term="Redfish.Yang.description" String=". . ."/>
</ComplexType>
```

1550 The *identity* statement from RFC7317 is shown below.

1551
```
identity authentication-method {
     description "Base identity for user authentication methods.";
}
```

1554 The resultant CSDL is shown below.

1555
```
<ComplexType Name="authentication method">
     <Annotation Term="Redfish.Yang.identity" String="authentication-method"/>
     <Annotation Term="Redfish.Yang.description"
         String="Base identity for user authentication methods."/>
</ComplexType>
```

1560 Table 20 shows the mapping of the *Identity* statement's sub-statements.

Table 20 – Identity statement mapping

Statement	Mapping
base	baseType = "the base identity string". See clause 6.16.2
	<Annotation Term="Redfish.Yang.base" String="text from the yang description statement"/>
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24

6.16.2 Base statement

The "base" statement is optional and takes as an argument a string that is the name of an existing identity, from which the new identity is derived. If no "base" statement is present, the identity is defined from scratch.

See clause 6.16.1 for the mapping.

6.17 Extension statement

From RFC6020, the "extension" statement allows the definition of new statements within the YANG language. This new statement definition can be imported and used by other modules.

The *extension* statement is mapped to an annotation. The extended statement is placed in a Redfish.Yang.statement annotation, in which the string attribute contains the entire YANG statement

```
<Annotation Term="Redfish.Yang.extension" String="[text from extension statement]"/ >
```

The general extension statement is shown below. Note that the extended statement, along with its value, is also shown.

```
Extension [extended statement] {
     argument [argument value]
}

[extended statement] [extended value];
```

The resultant CSDL is shown below. The extended statement and value are used in the Annotation for Redfish.Yang.statement.

```
<EntityType …>
     <Annotation Term="Redfish.Yang.extension" String="[extended statement]">
         <Annotation Term="Redfish.Yang.argument" String="[argument value]"/>
     </Annotation>
     <Annotation Term="Redfish.Yang.statement"
             String="[extended statement] [extended value]"/>
</EntityType>
```

Example YANG code for the *extension* statement from the MPLS OpenConfig RFC is shown below.

```
extension openconfig-version {
     argument "semver" {
         yin-element false;
     }
}

openconfig-version 6;
```

The resultant Redfish CSDL is shown below.

```
<EntityType …>
     <Annotation Term="Redfish.Yang.extension" String="openconfig-version">
         <Annotation Term="Redfish.Yang.argument" String="semver">
             <Annotation Term="Redfish.Yang.yin_element"
                 EnumMember="Redfish.Yang.YinElement/false"/>
         </Annotation>
     </Annotation>
     <Annotation Term="Redfish.Yang.statement" String="openconfig-version 6"/>
</EntityType>
```

Table 21 shows the mapping of the *Extension* statement's sub-statements.

Table 21 – Extension statement mapping

Statement	Mapping
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24

6.18 Argument statement

From RFC6020, the "argument" statement, which is optional, takes as an argument a string that is the name of the argument to the keyword.

The *argument* statement is mapped to the annotation, which is within the annotation of its parent statement.

```
<Annotation Term="Redfish.Yang.argument" String="[value of argument statement]"/ >
```

The YANG code for the argument statement from the MPLS OpenConfig RFC is shown below.

```
extension [extension value] {
     argument [augument value] {
         . . .
     }
}
```

The resultant Redfish CSDL is shown below.

```
<Annotation Terem=Redfish.Yang.extension" String="[extension value]: >
     <Annotation Term="Redfish.Yang.argument" String="[augument value]" >
         . . .
     </Annotation>
</Annotation>
```

Example YANG code for the argument statement from the MPLS OpenConfig RFC is shown below.

```
extension openconfig-version {
     argument "semver" {
         yin-element false;
     }
}
```

The resultant Redfish CSDL is shown below.

```
<Annotation Term="Redfish.Yang.extension" String="openconfig-version">
     <Annotation Term="Redfish.Yang.argument" String="semver">
         <Annotation Term="Redfish.Yang.yin element"
             EnumMember="Redfish.Yang.YinElement/false"/>
     </Annotation>
</Annotation>
```

Table 22 shows the mapping of the *argument* statement's sub-statements.

1642 **Table 22 – Argument statement mapping**

Statement	Mapping
yin-element	One of <Annotation Term="Redfish.Yang.yin_element" EnumMember="Redfish.Yang.YinElement/false"/> <Annotation Term="Redfish.Yang.yin_element" EnumMember="Redfish.Yang.YinElement/true"/>

1643 ## 6.19 Feature statement

1644 From RFC6020, the "feature" statement is used to define a mechanism by which portions of the schema
1645 are marked as conditional. A feature name is defined that can later be referenced using the "if-feature"
1646 statement.

1647 The *feature* statement is mapped to the annotation, within the scope of its parent statement.

1648
```
<Annotation Term="Redfish.Yang.feature" String="[value of feature statement]"/ >
```

1649 The YANG code for the *feature* statement from RFC 7277 is shown below.

1650
```
feature ipv4-non-contiguous-netmasks {
      description "Indicates support for configuring non-contiguous subnet masks.";
}
```

1653 Table 23 shows the mapping of the *Feature* statement.

1654 **Table 23 – Feature statement mapping**

YANG	Redfish JSON and CSDL
if-feature	See clause 6.20
status	See clause 6.24
reference	See clause 6.26

1655 ## 6.20 If-feature statement

1656 From RFC6020, the "if-feature" statement makes its parent statement conditional. The argument is the
1657 name of a feature, as defined by a "feature" statement.

1658 The *if-feature* statement is mapped to the annotation, within the annotation of its parent statement.

1659
```
<Annotation Term="Redfish.Yang.if-feature" String="[value of if-feature statement]"/ >
```

1660 The YANG code for the *if-feature* statement from RFC 7277 is shown below.

1661
```
          leaf create-temporary-addresses {
            if-feature ipv6-privacy-autoconf;
            type boolean;
            default false;
            description
              "If enabled, the host creates temporary addresses as
               described in RFC 4941.";
            reference
              "RFC 4941: Privacy Extensions for Stateless Address
                         Autoconfiguration in IPv6";
          }
```

The resultant Redfish CSDL is shown below.

```
<Property Name="create_temporary_address" Type="Redfish.Yang.boolean">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
     <Annotation Term="Redfish.Yang.YangType" String="boolean"/>
     <Annotation Term="Redfish.Yang.if-feature" String="ipv6-privacy-autoconf"/>
     . . .
</Property>
```

There are no sub-statements specified for the *if-feature* statement.

6.21 Deviation statement

From RFC6020, the "deviation" statement defines a hierarchy of a module that the device does not implement faithfully. The argument is a string that identifies the node in the schema tree where a deviation from the module occurs.

The *deviation* statement is mapped to the annotation.

```
<Annotation Term="Redfish.Yang.deviation" String="[value of deviation statement]"/ >
```

The YANG code for the *deviation* statement is shown below.

```
module [module value] {
   deviation [deviation value] {
         deviate [deviate value] {
             . . .
         }
   }
}
```

The resultant CSDL is show below.

```
<EntityType Name="[module value]" BaseType="Resource.v1_0_0.Resource">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/module"/>
     <Annotation Term="Redfish.Yang.deviation" String="[deviation value]"/ >
         <Annotation Term="Redfish.Yang.deviate" String="[deviate value]"/ >
     </Annotation>
     . . .
</EntityType>
```

The YANG code for the *deviation* statement is shown below.

```
   deviation /base:system/base:user/base:type {
         deviate add {
             default "admin"; // new users are 'admin' by default
         }
     }
```

The resultant CSDL is show below.

```
<EntityType Name="<module value>" BaseType="Resource.v1_0_0.Resource">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/module"/>
     <Annotation Term="Redfish.Yang.deviation"
             String="/base:system/base:user/base:type" >
         <Annotation Term="Redfish.Yang.deviate" String="add" />
     </Annotation>
     . . .
</EntityType>
```

Table 24 shows the mapping of the *deviation* statement's sub-statements.

1718 **Table 24 – Deviation statement mapping**

Statement	Mapping
description	See clause 6.25
deviate	See clause 6.22
reference	See clause 6.26

1719 ## 6.22 Deviate statement

1720 From RFC6020, The "deviate" statement defines how the device's implementation of the target node
1721 deviates from its original definition. The argument is one of the strings "not-supported", "add", "replace",
1722 or "delete".

1723 See clause 6.21 which shows the mapping of the *deviate* statement

1724 Table 25 shows the mapping of the *deviate* statement's sub-statements.

1725 **Table 25 – Deviate statement mapping**

Statement	Mapping
config	See clause 6.23
default	<Annotation Term="Redfish.Yang.default" String="text from default statement"/>
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
max-element	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-element	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
type	See clause 6.4
unique	<Annotation Term="Redfish.Yang.unique" String="text from unique statement"/>
units	<Annotation Term="Redfish.Yang.units" String="text from unit statement"/>

1726 ## 6.23 Config statement

1727 From RFC6020, the "config" statement takes as an argument the string "true" or "false". If "config" is
1728 "true", the definition represents a configuration.

1729 The config statement is mapped to one of two annotations.

1730 `<Annotation Term="Redfish.Yang.config" EnumMember="Redfish.Yang.ConfigPermission/false"/>`
1731 `<Annotation Term="Redfish.Yang.config" EnumMember="Redfish.Yang.ConfigPermission/true"/>`

The YANG code for a *leaf* statement is shown below.

```
leaf clientRequestCount {
     description "Client Request Count";
     type uint32;
     config "false";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="clientRequestCount" Type="Redfish.Yang.uint32">
     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
     <Annotation Term="Redfish.Yang.YangType" String="uint32"/>
     <Annotation Term="Redfish.Yang.config"
         EnumMember="Redfish.Yang.ConfigPermission/false"/>
</Property>
```

The *config* statement has no sub-statements.

6.24 Status statement

From RFC6020, the "status" statement takes as an argument one of the strings "current", "deprecated", or "obsolete".

The *status* statement is mapped to one of three annotations.

```
<Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/current"/ >
<Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/deprecated" />
<Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/obsolete" />
```

The YANG code for a *status* statement from RFC 7224 is shown below

```
identity iso88023Csmacd {
     base iana-interface-type;
     status deprecated;
     description "…";
     reference "…";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="iso88023Csmacd" Type="…">
     <Annotation Term="Redfish.Yang.status" String="deprecated"/>
     . . .
</Property>
```

The *status* statement has no sub-statements.

6.25 Description statement

From RFC6020, the "description" statement takes as an argument a string that contains a human-readable textual description of this definition.

The *description* statement is mapped to the annotation.

```
<Annotation Term="Redfish.Yang.description" String="[value of description statement]"/ >
<Annotation Term="OData.Description" String="[value from description statement]"/>
```

The YANG code for a *description* statement from RFC 7224 is shown below

```
identity iso88023Csmacd {
     base iana-interface-type;
     status deprecated;
     description "Deprecated via RFC 3635. Use ethernetCsmacd(6) instead";
     reference "…";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="iso88023Csmacd" Type="…">
     <Annotation Term="Redfish.Yang.description"
         String="Deprecated via RFC 3635. Use ethernetCsmacd(6) instead"/>
     <Annotation Term="OData.Description"
         String="Deprecated via RFC 3635. Use ethernetCsmacd(6) instead"/>
     . . .
</Property>
```

The *description* statement has no sub-statements.

Note: The string for the LongDescription annotation is constructed from the *reference* statement. The construction adds normative text to the value of the *reference* statement, such as "The element shall …".

6.26 Reference statement

The "reference" statement takes as an argument a string that is used to specify a textual cross-reference to an external document, either another module that defines related management information, or a document that provides additional information relevant to this definition.

The *reference* statement is mapped to an annotation.

```
<Annotation Term="Redfish.Yang.reference" String="[value of reference statement]"/ >
```

The YANG code for a *reference* statement from RFC 7224 is shown below

```
identity iso88023Csmacd {
     base iana-interface-type;
     status deprecated;
     description "…";
     reference "RFC 3635 - Definitions of Managed Objects for the Ethernet-like Interface
Types";
}
```

The resultant CSDL fragment for the JSON properties is shown below.

```
<Property Name="iso88023Csmacd" Type="…">
     <Annotation Term="Redfish.Yang.reference"
         String=" RFC 3635 - Definitions of Managed Objects for the Ethernet-like Interface
Types"/>
     . . .
</Property>
```

The *reference* statement has no sub-statements.

6.27 When statement

From RFC6020, the "when" statement makes its parent data definition statement conditional. The node defined by the parent data definition statement is only valid when the condition specified by the "when" statement is satisfied. The statement's argument is an XPath expression, which is used to formally specify this condition.

The *when* statement is mapped to the annotation.

```
<Annotation Term="Redfish.Yang.when" String="[value of when statement]"/ >
```

See clause 6.15 for an example of the YANG to CSDL mapping.

The *when* statement has no sub-statements.

6.28 Unmapped YANG statements

If YANG code is read which does not conform the statement format, then the following annotation should be added to the resultant CSDL. This will indicate that the original YANG file should be reviewed and the source of the "statement" annotation be found.

<Annotation Term="Redfish.Yang.statement" String="text from the yang statement"/>

An example of YANG code from RFC 7317, which may cause a "statement" annotation is shown below. The nacm:default-deny-all line does not follow the statement format.

```
rpc system-restart {
      nacm:default-deny-all;
      description
        "Request that the entire system be restarted immediately.
         A server SHOULD send an rpc reply to the client before
         restarting the system.";
}
```

1836

ANNEX A
(**informative**)

Change log

Version	Date	Description
0.1.0a	05/10/2016	Initial draft
0.2.0	05/23/2016	Incorporate the mapping from the Visio diagrams
0.3.0	05/25/2016	Clean up. Use RFC 6020 for ordering clauses. Rewrite "Lists" mapping to correspond to the DHCP collection resource construct.
0.4.0	05/29/2016	Added clauses for each YANG statement. Add cross-references in tables.
0.5.0	05/30/2016	Added clauses for each YANG sub-statement
0.5.1	06/05/2016	Revised based on June 2-3 meetings. Add examples.
0.5.2	05/06/2016	Minor fixes
0.5.3		Commented open issues.
0.5.6	10/13/2016	Modifications from the June 14 F2F review