Document Identifier: DSP2052

Date: 2018-04-05

Version: 1.0.0

Redfish and OData White Paper

Document Class: Informative

Document Status: Published

Document Language: en-US

CONTENTS

1. Introduction 6
2. Schema files 6
2.1. CSDL format 6
2.1.1. The Property element 7
2.1.2. The NavigationProperty element 8
2.1.3. The Collection type 8
2.1.4. The EnumType element 9
2.1.5. The ComplexType element 10
2.1.6. The EntityType element 11
2.1.7. The Action element 12
2.1.8. The Annotation element 14
2.1.9. Inheritance 15
2.2. Redfish modeling practices 16
2.2.1. Core Redfish definitions 16
2.2.2. Defining Redfish resources 16
2.2.3. Referenceable Members 19
2.2.4. Schema versioning 20
2.3. CSDL vs. JSON Schema 22
3. Payload annotations 22
3.1. Annotating a single property in a response 22
3.2. Annotating an object in a response 23
4. OData service document 24
5. Metadata document 25
6. Appendix 26
6.1. Primitive OData types used in Redfish 26
6.2. Schema annotations used in Redfish 27
6.2.1. Core annotations defined by OData 27
6.2.2. Redfish annotations defined in RedfishExtensions_v1.xml 28
6.3. Sample OData service document 29
6.4. Sample metadata document 30
6.5. References 34

Foreword

The Redfish and OData White Paper was prepared by the Redfish Forum of the DMTF.

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:

- Jeff Hilland - Hewlett Packard Enterprise
- Michael Raineri - Dell Inc.

1. Introduction

Redfish is a management standard using a data model representation inside of a hypermedia RESTful interface. It adheres to the OData v4 standard for defining schema and payload formats. This was done in order to allow off-the-shelf OData clients to interact natively with Redfish services. While the Redfish Specification only calls out a minimal set of OData functionality, implementations are allowed to extend their capabilities into the full range of OData support though doing so is outside the scope of Redfish and may provide interoperability challenges with non-OData clients. This white paper will provide details about how the Redfish Specification conforms to OData, such as how schema files are constructed and how services can construct required OData resources. For those interested in OData functionality that is outside the scope of Redfish, refer to the OData documentation link in the References section.

2. Schema files

Redfish defines its payload definitions in the Common Schema Definition Language (CSDL) as defined by OData v4. CSDL is designed to allow for clients to dynamically scan and adapt to a service's data model. It also provides documentation for developers when writing purpose built clients. Redfish CSDL files are written in XML, and the structures in the XML file define the JSON properties and objects that a service uses in its payloads. Inline annotations are also used to provide clients and users with more detailed information about a given property or object.

2.1. CSDL format

The primary body of a schema file contains namespace definitions; this is found between the `<edmx:DataServices>` tags. A namespace is a unique name for a set of type definitions being declared, which include things like enum definitions and JSON objects. Multiple namespaces can be defined in a single file, and they can reference each other's definitions. Type definitions are referenced as `Namespace.TypeDefinition`, where `Namespace` is the string name of the namespace, and `TypeDefinition` is the name of the definition being referenced.

If a schema file requires references to namespaces defined in other schema files, a reference to the namespace must be included. This is typically done at the top of the document within a `<edmx:Reference>` section using an `<edmx:Include>` statement. The reference includes the URI of the schema file being referenced in addition to which namespaces in the schema file to include. Primitive types defined by OData, which begin with `Edm.`, do not need additional files to be included.

Below is a sample schema file showing the general format discussed above. In the example below, there are references to the external file "ExternalSchema.xml", which is calling out references to two Namespaces: `ExternalNamespace` and `Other.Namespace`. In the DataServices section, one Namespace is defined: `MyNewNamespace`. There is a single ComplexType definition called

`MyDataType`, and it contains three properties: `MyProperty`, `MyProperty2`, and `MyProperty3`. `MyProperty` and `MyProperty2` both reference external definitions found in the `ExternalNamespace` and `Other.Namespace` namespaces. Those definitions would be found by going into the ExternalSchema.xml file. `MyProperty3` has the type set to `Edm.Int64`, which is simply a 64-bit integer. The following sections will describe the different elements found in the namespace definition in detail.

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

  <edmx:Reference Uri="http://contoso.org/schemas/ExternalSchema.xml">
    <edmx:Include Namespace="ExternalNamespace"/>
    <edmx:Include Namespace="Other.Namespace"/>
  </edmx:Reference>

  <edmx:DataServices>

    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="MyNewNamespace">

      <ComplexType Name="MyDataType">
        <Property Name="MyProperty" Type="ExternalNamespace.ReferencedDataType"/>
        <Property Name="MyProperty2" Type="Other.Namespace.OtherDataType"/>
        <Property Name="MyProperty3" Type="Edm.Int64"/>
      </ComplexType>

    </Schema>

  </edmx:DataServices>

</edmx:Edmx>
```

2.1.1. The Property element

The `<Property>` element is used to define a property inside of a JSON object. It provides the name of the property and the data type of the property.

In the CSDL sample shown below, a Property named `SerialNumber` is defined, and the type is `Edm.String`. This defines a name-value pair in a JSON object where the name is `SerialNumber` and the data type to expect for that name is a string.

CSDL sample:

```
  <Property Name="SerialNumber" Type="Edm.String"/>
```

The JSON representation for the above CSDL sample is shown below. It contains a JSON object with a

property that has the name `SerialNumber` and its value is the string `123456789`.

JSON representation:

```
{
    "SerialNumber": "123456789",
    ...
}
```

2.1.2. The NavigationProperty element

The `<NavigationProperty>` element is used to define a property inside of a JSON object that provides a link to another resource within the service. It provides the name of the property and the data type of the resource it links.

In the CSDL sample shown below, a NavigationProperty named `Thermal` is defined, and the data type of the resource it links will follow the `Thermal.Thermal` definition.

CSDL sample:

```
<NavigationProperty Name="Thermal" Type="Thermal.Thermal"/>
```

In JSON, shown below, this is represented as a property named `Thermal` whose value contains an object with the property `@odata.id`. The value of the `@odata.id` property is a URI to the resource being linked. In this case, the URI is `/redfish/v1/Chassis/1/Thermal`, so a client can expect to receive a payload conforming to the `Thermal.Thermal` definition if they perform a GET on that URI.

JSON representation:

```
{
    "Thermal": {
        "@odata.id": "/redfish/v1/Chassis/1/Thermal"
    },
    ...
}
```

2.1.3. The Collection type

When defining Property or NavigationProperty elements, the keyword `Collection` can be used to turn the Property or NavigationProperty element into an array. This is done in the Type field using the format

`Collection(TypeDefinition)`, where `TypeDefinition` is the underlying type of the instances in the array.

It should be noted that a "CSDL Collection" should not be confused with a "Resource Collection". In JSON terms, a "CSDL Collection" is a JSON array, and a "Resource Collection" is a JSON object that contains a set of links to "Resources" of a given type. The Defining Redfish resources section contains more information about "Resource Collections".

In the CSDL sample shown below, a Property named `AllowedSpeedsMHz` is defined, and the type is `Collection(Edm.Int64)`. This means that the value for the property `AllowedSpeedsMHz` in a JSON payload will be an array of 64-bit integers.

CSDL sample:

```
<Property Name="AllowedSpeedsMHz" Type="Collection(Edm.Int64)"/>
```

The JSON representation for the above CSDL sample is shown below. It contains a JSON object with a property that has the name `AllowedSpeedsMHz` and its value is an array containing the numbers 2133, 2400, and 2667.

JSON representation:

```
{
    "AllowedSpeedsMHz": [
        2133,
        2400,
        2667
    ],
    ...
}
```

2.1.4. The EnumType element

The `<EnumType>` element is used to define a set of valid values for a given property. Within the EnumType definition, a set of Members define the string values that are allowed when using the EnumType. An EnumType definition is referenced by a Property definition using the Type field for the property.

In the first sample shown below, an EnumType called `IndicatorLED` is defined as part of the namespace called `Resource.v1_1_0`. Within that definition are three Members: `Lit`, `Blinking`, and `Off`. Those three Members are the three string values that a service is allowed to use for that EnumType. The second sample shows how to use that EnumType with a Property. In this case, we have a Property

defined called `IndicatorLED`, and it's referencing the `IndicatorLED` definition found in the `Resource.v1_1_0` Namespace via the Type field.

CSDL sample:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Resource.v1_1_0">
  <EnumType Name="IndicatorLED">
    <Member Name="Lit"/>
    <Member Name="Blinking"/>
    <Member Name="Off"/>
  </EnumType>
</Schema>
```

EnumType usage sample:

```
<Property Name="IndicatorLED" Type="Resource.v1_1_0.IndicatorLED"/>
```

When representing the above property in JSON, this means that when the service provides the `IndicatorLED` property in its JSON object, it must return one of the three values specified by the `IndicatorLED` EnumType definition. An example of this is shown below.

JSON representation:

```
{
    "IndicatorLED": "Blinking",
    ...
}
```

2.1.5. The ComplexType element

The `<ComplexType>` element is used to define a JSON object. Inside of the ComplexType definition, there will be Property and NavigationProperty elements that describe the different properties that will be found inside of the JSON object.

In the first sample shown below, a ComplexType called `ProcessorId` is defined as part of the Namespace called `Processor.v1_0_0`. Within that definition are six Property elements named `VendorId`, `IdentificationRegisters`, `EffectiveFamily`, `EffectiveModel`, `Step`, and `MicrocodeInfo`, all of which are strings. The second sample shows how to use that ComplexType with a Property. In this case, we have a Property defined called `ProcessorId`, and it's referencing the `ProcessorId` definition found in the `Processor.v1_0_0` Namespace via the Type field.

CSDL sample:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Processor.v1_0_0">
  <ComplexType Name="ProcessorId">
    <Property Name="VendorId" Type="Edm.String"/>
    <Property Name="IdentificationRegisters" Type="Edm.String"/>
    <Property Name="EffectiveFamily" Type="Edm.String"/>
    <Property Name="EffectiveModel" Type="Edm.String"/>
    <Property Name="Step" Type="Edm.String"/>
    <Property Name="MicrocodeInfo" Type="Edm.String"/>
  </ComplexType>
</Schema>
```

ComplexType usage sample:

```
<Property Name="ProcessorId" Type="Processor.v1_0_0.ProcessorId"/>
```

A service represents the above structure in a payload with the property `ProcessorId`, and the value is an object containing the six properties specified in the ComplexType definition. An example of this is shown below.

JSON representation:

```
{
    "ProcessorId": {
        "VendorId": "GenuineIntel",
        "IdentificationRegisters": "0x34AC34DC8901274A",
        "EffectiveFamily": "0x42",
        "EffectiveModel": "0x61",
        "Step": "0x1",
        "MicrocodeInfo": "0x429943"
    },
    ...
}
```

2.1.6. The EntityType element

The `<EntityType>` element is used to define a JSON object while also defining a uniquely identifiable key for that object. Inside of the EntityType definition, there will be Property and NavigationProperty elements that describe the different properties that will be found inside of the JSON object. Within Redfish, the EntityType definitions are used to define the Redfish resources.

In the CSDL sample shown below, an EntityType named `Processor` is defined. Within the definition are four Property elements: `Id`, `Name`, `MaxSpeedMhz`, and `TotalCores`. `Id` and `Name` are both strings, and `MaxSpeedMhz` and `TotalCores` are both 64-bit integers. Using the `<Key>` element, the Property named `Id` is established to be the key. This means that if there are a set of Processor instances, the `Id` property must be a unique value amongst the individual Processor instances.

CSDL sample:

```
<EntityType Name="Processor">
  <Key>
    <PropertyRef Name="Id"/>
  </Key>
  <Property Name="Id" Type="Edm.String"/>
  <Property Name="Name" Type="Edm.String"/>
  <Property Name="MaxSpeedMhz" Type="Edm.Int64"/>
  <Property Name="TotalCores" Type="Edm.Int64"/>
</EntityType>
```

A service represents the above definition as a JSON object with four properties: `Id`, `Name`, `MaxSpeedMhz`, and `TotalCores`. An example of this is shown below.

JSON representation:

```
{
    "Id": "CPU0",
    "Name": "Processor in Socket 0",
    "MaxSpeedMhz": 2000,
    "TotalCores": 16
}
```

2.1.7. The Action element

The `<Action>` element is used to define an operation that a client can perform by submitting a POST request to the URI specified by the Action. As part of the definition, the parameters for the Action are established. A service advertises supported Actions for a given resource by supplying the information as part of the response to a GET on the resource.

In the CSDL sample shown below, an Action named `Reset` is defined. The `IsBound` facet is set to `true` in the Action definition; this means that this Action has an association with a particular resource. The first parameter is the binding parameter, which shows this Action is being bound to the `Actions` object for a `Manager` instance. As a matter of convention, Redfish only uses bound Actions, and they are bound to the `Actions` object for a given resource. The second parameter is named `ResetType`, and is an enum

defined by `ResetType` in the `Resource` Namespace.

CSDL sample:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Manager">
  <Action Name="Reset" IsBound="true">
    <Parameter Name="Manager" Type="Manager.v1_0_0.Actions"/>
    <Parameter Name="ResetType" Type="Resource.ResetType"/>
  </Action>
</Schema>
```

When a client performs a GET on the Manager instance with the Action, the service responds with the Action representation in the `Actions` object, which is shown below. The Action itself is represented as a JSON object with the name in the format `#Namespace.ActionName`, where `Namespace` is the string name of the Namespace where the Action is defined, and `ActionName` is the name of the Action. In this case, because the CSDL definition shows the Action is named `Reset` and is within the `Manager` Namespace, the property name used is `#Manager.Reset`. Inside of the object a property named `target`, which shows the URI the client uses in the POST request to perform the Action; in this case the URI is `/redfish/v1/Managers/1/Actions/Manager.Reset`. The object also contains payload annotations to help the client identify constraints on the parameters; in this case, it shows the client is allowed to submit requests with the `ResetType` parameter set to `On`, `ForceOff`, `GracefulShutdown`, `GracefulRestart`, `ForceRestart`, or `ForceOn`. These annotations are discussed further in the Payload annotations section.

JSON representation:

```
{
    "Actions": {
        "#Manager.Reset": {
            "target": "/redfish/v1/Managers/1/Actions/Manager.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOff",
                "GracefulShutdown",
                "GracefulRestart",
                "ForceRestart",
                "ForceOn"
            ]
        }
    },
    ...
}
```

Using the above information, if a client wants to perform a `Reset` of the Manager by using a `GracefulRestart`, it will submit a POST request to the URI given in `target`, and the body of the request will contain a JSON payload that contains the property `ResetType` with the value `GracefulRestart`. An example of this is shown below.

Client POST sample:

```
POST /redfish/v1/Managers/1/Actions/Manager.Reset HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
    "ResetType": "GracefulRestart"
}
```

2.1.8. The Annotation element

The `<Annotation>` element is used to provide inline documentation for anything defined in the schema file. Annotation elements give guidance to developers, and can also express conformance rules for clients and services. Annotation elements contain a `Term` to describe what type of annotation is being used, and sometimes contains data to go along with it. Redfish uses only two types of annotations: those defined in OData and those defined by Redfish. OEM annotations are not allowed.

In the CSDL example below, the Property `UserName` contains three Annotations: `Redfish.RequiredOnCreate`, `OData.Permissions`, and `OData.Description`. The first Annotation contains the term `Redfish.RequiredOnCreate`; it contains no data, but its presence indicates that a client is required to supply the `UserName` property when creating a new resource. The second Annotation contains the term `OData.Permissions`, which has the enum value `OData.Permission/ReadWrite` to indicate that `UserName` can be read and written by a client. The third Annotation contains the term `OData.Description`, which contains a string description of what this Property represents.

```
  <Property Name="UserName" Type="Edm.String">
    <Annotation Term="Redfish.RequiredOnCreate"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permission/ReadWrite"/>
    <Annotation Term="OData.Description" String="This property contains the user name
for the account."/>
  </Property>
```

2.1.9. Inheritance

EntityType and ComplexType elements are both allowed to use a `BaseType` in their definition. The value for the BaseType is the name of the EntityType or ComplexType in which the new type is referencing. It's not possible to mix EntityType or ComplexType references; the BaseType value in an EntityType must reference another EntityType, and likewise for a ComplexType definition. All properties defined by the BaseType become available to the newly defined type.

In the CSDL sample below, two ComplexType elements are defined: `Protocol` and `SSDProtocol`. `Protocol` contains the Property elements `ProtocolEnabled` and `Port`, and `SSDProtocol` contains the Property elements `NotifyMulticastIntervalSeconds` and `NotifyTTL`. `SSDProtocol` is defined with the BaseType set to `ManagerNetworkProtocol.v1_0_0.Protocol`.

CSDL sample:

```
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="ManagerNetworkProtocol.v1_0_0.Protocol">
    <ComplexType Name="Protocol">
      <Property Name="ProtocolEnabled" Type="Boolean"/>
      <Property Name="Port" Type="Edm.Int64"/>
    </ComplexType>

    <ComplexType Name="SSDProtocol" BaseType="ManagerNetworkProtocol.v1_0_0.Protocol">
      <Property Name="NotifyMulticastIntervalSeconds" Type="Edm.Int64"/>
      <Property Name="NotifyTTL" Type="Edm.Int64"/>
    </ComplexType>
  </Schema>
```

Using the above CSDL definitions, the JSON representation of `SSDProtocol` is shown below. Note that the JSON object contains the properties defined by both `SSDProtocol` and `Protocol`.

JSON representation:

```
{
    "ProtocolEnabled": true,
    "Port": 1900,
    "NotifyMulticastIntervalSeconds": 600,
    "NotifyTTL": 5
}
```

2.2. Redfish modeling practices

2.2.1. Core Redfish definitions

Redfish created two core schema files: Resource_v1.xml and RedfishExtensions_v1.xml. All other schema files published by Redfish leverage these files in some form.

The Resource_v1.xml schema file contains the base definitions for all Redfish resources, which includes:

- The base type definitions for all resources
  - "Resources" inherit from Resource.v1_0_0.Resource
  - "Resource Collections" inherit from Resource.v1_0_0.ResourceCollection
- Common properties found in all resources
  - Id: The unique identifier for a "Resource" in a given "Resource Collection"
  - Name: The string name for the "Resource" or "Resource Collection"
  - Description: The string description for the "Resource" or "Resource Collection"
  - Oem: An empty object that vendors are allowed to fill with custom properties
- Common structures and definitions leveraged by particular resources
  - Status: Contains health information for a given resource
  - Location: Contains information relating to how a user can find the physical equipment
  - Common enumerated lists such as IndicatorLED, PowerState, and ResetType

The RedfishExtensions_v1.xml scheme file contains Annotation elements to further enhance documentation and rules regarding payloads. See the Schema annotations used in Redfish in the Appendix for a list of terms defined by Redfish.

2.2.2. Defining Redfish resources

As a matter of convention, Redfish creates a single CSDL file per resource type, and the file is named after the resource. For example, the CSDL for the `ComputerSystem` resource can be found in the file ComputerSystem_v1.xml.

All resources are put into two categories: "Resources" or "Resource Collections".

A "Resource" represents a single resource, such as the `Thermal` EntityType defined in the Thermal_v1.xml schema file. All "Resources" inherit from Resource.v1_0_0.Resource. The `Id` property is defined as the key property in the EntityType definition.

"Resource Collections" represent a set of "Resources", such as the `ComputerSystemCollection` EntityType defined in the ComputerSystemCollection_v1.xml schema file. All "Resource Collections" inhert from Resource.v1_0_0.ResourceCollection. The `Name` property is defined as the key property in the EntityType definition. All "Resource Collections" contain a single NavigationProperty called `Members`, which is an array of references to the underlying "Resources" in the collection. For example, the

`ComputerSystemCollection` will have an array of references to `ComputerSystem` resources.

"Resources" typically contain a `Links` property. `Links` is a JSON object that contains different types of NavigationProperty elements to show how different resources in data model relate to one another. For example, in the ComputerSystem definition, there is a NavigationProperty called `ManagedBy` that is of type `Collection(Manager.Manager)`. This allows a ComputerSystem instance to reference a set of Managers in a different portion of the service in order to show which Managers are used to manage the given ComputerSystem. The CSDL for this is shown below.

Links CSDL sample:

```
  <NavigationProperty Name="ManagedBy" Type="Collection(Manager.Manager)">
    <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
    <Annotation Term="OData.Description" String="An array of references to the
Managers responsible for this system."/>
    <Annotation Term="OData.AutoExpandReferences"/>
  </NavigationProperty>
```

All "Resources" and "Resource Collections" have an optional `Oem` property. This property is an empty object that organizations are allowed to populate with their own data structure. In order to do this, the organization uses another object named after their organization within the `Oem` object; this is done in order to allow multiple organizations to make extensions on the same resource simultaneously without collisions. Inside the organization's object are all the properties being added to the resource. The example below shows the Contoso organization adding new properties to a ComputerSystem instance.

OEM example:

```
{
    "@odata.type": "#ComputerSystem.v1_5_0.ComputerSystem",
    "Id": "437XR1138R2",
    "Name": "WebFrontEnd483",
    "SystemType": "Physical",
    "Oem": {
        "Contoso": {
            "@odata.type": "#Contoso.v1_2_0.AnvilType1",
            "slogan": "Contoso anvils never fail",
            "disclaimer": "* Most of the time"
        }
    },
    ...
}
```

2.2.2.1. Resources in multiple Resource Collections

There are certain cases where a single "Resource" might belong in multiple "Resource Collections". The simple example is with the `Systems` and `StorageSystems` properties found on the Service Root. Both of these links go to resources that contain a collection of ComputerSystems. While these "Resource Collections" have their own unique URIs, the intent of the data model is that all instances of ComputerSystems found in the "Resource Collection" found via the `StorageSystems` property will also be found in the "Resource Collection" found via the `Systems` property. This type of practice is not common in generic OData implementations.

The two payloads below show samples of "Resource Collections" for the `Systems` and `StorageSystems` links respectively. Notice that the URIs in the `StorageSystems` payload are a subset of the URIs in the `Systems` payload.

```
{
    "@odata.id": "/redfish/v1/Systems",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Systems Collection",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/1"
        },
        {
            "@odata.id": "/redfish/v1/Systems/2"
        },
        {
            "@odata.id": "/redfish/v1/Systems/3"
        },
        {
            "@odata.id": "/redfish/v1/Systems/4"
        }
    ]
}
```

```
{
    "@odata.id": "/redfish/v1/StorageSystems",
    "@odata.type": "#StorageSystemCollection.StorageSystemCollection",
    "Name": "Storage Systems Collection",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/2"
        },
        {
            "@odata.id": "/redfish/v1/Systems/4"
```

```
        }
    ]
}
```

2.2.3. Referenceable Members

In some cases, Redfish uses EntityType elements to define embedded objects within a given Resource. These EntityType elements inherit from Resource.v1_0_0.ReferenceableMember, which is defined in the Resource_v1.xml schema file. All "Referenceable Members" contain a "MemberId" property. A client is not able to perform HTTP operations, such as GET, on these EntityType elements. AutoExpand is also included to ensure the properties of the resource are populated within the JSON body. The purpose of defining these embedded objects as EntityType elements as opposed to ComplexType elements is to leverage the `@odata.id` property in order to allow other portions of the Redfish data model to provide a URI to an individual structure, such as a RelatedItem link pointing to a single Temperature object. The `@odata.id` property is structured as a URI with a JSON fragment identifier in these cases.

In the CSDL sample below, the Resource `Thermal` is defined. It contains a single NavigationProperty element named `Temperatures`, which is an array of `Thermal.v1_0_0.Temperature` elements. This NavigationProperty also contains the `OData.AutoExpand` Annotation element, meaning that the properties defined by `Thermal.v1_0_0.Temperature` will be contained in the payload for `Thermal.v1_0_0.Thermal`.

CSDL sample:

```
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Thermal.v1_0_0">
    <EntityType Name="Thermal" BaseType="Thermal.Thermal">
      <NavigationProperty Name="Temperatures"
Type="Collection(Thermal.v1_0_0.Temperature)" ContainsTarget="true">
        <Annotation Term="OData.Permissions" EnumMember="OData.Permission/ReadWrite"/>
        <Annotation Term="OData.Description" String="This is the definition for
temperature sensors."/>
        <Annotation Term="OData.LongDescription" String="These properties shall be the
definition for temperature sensors for a Redfish implementation."/>
        <Annotation Term="OData.AutoExpand"/>
      </NavigationProperty>
    </EntityType>
    <EntityType Name="Temperature" BaseType="Resource.v1_0_0.ReferenceableMember">
      <Property Name="Name" Type="Edm.String">
        <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
        <Annotation Term="OData.Description" String="Temperature sensor name."/>
        <Annotation Term="OData.LongDescription" String="The value of this property
shall be the name of the temperature sensor."/>
      </Property>
```

```
    </EntityType>
  </Schema>
```

Using the above CSDL definitions, the JSON representation of `Thermal.v1_0_0.Temperature` is shown below. It contains a property named `Temperatures`, which contains an array of objects. In this case, there are two instances, each of which contain the properties `@odata.id`, `MemberId`, and `Name`. The `@odata.id` property in each of the Temperature objects contains the URI of the Thermal resource, and a JSON fragment identifier that identifies where in the JSON response the object resides.

JSON representation:

```
{
    "@odata.id": "/redfish/v1/Chassis/1/Thermal",
    "Temperatures": [
        {
            "@odata.id": "/redfish/v1/Chassis/1/Thermal#/Temperatures/0",
            "MemberId": "0",
            "Name": "CPU1 Temp"
        },
        {
            "@odata.id": "/redfish/v1/Chassis/1/Thermal#/Temperatures/1",
            "MemberId": "1",
            "Name": "Intake Temp"
        }
    ],
    ...
}
```

2.2.4. Schema versioning

As stated in the previous section, all resources are put into two categories: "Resources" or "Resource Collections".

"Resource Collections" do not contain any version information. This is because "Resource Collections" contain a single `Members` property, and the overall definition never grows over time. The Namespace used in these definitions is always the same as the EntityType name. For example, the ChassisCollection_v1.xml schema file contains a single Namespace called `ChassisCollection`, and within that namespace is a single EntityType definition also called `ChassisCollection`.

"Resources" contain version information encoded in the name of the Namespaces used in the schema files. The first Namespace for a "Resource" is unversioned, and is the same name of the "Resource" itself. This Namespace also contains a single EntityType definition for the "Resource", and is defined to

be abstract. Subsequent Namespaces contain version information, and the definitions within each Namespace inherits from the previous versions. Versioned Namespaces are in the format of `ResourceName.vX_Y_Z`, where `X` is the major version, `Y` is the minor version, and `Z` is the errata version.

When new functionality is added, such as adding a new Property, a new minor version of the "Resource" is created. When an existing definition is corrected, such as fixing an Annotation term on a Property, a new errata version is created. Major versions are reserved for definitions that break backward compatibility with existing definitions. For a complete definition of versioning, see the Redfish Specification.

The CSDL below contains a collapsed definition of the Session resource to highlight the versioning.

- The first Namespace is called `Session`, and contains a single EntityType definition also called `Session`.
- The second Namespace is called `Session.v1_0_0`, which is the 1.0.0 definition, and the `Session` EntityType inherits from `Session.Session`.
- The next two Namespaces are `Session.v1_0_2` and `Session.v1_0_3`, which are versions 1.0.2 and 1.0.3 respectively. Their `Session` EntityType definitions inherit from the previous versions. These were created to fix Annotations found in the 1.0.0 definition.
- The last Namespace is `Session.v1_1_0`, which is the 1.1.0 definition. The `Session` EntityType added a new `Actions` property to the existing `Session` definition.

Session CSDL versioning:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Session">
  <EntityType Name="Session" BaseType="Resource.v1_0_0.Resource" Abstract="true"/>
</Schema>

<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Session.v1_0_0">
  <EntityType Name="Session" BaseType="Session.Session"/>
</Schema>

<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Session.v1_0_2">
  <EntityType Name="Session" BaseType="Session.v1_0_0.Session"/>
</Schema>

<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Session.v1_0_3">
  <EntityType Name="Session" BaseType="Session.v1_0_2.Session"/>
</Schema>

<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Session.v1_1_0">
  <EntityType Name="Session" BaseType="Session.v1_0_3.Session">
    <Property Name="Actions" Type="Session.v1_1_0.Actions" Nullable="false"/>
```

```
    </EntityType>
  </Schema>
```

2.3. CSDL vs. JSON Schema

The DMTF publishes all Redfish schema files in two formats: CSDL and JSON Schema. Both formats are functionally equivalent, and it's up to the client's design whether it uses one form versus the other. Currently, the DMTF uses a tool to automatically generate all of the JSON Schema files based off the CSDL definitions. Other than the language of the schema files themselves, the distinct difference between the two formats is CSDL has one file per resource type, whereas JSON Schema uses one file per version per resource type. For example, the Session CSDL file shown in the Schema versioning section will generate five JSON Schema files: Session.json, Session.v1_0_0.json, Session.v1_0_2.json, Session.v1_0_3.json, and Session.v1_1_0.json.

For those interested in CSDL to JSON Schema conversion process, refer to the Redfish Tools repository link in the References section. A tool to convert from JSON Schema to CSDL has yet to be released.

3. Payload annotations

Payload annotations are a mechanism in which a service can provide additional information about a given property or object within a response. The definitions for these annotations are the same as annotations used in the CSDL files; an Annotation element in a given Namespace can be used to define payload annotations.

Redfish limits the scope of these to only core terms defined by OData, as well as Annotation elements defined in the `Redfish` and `Message` Namespaces. The `Redfish` Namespace is an alias for the `RedfishExtensions.v1_0_0` Namespace found in RedfishExtensions_v1.xml. The `Message` Namespace is found in Message_v1.xml.

3.1. Annotating a single property in a response

A payload annotation for a single property takes the form of `Property@Namespace.Term`, where `Property` is the JSON property being annotated, `Namespace` is the Namespace in the CSDL file where the definition is found, and `Term` is the name of the Annotation element found in the Namespace.

In the example below, the property `ResetType` is being annotated with the `AllowableValues` term, which is defined in the `Redfish` Namespace. This is used to indicate to the client that the service supports the values `On` and `ForceOff` for `ResetType`.

Property annotation example:

```
{
    "ResetType@Redfish.AllowableValues": [
        "On",
        "ForceOff"
    ],
    ...
}
```

Common property annotations in payloads:

Term	Usage
`@Redfish.AllowableValues`	Indicates to the client the different string values the service accepts for a given action parameter
`@Message.ExtendedInfo`	Allows the service to provide a set of Message structures for a given property to indicate additional information; this can be useful when a property is `null` due to an error condition, and the service wants to convey why the property is `null`
`@odata.count`	Can be used on properties that are arrays in order to indicate their size so that a client does not need to count the array members

3.2. Annotating an object in a response

A payload annotation for an object takes the form of `@Namespace.Term`, where `Namespace` is the Namespace in the CSDL file where the definition is found and `Term` is the name of the Annotation element found in the Namespace. These payload annotations are used to provide further information about the object itself.

In the example below, the object is being annotated with the `ActionInfo` term, which is defined in the `Redfish` Namespace. This is used to indicate to the client that it can find more information about the given action, in this case `#ComputerSystem.Reset`, at the URI `/redfish/v1/Systems/1/ResetActionInfo`.

Object annotation example:

```
{
    "#ComputerSystem.Reset": {
        "target": "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset",
```

```
        "@Redfish.ActionInfo": "/redfish/v1/Systems/1/ResetActionInfo"
    },
    ...
}
```

Common object annotations in payloads:

Term	Usage
`@Redfish.Settings`	Gives the client a reference to the resource that represents the future property settings to be applied to this object
`@Redfish.ActionInfo`	Used on actions to provide the client a reference to an ActionInfo resource, which gives detailed information about a given action's parameters
`@Message.ExtendedInfo`	Allows the service to provide a set of Message structures for a given object to indicate additional information; this can be useful when an error condition is reached, and the service wants to convey what error was encountered
`@odata.id`	Provides the unique URI for a given resource
`@odata.type`	Provides the type definition of the object in the format of `#Namespace.Type`, where `Namespace` is the Namespace in the CSDL file where the definition is found and `Type` is the name of the ComplexType or EntityType element found in the Namespace
`@odata.context`	Provides an OData client with a descriptor for the content of the payload; in Redfish, this is simply always going to be `/redfish/v1/$metadata#Namespace.Entity`, where `Namespace` is the unversioned Namespace in the CSDL file where the definition is found and `Entity` is the name of the EntityType element being used

4. OData service document

All services must provide the OData service document at the URI `/redfish/v1/odata`. This document is not a Redfish resource, but rather is the entry point to the service for OData clients. The OData service document is a JSON object and contains two properties: `value` and `@odata.context`. The `value` property consists of an array of the top level entry points to the service, and the `@odata.context` property contains the URI to the metadata document. A sample OData service document can be found in

the Appendix.

The CSDL definition for what is defined in the OData service document is found in the EntityContainer element. Within Redfish, this is defined in the ServiceRoot_v1.xml file, and is given the name `ServiceContainer`. As a general rule, any of the NavigationProperty elements defined in the ServiceRoot EntityType definition are also put into the EntityContainer definition. Redfish also only uses Singletons. Similar to how the BaseType term can be used to extend the definition of an existing ComplexType or EntityType element, the Extends term can be used in an EntityContainer element to add new definitions to an existing EntityContainer. Whenever new NavigationProperty elements are added to the ServiceRoot resource, the `ServiceContainer` definition is also expanded accordingly.

The OData service document for a given service must match the ServiceRoot resource for that same service. For example, if a service supports the AccountService resource, it must be in both the OData service document as well as the ServiceRoot resource.

Vendors can also create their own OEM services and tie them into the OData service document. Using the same methodology as when ServiceRoot is published with new standard services, a vendor can extend the standard `ServiceContainer` definition for their own purposes. In the example below, Contoso created their own extension to the `ServiceContainer` by basing it on the `ServiceContainer` definition found in the `ServiceRoot.v1_2_0` Namespace, and adding a new service called `TurboencabulatorService`, which is of type `TurboencabulatorService.TurboencabulatorService`.

Example of extending `ServiceContainer`:

```
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="ContosoExtensions.v1_0_0">

    <EntityContainer Name="ServiceContainer"
Extends="ServiceRoot.v1_2_0.ServiceContainer">
      <Singleton Name="TurboencabulatorService"
Type="TurboencabulatorService.TurboencabulatorService"/>
    </EntityContainer>

  </Schema>
```

5. Metadata document

All services must provide the metadata document at the URI `/redfish/v1/$metadata`. This document is used by OData clients to resolve definitions for payloads it finds in the service. The metadata document is an XML file, and in typical OData cases it contains the entire schema definition for the service. However, Redfish uses this document to simply point to the CSDL files that the DMTF has published, as

well as any OEM CSDL files a given service may require. This is done in order to maintain consistency with what has been published by the DMTF. In Redfish, this document is typically consists of a set of references to Namespaces the service references, as well as the EntityContainer definition for the service.

The metadata document does not need to include every Namespace ever defined by Redfish; it just needs to include the Namespaces referenced by the service. There are a few things to examine to help a developer construct the metadata document for their own service:

- Include the Namespaces referenced by the `@odata.type` properties returned by the service
  - Example: if the service can return the `@odata.type` property with the value `#ComputerSystem.v1_5_0.ComputerSystem`, it must include the Namespace `ComputerSystem.v1_5_0` in the metadata document
- Include the Namespaces referenced by the `@odata.context` properties returned by the service
  - Example: if the service can return the `@odata.context` property with the value `/redfish/v1/$metadata#ComputerSystem.ComputerSystem`, it must include the Namespace `ComputerSystem` in the metadata document
- Include the Namespaces referenced by payload annotations returned by the service
  - This is limited to the `Redfish` and `Message` Namespaces
  - The `Redfish` Namespace is an alias for the `RedfishExtensions.v1_0_0` Namespace; the alias must be defined in the metadata document
- Do not forget to include Namespaces referenced by error responses
  - Typically this is limited to the `Message` Namespace and one of the versioned Namespaces inside Message_v1.xml, such as `Message.v1_0_0.Message`
- Do not forget to include the Namespace where the `ServiceContainer` will be referenced

The EntityContainer definition for the service is placed between the `<edmx:DataServices>` tags. This definition will simply reference the `ServiceContainer` definition found in ServiceRoot_v1.xml, or the OEM definition of the `ServiceContainer` if needed.

A sample metadata document can be found in the Appendix.

6. Appendix

6.1. Primitive OData types used in Redfish

Type	JSON Representation
`Edm.Boolean`	Boolean

Type	JSON Representation
`Edm.DateTimeOffset`	String, formatted as a date-time value with offset
`Edm.Decimal`	Number, optionally containing a decimal point
`Edm.Double`	Number, optionally containing a decimal point and optionally containing an exponent
`Edm.Guid`	String, matching the pattern ([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})
`Edm.Int64`	Number with no decimal point
`Edm.String`	String

6.2. Schema annotations used in Redfish

6.2.1. Core annotations defined by OData

Term	Usage
`OData.Description`	Provides a human-readable string to describe what a Property, NavigationProperty, ComplexType, or other definition is.
`OData.LongDescription`	Provides a string containing normative language about a Property, NavigationProperty, ComplexType, or other definition.
`OData.Permissions`	Dictates whether a given Property is writable or read only.
`OData.AdditionalProperties`	Shows whether a given ComplexType or EntityType is allowed to have more properties than what is defined in the schema.
`OData.AutoExpandReferences`	Shows whether a given NavigationProperty contains its reference (`@odata.id` property).
`OData.AutoExpand`	Shows whether the service will expand the properties found in the NavigationProperty in the current payload.
`Measures.Unit`	Documents the units of measurement for a value; the

Term	Usage
	Unified Code for Units of Measure (UCUM) notation used.
`Capabilities.InsertRestrictions`	Shows whether a client is allowed to add new members for a given "Resource Collection".
`Capabilities.UpdateRestrictions`	Shows whether a client is allowed to modify the resource.
`Capabilities.DeleteRestrictions`	Shows whether a client is allowed to delete a given resource.

6.2.2. Redfish annotations defined in RedfishExtensions_v1.xml

Term	Usage
`Redfish.Required`	Indicates whether a Property or NavigationProperty is required to be implemented by the service.
`Redfish.RequiredOnCreate`	Indicates whether a Property or NavigationProperty is required to be provided by the client as part of a create request.
`Redfish.IPv6Format`	Indicates whether a Property follows IPv6 addressing or formatting rules.
`Redfish.Deprecated`	Indicates whether a Property, NavigationProperty, or other definition should no longer be used; also provides guidance on what should be done instead.
`Redfish.DynamicPropertyPatterns`	Indicates whether a service can add additional properties that conform to the patterns specified by the schema file.
`Validation.Pattern`	Gives a regex string to show proper formatting for a string property.
`Validation.Minimum`	Gives a minimum value a numeric property is allowed to return.
`Validation.Maximum`	Gives a maximum value a numeric property is allowed to return.

6.3. Sample OData service document


```
{
    "@odata.context": "/redfish/v1/$metadata",
    "value": [
        {
            "name": "Service",
            "kind": "Singleton",
            "url": "/redfish/v1/"
        },
        {
            "name": "Systems",
            "kind": "Singleton",
            "url": "/redfish/v1/Systems"
        },
        {
            "name": "Chassis",
            "kind": "Singleton",
            "url": "/redfish/v1/Chassis"
        },
        {
            "name": "Managers",
            "kind": "Singleton",
            "url": "/redfish/v1/Managers"
        },
        {
            "name": "TaskService",
            "kind": "Singleton",
            "url": "/redfish/v1/TaskService"
        },
        {
            "name": "AccountService",
            "kind": "Singleton",
            "url": "/redfish/v1/AccountService"
        },
        {
            "name": "SessionService",
            "kind": "Singleton",
            "url": "/redfish/v1/SessionService"
        },
        {
            "name": "EventService",
            "kind": "Singleton",
            "url": "/redfish/v1/EventService"
        },
        {
```

```
            "name": "Sessions",
            "kind": "Singleton",
            "url": "/redfish/v1/SessionService/Sessions"
        }
    ]
}
```

6.4. Sample metadata document

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">

  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ServiceRoot_v1.xml">
    <edmx:Include Namespace="ServiceRoot"/>
    <edmx:Include Namespace="ServiceRoot.v1_2_0"/>
  </edmx:Reference>

  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/AccountService_v1.xml">
    <edmx:Include Namespace="AccountService"/>
    <edmx:Include Namespace="AccountService.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Bios_v1.xml">
    <edmx:Include Namespace="Bios"/>
    <edmx:Include Namespace="Bios.v1_0_2"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Chassis_v1.xml">
    <edmx:Include Namespace="Chassis"/>
    <edmx:Include Namespace="Chassis.v1_5_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ChassisCollection_v1.xml">
    <edmx:Include Namespace="ChassisCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ComputerSystem_v1.xml">
    <edmx:Include Namespace="ComputerSystem"/>
    <edmx:Include Namespace="ComputerSystem.v1_4_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
ComputerSystemCollection_v1.xml">
    <edmx:Include Namespace="ComputerSystemCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/EthernetInterface_v1.xml">
    <edmx:Include Namespace="EthernetInterface"/>
    <edmx:Include Namespace="EthernetInterface.v1_3_0"/>
  </edmx:Reference>
```

```
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
EthernetInterfaceCollection_v1.xml">
    <edmx:Include Namespace="EthernetInterfaceCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/EventDestination_v1.xml">
    <edmx:Include Namespace="EventDestination"/>
    <edmx:Include Namespace="EventDestination.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
EventDestinationCollection_v1.xml">
    <edmx:Include Namespace="EventDestinationCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/EventService_v1.xml">
    <edmx:Include Namespace="EventService"/>
    <edmx:Include Namespace="EventService.v1_0_4"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/LogEntry_v1.xml">
    <edmx:Include Namespace="LogEntry"/>
    <edmx:Include Namespace="LogEntry.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/LogEntryCollection_v1.xml">
    <edmx:Include Namespace="LogEntryCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/LogService_v1.xml">
    <edmx:Include Namespace="LogService"/>
    <edmx:Include Namespace="LogService.v1_0_4"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/LogServiceCollection_v1.xml">
    <edmx:Include Namespace="LogServiceCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Manager_v1.xml">
    <edmx:Include Namespace="Manager"/>
    <edmx:Include Namespace="Manager.v1_3_1"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ManagerCollection_v1.xml">
    <edmx:Include Namespace="ManagerCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ManagerAccount_v1.xml">
    <edmx:Include Namespace="ManagerAccount"/>
    <edmx:Include Namespace="ManagerAccount.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
ManagerAccountCollection_v1.xml">
    <edmx:Include Namespace="ManagerAccountCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
ManagerNetworkProtocol_v1.xml">
```

```
    <edmx:Include Namespace="ManagerNetworkProtocol"/>
    <edmx:Include Namespace="ManagerNetworkProtocol.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Memory_v1.xml">
    <edmx:Include Namespace="Memory"/>
    <edmx:Include Namespace="Memory.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/MemoryCollection_v1.xml">
    <edmx:Include Namespace="MemoryCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Message_v1.xml">
    <edmx:Include Namespace="Message"/>
    <edmx:Include Namespace="Message.v1_0_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Power_v1.xml">
    <edmx:Include Namespace="Power"/>
    <edmx:Include Namespace="Power.v1_3_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Processor_v1.xml">
    <edmx:Include Namespace="Processor"/>
    <edmx:Include Namespace="Processor.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ProcessorCollection_v1.xml">
    <edmx:Include Namespace="ProcessorCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Role_v1.xml">
    <edmx:Include Namespace="Role"/>
    <edmx:Include Namespace="Role.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/RoleCollection_v1.xml">
    <edmx:Include Namespace="RoleCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/SerialInterface_v1.xml">
    <edmx:Include Namespace="SerialInterface"/>
    <edmx:Include Namespace="SerialInterface.v1_1_1"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
SerialInterfaceCollection_v1.xml">
    <edmx:Include Namespace="SerialInterfaceCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Session_v1.xml">
    <edmx:Include Namespace="Session"/>
    <edmx:Include Namespace="Session.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/SessionCollection_v1.xml">
    <edmx:Include Namespace="SessionCollection"/>
  </edmx:Reference>
```

```
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/SessionService_v1.xml">
    <edmx:Include Namespace="SessionService"/>
    <edmx:Include Namespace="SessionService.v1_1_2"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Settings_v1.xml">
    <edmx:Include Namespace="Settings.v1_0_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/SimpleStorage_v1.xml">
    <edmx:Include Namespace="SimpleStorage"/>
    <edmx:Include Namespace="SimpleStorage.v1_2_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
SimpleStorageCollection_v1.xml">
    <edmx:Include Namespace="SimpleStorageCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Task_v1.xml">
    <edmx:Include Namespace="Task"/>
    <edmx:Include Namespace="Task.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/TaskCollection_v1.xml">
    <edmx:Include Namespace="TaskCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/TaskService_v1.xml">
    <edmx:Include Namespace="TaskService"/>
    <edmx:Include Namespace="TaskService.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Thermal_v1.xml">
    <edmx:Include Namespace="Thermal"/>
    <edmx:Include Namespace="Thermal.v1_3_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/VirtualMedia_v1.xml">
    <edmx:Include Namespace="VirtualMedia"/>
    <edmx:Include Namespace="VirtualMedia.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
VirtualMediaCollection_v1.xml">
    <edmx:Include Namespace="VirtualMediaCollection"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/VLanNetworkInterface_v1.xml">
    <edmx:Include Namespace="VLanNetworkInterface"/>
    <edmx:Include Namespace="VLanNetworkInterface.v1_1_0"/>
  </edmx:Reference>
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/
VLanNetworkInterfaceCollection_v1.xml">
    <edmx:Include Namespace="VLanNetworkInterfaceCollection"/>
  </edmx:Reference>
```

```
  <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/RedfishExtensions_v1.xml">
    <edmx:Include Namespace="RedfishExtensions.v1_0_0" Alias="Redfish"/>
  </edmx:Reference>

  <edmx:DataServices>

    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Service">
      <EntityContainer Name="Service" Extends="ServiceRoot.v1_2_0.ServiceContainer"/>
    </Schema>

  </edmx:DataServices>

</edmx:Edmx>
```

6.5. References

- OData Documentation: [http://www.odata.org/documentation/]
- Redfish Tools Repo: [https://github.com/DMTF/Redfish-Tools]