Document Identifier: DSP2050

Date: 2023-10-04

Version: 1.3.0

Redfish Composability White Paper

Supersedes: 1.2.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

Foreword

The Redfish Composability White Paper was prepared by DMTF's Redfish Forum.

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

Acknowledgments

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

- Rafiq Ahamed K — Hewlett Packard Enterprise
- Jeff Autor — Hewlett Packard Enterprise
- Michael Du — Lenovo
- Jeff Hilland — Hewlett Packard Enterprise
- John Leung — Intel Corporation
- Steve Lyle — Hewlett Packard Enterprise
- Michael Raineri — Dell Technologies
- Paul von Behren — Intel Corporation

1 Introduction

As the world is transitioning to a software defined paradigm, there is a need for hardware management capabilities to evolve to address that shift in the data center. In the context of disaggregated hardware, management software needs the ability to conjoin the independent pieces of hardware, such as trays, modules, silicon, etc., together to create a composed logical system. These logical systems function just like traditional industry standard rack-mount systems. This allows users to dynamically configure their hardware to meet the needs of their workloads. In addition, users are able to manage the life cycle of their systems, such as adding more compute to their logical system, without having to physically move any equipment.

Redfish is an evolving hardware management standard that is designed to be flexible, extensible, and interoperable. Redfish contains a data model that is used to describe composable hardware, as well as an interface for clients to manage their compositions. This document helps implementers and clients understand the Redfish Composability data model as well as how composition requests are expected to be formed.

2 Modeling for composability

If a service supports composability, the service root resource will contain the `CompositionService` property. Within the Composition Service, a client will find the inventory of all components that can be composed into new things (Resource Blocks), descriptors containing the binding restrictions of the different components (Resource Zones), and annotations informing the client as to how to form composition requests (collection capabilities). The following sections detail how these things are reported by a service.

2.1 Composition service

The composition service, defined by the `CompositionService` schema, is the top-level resource for all things related to composability. It contains status and control indicator properties such as `Status` and `ServiceEnabled`. These are common properties found on various Redfish service instances.

The composition service contains the `AllowOverprovisioning` property. This is used to indicate if the service can accept constrained composition requests where the client may allow for more resources than those requested.

The composition service contains the `AllowZoneAffinity` property. This is used to indicate if the service can accept constrained composition requests where the client may desire a given composition request to be fulfilled using resource blocks from a particular resource zones.

The composition service contains links to its collections of resource blocks and resource zones through the properties `ResourceBlocks` and `ResourceZones` respectively. It may also report resource block utilization with the collections referenced by the `ActivePool` and `FreePool` properties. These collections allow a user to determine which resource blocks are participating in compositions and which are not utilized without the need to perform collection filtering.

The composition service can also contain the `#CompositionService.Compose` action to perform manifest-driven composition requests for bulk operations. This also allows a user to reserve the requested configuration of the operation, for up to the duration specified by the `ReservationDuration` property, without applying it yet. These composition reservations are tracked in the collection referenced by the `CompositionReservations` property.

Example `CompositionService` resource:


```
{
    "@odata.id": "/redfish/v1/CompositionService",
    "@odata.type": "#CompositionService.v1_2_1.CompositionService",
    "Id": "CompositionService",
    "Name": "Composition Service",
    "Status": {
        "State": "Enabled",
```

```
        "Health": "OK"
    },
    "ServiceEnabled": true,
    "AllowOverprovisioning": true,
    "AllowZoneAffinity": true,
    "ResourceBlocks": {
        "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks"
    },
    "ResourceZones": {
        "@odata.id": "/redfish/v1/CompositionService/ResourceZones"
    },
    "ActivePool": {
        "@odata.id": "/redfish/v1/CompositionService/ActivePool"
    },
    "FreePool": {
        "@odata.id": "/redfish/v1/CompositionService/FreePool"
    },
    "CompositionReservations": {
        "@odata.id": "/redfish/v1/CompositionService/CompositionReservations"
    },
    "ReservationDuration": "PT3H",
    "Actions": {
        "#CompositionService.Compose": {
            "target": "/redfish/v1/CompostionService/Actions/CompositionService.Compose",
            "@Redfish.ActionInfo": "/redfish/v1/CompositionService/ComposeActionInfo"
        }
    }
}
```


2.2 Resource blocks

Resource blocks, defined by the `ResourceBlock` schema, are the lowest level building blocks for composition requests. Resource blocks contain status and control information about itself. They also contain the list of components found within the resource block. For example, if a resource block contains 1 processor and 4 DIMMs, then all of those components will be part of the same composition request, even if only one of them is needed. In a completely disaggregated system, a client would likely find one component within each resource block. Resource blocks, and their components, are not in a state where system software is able to use them until they belong in a composition. For example, if a resource block contains a drive, the drive will not belong to any given computer system until a composition request is made that makes use of its resource block.

The property `ResourceBlockType` contains classification information about the types of components found on the resource block that can be used to help clients quickly identify a resource block. Each `ResourceBlockType` is associated with specific schema elements, which will be contained within that resource block. For example, if the value `Storage` was found in this property, then a client would know that this particular resource block contains storage related devices, such as storage controllers or drives, without having to inspect the individual component resources. The value `Compute` has special meaning: this is used to describe resource blocks that have bound

processor and memory components that operate together as a compute subsystem. The value `Expansion` is also a special indicator that shows a particular resource block may have different types of devices over time, such as when a resource block contains plug-in cards where a user may replace the components at any time. `ResourceBlockType` is an array, meaning that it's possible for a resource block to have multiple types associated with it. For example, if the resource block in question is a CPU-memory complex, but not a full computer system, and also has an integrated network controller, `ResourceBlockType` could contain both `Compute` and `Network`.

The property `CompositionStatus` is an object that contains several properties:

- `CompositionState` is used to inform the client of the state of this resource block regarding its use in a composition.
- `Reserved` is a writable flag that clients can use to help convey that this resource block has been identified by a client, and that the client will be using it for a composition. If a second client that is attempting to identify resources for a composition sees the `Reserved` flag set to true, the second client should consider it allocated and not use it; the second client should move on to the next resource block for further processing. The service does not provide any sort of protection with the `Reserved` flag; any client can change its state and it's up to clients to behave fairly.
- `SharingCapable` is a flag to indicate if the resource block is capable of participating in multiple compositions simultaneously.
- `SharingEnabled` is a writable flag to indicate if the resource block is allowed to participate in multiple compositions simultaneously.
- `MaxCompositions` is used to indicate the maximum number of compositions in which the resource block is capable of participating simultaneously.
- `NumberOfCompositions` is used to indicate the number of compositions in which the resource block is currently participating.

There are several arrays of links to various component types, such as the `Processors`, `Memory`, and `Storage` arrays. These links reference resources that represent the individual components that are within the resource block. These components are made available to the new composition after the composition request completes. The `ComputerSystems` array is used when a resource block contains one or more whole computer systems. This gives the client the ability to create a single composed computer system from a set of smaller computer systems.

The `Links` property contains references to related resources. The `Chassis` array contains the chassis instances that contain the resources within the resource block. The `ComputerSystems` array contains the computer systems that are consuming the resource block as part of a composition. The `Zones` array contains links to the resource zones that contain the resource block.

Example `ResourceBlock` resource:


```
{
    "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3",
    "@odata.type": "#ResourceBlock.v1_4_1.ResourceBlock",
    "Id": "DriveBlock3",
```

```
    "Name": "Drive Block 3",
    "ResourceBlockType": [
        "Storage"
    ],
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "CompositionStatus": {
        "Reserved": false,
        "CompositionState": "ComposedAndAvailable",
        "SharingCapable": true,
        "SharingEnabled": true,
        "MaxCompositions": 8,
        "NumberOfCompositions": 2
    },
    "Drives": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3/Drives/Block3Slot0"
        }
    ],
    "Links": {
        "ComputerSystems": [
            {
                "@odata.id": "/redfish/v1/Systems/ComposedSystem"
            },
            {
                "@odata.id": "/redfish/v1/Systems/ComposedSystem2"
            }
        ],
        "Chassis": [
            {
                "@odata.id": "/redfish/v1/Chassis/ComposableModule3"
            }
        ],
        "Zones": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceZones/1"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceZones/2"
            }
        ]
    }
}
```


In the previous example, the resource block is of type `Storage` and it contains a single drive. From the `CompositionStatus` property, the resource block is currently participating in two compositions and can be used in

more compositions. In the `Links` property, it's consumed by the computer systems `ComposedSystem` and `ComposedSystem2`.

2.2.1 Recommended state diagrams for CompositionState

As clients make requests to create or delete composed resources, a resource block will transition between different states and is reflected by the `CompositionState` property within the `CompositionStatus` object. Figure 1 shows the recommended state diagram for `CompositionState` involving a resource block that is not shareable. Figure 2 shows the recommended state diagram for `CompositionState` involving a resource block that is shareable. While not shown in the diagrams, client requests can fail for precondition checks, such as a device not powered, thus leaving the state unchanged.

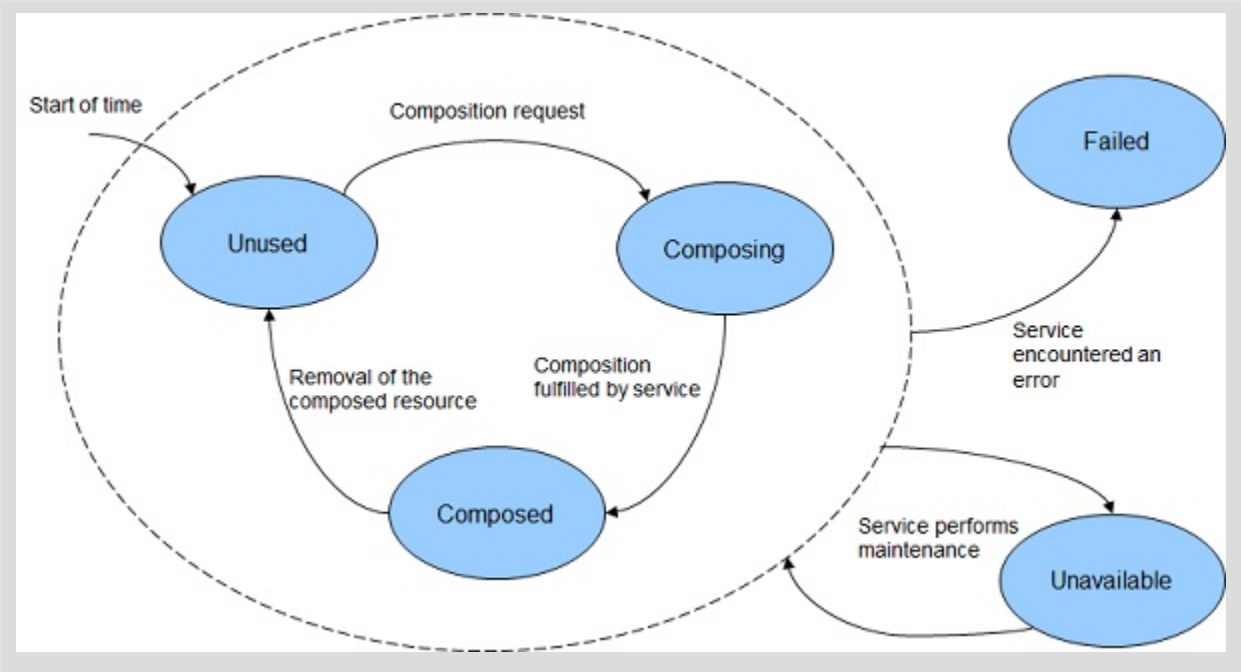

Figure 1

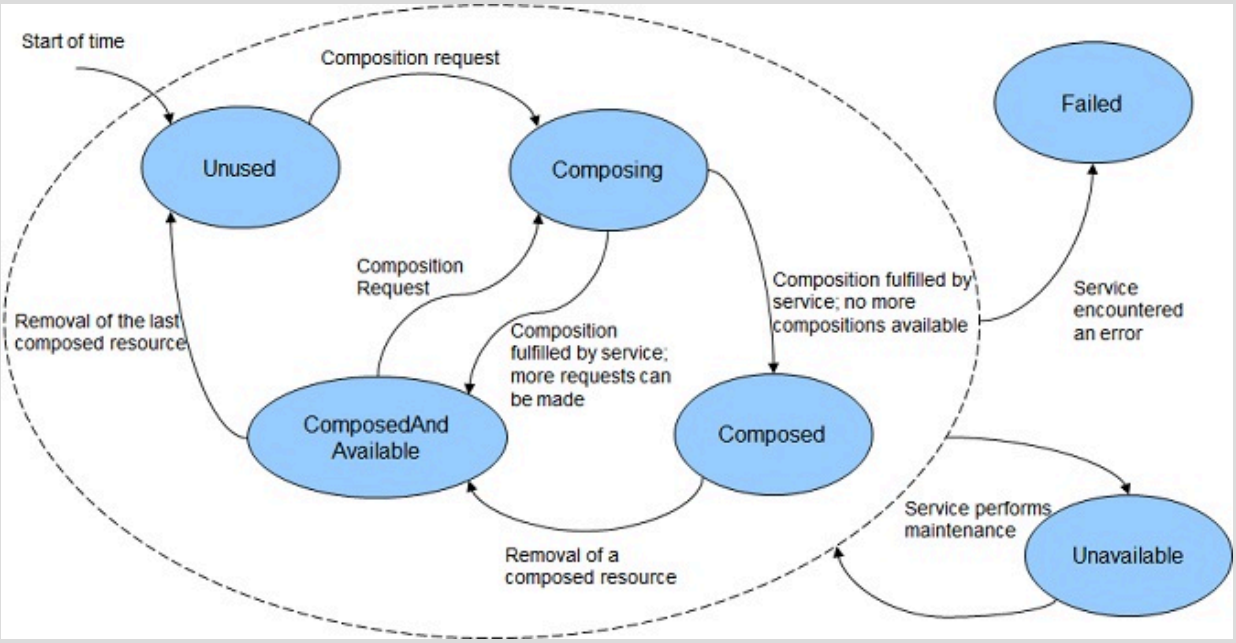

Figure 2

2.3 Resource zones

Resource zones, defined by the `Zone` schema, describe to the client the different composition restrictions of the resource blocks reported by the service. Resource blocks that are reported in the same resource zone are allowed to participate in the same compositions. It's possible for a resource block to belong to multiple resource zones. This allows the clients to determine restrictions with combinations of resource blocks before attempting to perform a composition request.

Example `Zone` resource:

```
{
    "@odata.type": "#Zone.v1_6_0.Zone",
    "@odata.id": "/redfish/v1/CompositionService/ResourceZones/1",
    "Id": "1",
    "Name": "Resource Zone 1",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "ZoneType": "ZoneOfResourceBlocks",
    "Links": {
```

```
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock1"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock4"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock5"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock6"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock7"
            }
        ]
    }
}
```

In the previous example, the resource blocks `ComputeBlock1`, `DriveBlock3`, `DriveBlock4`, `DriveBlock5`, `DriveBlock6`, and `DriveBlock7` are all in the same resource zone. This signifies that these six resource blocks are allowed to participate in the same compositions.

2.4 Collection capabilities

Collection capabilities can be found resource collections that allow a user to add new members to the collection. Collection capabilities are identified by the `@Redfish.CollectionCapabilities` annotation in the response body of the resource collection. This annotation is used to inform the client how to form the request body for an HTTP `POST` operation to a given collection based on a specified use case for creating a new resource for the collection.

For composability, the collection capabilities annotation can be found in the following resource collections:

- In the `ComputerSystemCollection` resource to create a composed system.
- In the `ResourceBlockCollection` resource to create a composed resource block.

2.4.1 Collection capabilities annotation

Within the collection capabilities annotation, there is a property called `Capabilities`. This is an array to identify the different methods a client can use to form the request body for a create (`POST`) operation for the resource collection. Each member of this property contains an object to describe a particular capability.

The property `CapabilitiesObject` contains a URI to the underlying object instance that describes the payload format. This is described further in the next section.

The property `UseCase` is used to inform the client of the context of a particular create (`POST`) operation. The table below shows the different values for `UseCase` as used by Redfish composability.

`UseCase` value	Composed resource	Type of composition
`ComputerSystemComposition`	`ComputerSystem`	Specific composition.
`ComputerSystemConstrainedComposition`	`ComputerSystem`	Constrained composition.
`ResourceBlockComposition`	`ResourceBlock`	Specific composition.
`ResourceBlockConstrainedComposition`	`ResourceBlock`	Constrained composition.

Example collection capabilities annotation:

```
{
    "@Redfish.CollectionCapabilities": {
        "@odata.type": "#CollectionCapabilities.v1_1_0.CollectionCapabilities",
        "Capabilities": [
            {
                "CapabilitiesObject": {
                    "@odata.id": "/redfish/v1/Systems/Capabilities"
                },
                "UseCase": "ComputerSystemComposition"
            },
            {
                "CapabilitiesObject": {
                    "@odata.id": "/redfish/v1/Systems/ConstrainedCompositionCapabilities"
                },
                "UseCase": "ComputerSystemConstrainedComposition"
            }
        ]
    },
    ...
}
```

The previous annotation contains two capabilities. In the first capability object, the `UseCase` property shows that this capability describes how to form a create (`POST`) request to create a new `ComputerSystem` resource from a set of specific resource blocks. In the second capability object, the `UseCase` property shows that this capability describes how to form a create (`POST`) request to create a new `ComputerSystem` from a set of constraints.

2.4.2 Collection capabilities object

The collection capabilities object follows the schema of the new resource a client is able to create. For example, if the object is describing how to form a request to create a new `ComputerSystem` resource, then the object's type will be `ComputerSystem.vX_Y_Z.ComputerSystem`, where `vX_Y_Z` is the version of `ComputerSystem` resource supported by the service.

The object itself contains annotated properties the client can use in the body of the create (`POST`) operation. It also lists out optional properties, and any restrictions properties may have after the new resource is created. The table below describes the different annotations used on the properties within the collection capabilities object.

Property annotation	Description
`@Redfish.RequiredOnCreate`	The client must provide the given property in the body of the create (`POST`) request.
`@Redfish.OptionalOnCreate`	The client may provide the property in the body of the create (`POST`) request.
`@Redfish.SetOnlyOnCreate`	If the client has a specific value needed for the property, it must be provided in the body of the create (`POST`) request. This property is likely a read-only property after the resource is created.
`@Redfish.UpdatableAfterCreate`	The client is allowed to update the property after the resource is created.
`@Redfish.AllowableValues`	The client is allowed to use any of the specified values in the body of the create (`POST`) request for the given property.

In the above table, some of the annotation terms can conflict with one another if used incorrectly. This may be due to conflicting logical semantics with the term definitions. Services need to ensure their collection capabilities objects do not have the following types of conflicts:

- Do not annotate a property with both `@Redfish.RequiredOnCreate` and `@Redfish.OptionalOnCreate`. A property cannot be both required and optional.
- Do not annotate a property with both `@Redfish.SetOnlyOnCreate` and `@Redfish.UpdatableAfterCreate`. A property can only be one of these.

The object can also contain object level annotations to describe other types of payload rules to the client. The table below describes the different annotations used at the object level within the collection capabilities object.

Object annotation	Description
`@Redfish.RequestedCountRequired`	Indicates that the client is required to annotate the corresponding object in the request payload with `@Redfish.RequestedCount` to show how many instances of the object the client is requesting.
`@Redfish.ResourceBlockLimits`	Indicates any restrictions regarding quantities of resource blocks of a given type in a given composition request.

Example collection capabilities object:

```
{
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "@odata.id": "/redfish/v1/Systems/Capabilities",
    "Id": "Capabilities",
    "Name": "Capabilities for the system collection",
    "Name@Redfish.RequiredOnCreate": true,
    "Name@Redfish.SetOnlyOnCreate": true,
    "Description@Redfish.OptionalOnCreate": true,
    "Description@Redfish.SetOnlyOnCreate": true,
    "HostName@Redfish.OptionalOnCreate": true,
    "HostName@Redfish.UpdatableAfterCreate": true,
    "Boot@Redfish.OptionalOnCreate": true,
    "Boot": {
        "BootSourceOverrideEnabled@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideEnabled@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideTarget@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "Usb",
            "Hdd"
        ]
    },
    "Links@Redfish.RequiredOnCreate": true,
    "Links": {
        "ResourceBlocks@Redfish.RequiredOnCreate": true,
        "ResourceBlocks@Redfish.UpdatableAfterCreate": true
    },
    "@Redfish.ResourceBlockLimits": {
        "MinCompute": 1,
        "MaxCompute": 1,
        "MaxStorage": 8
    }
}
```

In the previous example, three properties are marked with the `@Redfish.RequiredOnCreate` annotation: `Name`, `Links`, and `ResourceBlocks` inside of `Links`. All other properties are annotated with `@Redfish.OptionalOnCreate`. However, both `Name` and `Description` are annotated with `@Redfish.SetOnlyOnCreate`, meaning they cannot be modified after the new resource is created. The response also tells us in the `@Redfish.ResourceBlockLimits` annotation that all requests must have exactly one resource block of type `Compute`, and can up have to eight resource blocks of type `Storage`.

3 Types of compositions

The Redfish composability data model provides flexibility for service implementers to support different composition types based on their needs. The service informs the client of the type of composition request based on the `UseCase` property found in the collection capabilities annotation. The following sections describe the different types of compositions. The existing Redfish composability model defines three types: specific composition, constrained composition, and expandable resources

3.1 Specific composition

Specific composition requests allow clients to create and manage the life cycle of composed resources through predefined resource blocks and resource zones. Since resource blocks are self contained entities within a resource zone, clients are able to pick and choose specific resource blocks for their composition request. An example end-to-end workflow for locating resource blocks and performing the specific composition request can be found in the Specific composition workflow section.

Another industry standard server design that fits into the example of specific composition is defined in the "Bladed Partitions" mockup. In this example, a multi-blade enclosure consisting of a disaggregated hardware chassis can be bound together to create what are called partitioned servers. These partitions can be composed using the specific composition. The Redfish service implements each blade within the enclosure as a resource block with `ResourceBlockType` set to either `Compute` or `Storage`, and allows the clients to combine multiple resource blocks to create a composed computer system, which is a partitioned server.

Example create (`POST`) request body for a specific composition:

```
{
    "Name": "Sample Composed System",
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/BladeComputeBlock1"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/BladeComputeBlock5"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/BladeStorageBlock8"
            }
        ]
    }
}
```

3.2 Constrained composition

Constrained composition requests allow clients to request a composition by specifying the number and characteristics of the components to assemble into a composition. The selection of the resource blocks is delegated by client to the composition service. An example of this type of composition can be found in the Constrained composition workflow section.

3.3 Expandable resources

In some cases, clients may not be able to directly compose new resources. Instead, the service may have a baseline resource, and the client is only able to add additional components, or remove them. For `ComputerSystem` resources, a client can identify this case with the `UseCases` property inside the `Composition` property. If `UseCases` contains `ExpandableSystem`, then the system is an expandable resource. The resource will also show a `ResourceBlocks` array in its `Links` property, which allows a user to inspect the resource block representation of the resource, which contains composition information about the resource.

Example expandable `ComputerSystem` resource:


```
{
    "@odata.id": "/redfish/v1/Systems/Expandable"
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "Id": "Expandable",
    "Name": "Sample Expandable System",
    "SystemType": "Physical",
    "UUID": "37596f04-b032-48ce-9bad-7d55a61fe574",
    "PowerState": "On",
    "Boot": {
        "BootSourceOverrideEnabled": "Once",
        "BootSourceOverrideTarget": "Pxe",
        "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "Usb",
            "Hdd"
        ]
    },
    "Composition": {
        "UseCases": [
            "ExpandableSystem"
        ]
    },
    "Processors": {
        "@odata.id": "/redfish/v1/Systems/ComposedSystem/Processors"
    },
```

```
    "Memory": {
        "@odata.id": "/redfish/v1/Systems/ComposedSystem/Memory"
    },
    "EthernetInterfaces": {
        "@odata.id": "/redfish/v1/Systems/ComposedSystem/EthernetInterfaces"
    },
    "Storage": {
        "@odata.id": "/redfish/v1/Systems/ComposedSystem/Storage"
    },
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock1"
            }
        ]
    },
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/ComposedSystem/Actions/ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOff",
                "GracefulShutdown",
                "GracefulRestart",
                "ForceRestart",
                "Nmi",
                "ForceOn",
                "PushPowerButton"
            ]
        },
        "#ComputerSystem.AddResourceBlock": {
            "target": "/redfish/v1/Systems/Expandable/Actions/ComputerSystem.AddResourceBlock"
        },
        "#ComputerSystem.RemoveResourceBlock": {
            "target": "/redfish/v1/Systems/Expandable/Actions/ComputerSystem.RemoveResourceBlock"
        }
    }
}
```

In the previous example, the `ComputerSystem` resource shows that it's a physical system since the `SystemType` is set to `Physical`. However, the presence of the `Composition` property with `ExpandableSystem` in its `UseCases` array shows that a user can assign more resources to the system by either modifying the `ResourceBlocks` array in `Links` or (performing the compose action)(#compose-action). The same operations are used to remove the additional resources.

Also in the previous example there is a reference to a `ResourceBlock` resource named `ComputeBlock1` found in the `Links` property. A user can follow this link to find other composition information about the resource, in particular the `CompositionStatus` property. Expandable resources follow the same state flows as other resource blocks. Like other

resource blocks, they start off in the `Unused` state and transition to the `Composed` state once additional resource blocks are assigned to it. Later, when all resource blocks are removed from the expandable resource, it transitions back to the `Unused` state.

4 Workflows to perform a composition request

The Identify whether the Redfish service supports composition workflow support composition requests.

There are two workflows for a client to create new composed resourcest: a specific composition workflow and a constrained composition workflow. Each workflow demonstrates how a client can create a composed system with the Redfish composition service.

Additional workflows describe other operations a client can perform:

- Modify a composed resource: Add or remove resources from a composed resource.
- Delete a composed resource: Delete a composed resource when it's no longer needed.

The examples assume the client provides valid authentication headers for performing the requests.

4.1 Identify whether the Redfish service supports composition

Clients should always start at the root: `/redfish/v1/`

1. Read the service root.
    i. Find the `CompositionService` property.
    ii. Perform a `GET` operation on the URI specified by that property.
    iii. Check that the value of the `ServiceEnabled` property contains `true`.

General flow diagram:

```
Client|                                          | Redfish service
      |---- GET /redfish/v1/CompositionService ----->|
      |<--- { ..., "ServiceEnabled": true, ... } <---|
```

4.2 Specific composition workflow

To perform specific composition requests, the client needs to understand the composition model reported by the composition service by reading the resource block and resource zone collections. This relationship will be used to build the composition request in the last step of the workflow.

4.2.1 Read the resource blocks

1. Perform a `GET` on the URI for the `CompositionService` resource.

2. Look for the `ResourceBlocks` property and perform a `GET` on the contained URI to get a list of all resource blocks.
3. To access details of a particular resource block, perform a `GET` on the associated URI listed for a given entry in the `Members` array.
4. The `CompositionStatus` property in each resource block identifies the availability of the resource block in composition requests.
    - Clients should take note of this when making decisions on what resource blocks to use in a composition request.
    - Depending on what's contained in the `CompositionStatus` property, a given resource block may not be currently available for composition.

Example `ResourceBlockCollection` resource:


```
{
    "@odata.type": "#ResourceBlockCollection.ResourceBlockCollection",
    "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks",
    "Name": "Resource Block Collection",
    "Members@odata.count": 9,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock1"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock2"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock4"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock5"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock6"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock7"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/NetworkBlock8"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/OffloadBlock9"
        }
    ]
}
```

4.2.2 Read the resource zones

1. Perform a `GET` on the URI for the `CompositionService` resource.
2. Look for the `ResourceZones` property and perform a `GET` on the contained URI to get a list of all resource zones.
3. To access details of a particular resource zone, perform a `GET` on the associated URI listed for a given entry in the `Members` array.

Example `ZoneCollection` resource:


```
{
    "@odata.type": "#ZoneCollection.ZoneCollection",
    "@odata.id": "/redfish/v1/CompositionService/ResourceZones",
    "Name": "Resource Zone Collection",
    "Members@odata.count": 2,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceZones/1"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceZones/2"
        }
    ]
}
```


4.2.3 Read the collection capabilities for the target resource collection

1. Perform a `GET` on the target resource collection for the composition request.
   - For example, when creating a composed system, perform a `GET` on `/redfish/v1/Systems`.
2. Look for the `@Redfish.CollectionCapabilities` annotation.
3. Iterate through the `Capabilities` array until an applicable value for `UseCase` is found.
   - For example, when creating a composed system with a specific composition request, look for the value `ComputerSystemComposition`.

Example collection capabilities for a `ComputerSystemCollection` resource:


```
{
    "@odata.id": "/redfish/v1/Systems",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [
```

```
        {
            "@odata.id": "/redfish/v1/Systems/ComposedSystem"
        }
    ],
    "@Redfish.CollectionCapabilities": {
        "@odata.type": "#CollectionCapabilities.v1_4_0.CollectionCapabilities",
        "Capabilities": [
            {
                "CapabilitiesObject": {
                    "@odata.id": "/redfish/v1/Systems/Capabilities"
                },
                "UseCase": "ComputerSystemComposition"
            }
        ]
    }
}
```


4.2.4 Read the capabilities object

1. Perform a `GET` on the URI listed in the `CapabilitiesObject` property for the matching capability.

Example capabilities object for a specific composition of a `ComputerSystem` resource:


```
{
    "@odata.id": "/redfish/v1/Systems/Capabilities",
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "Id": "Capabilities",
    "Name": "Capabilities for the system collection",
    "Name@Redfish.RequiredOnCreate": true,
    "Name@Redfish.SetOnlyOnCreate": true,
    "Description@Redfish.OptionalOnCreate": true,
    "Description@Redfish.SetOnlyOnCreate": true,
    "HostName@Redfish.OptionalOnCreate": true,
    "HostName@Redfish.UpdatableAfterCreate": true,
    "Boot@Redfish.OptionalOnCreate": true,
    "Boot": {
        "BootSourceOverrideEnabled@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideEnabled@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideTarget@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "Usb",
            "Hdd"
        ]
    },
```

```
    "Links@Redfish.RequiredOnCreate": true,
    "Links": {
        "ResourceBlocks@Redfish.RequiredOnCreate": true,
        "ResourceBlocks@Redfish.UpdatableAfterCreate": true
    },
    "@Redfish.ResourceBlockLimits": {
        "MinCompute": 1,
        "MaxCompute": 1,
        "MaxStorage": 8
    },
}
```

4.2.5 Create the composed resource

1. Using all the properties that were annotated with `RequiredOnCreate`, build a create (`POST`) request body to send to the target resource collection.
   - In the example of the previous step, only `Name` and `ResourceBlocks` found in `Links` are required.
   - The Redfish service may accept other properties as part of the request so they do not need to be updated later.
2. The `Location` HTTP header in the service response contains the URI of the composed resource.

Example client request:

```
POST /redfish/v1/Systems HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
OData-Version: 4.0

{
    "Name": "Sample Composed System",
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock0"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock2"
            }
        ]
    }
}
```

Example service response:

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
Location: /redfish/v1/Systems/NewSystem

{
    "@odata.id": "/redfish/v1/Systems/NewSystem",
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "Id": "NewSystem",
    "Name": "Sample Composed System",
    "Processors": {
        "@odata.id": "/redfish/v1/Systems/NewSystem/Processors",
    },
    "Memory": {
        "@odata.id": "/redfish/v1/Systems/NewSystem/Memory",
    },
    "Storage": {
        "@odata.id": "/redfish/v1/Systems/NewSystem/Storage",
    },
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock0"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock2"
            }
        ]
    },
    <Other ComputerSystem properties>
}
```

The previous client request shows a specific composition request to create a new composed system found at `/redfish/v1/Systems`. In the request, the client is specifying resource blocks `ComputeBlock0` and `DriveBlock2` are desired for this composition. In the previous service response, the service responded with a successful `201 Created` response, and indicated that the new composed system can be found at `/redfish/v1/Systems/NewSystem`.

4.3 Constrained composition workflow

To perform constrained composition requests, the client is not required to understand the relationship between resource blocks and resource zones. The specific selection of resource blocks is delegated to the composition service based on the specified criteria provided by the client, such as quantities and capacities of devices for the composed resource.

4.3.1 Determine the provisioning capabilities of the service

1. Perform a `GET` on the URI for the `CompositionService` resource.

2. Look for the `AllowOverprovisioning` property.
   - If this property is missing or contains `false`, `@Redfish.AllowOverprovisioning` is not allowed to be supplied in the composition request.
3. Look for the `AllowZoneAffinity` property.
   - If this property is missing or contains `false`, `@Redfish.ZoneAffinity` is not allowed to be supplied in the composition request.

4.3.2 Read the collection capabilities for the target resource collection

1. Perform a `GET` on the target resource collection for the composition request.
   - For example, when creating a composed system, perform a `GET` on `/redfish/v1/Systems`.
2. Look for the `@Redfish.CollectionCapabilities` annotation.
3. Iterate through the `Capabilities` array until an applicable value for `UseCase` is found.
   - For example, when creating a composed system with a constrained composition request, look for the value `ComputerSystemConstrainedComposition`.

Example collection capabilities for a `ComputerSystemCollection` resource:

```
{
    "@odata.id": "/redfish/v1/Systems",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Name": "Computer System Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/ComposedSystem"
        }
    ],
    "@Redfish.CollectionCapabilities": {
        "@odata.type": "#CollectionCapabilities.v1_4_0.CollectionCapabilities",
        "Capabilities": [
            {
                "CapabilitiesObject": {
                    "@odata.id": "/redfish/v1/Systems/ConstrainedCompositionCapabilities"
                },
                "UseCase": "ComputerSystemConstrainedComposition"
            }
        ]
    }
}
```

4.3.3 Read the capabilities object

1. Perform a `GET` on the URI listed in the `CapabilitiesObject` property for the matching capability.

Example capabilities object for a constrained composition of a `ComputerSystem` resource:


```
{
    "@odata.context": "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "@odata.id": "/redfish/v1/Systems/ConstrainedCompositionCapabilities",
    "Id": "ConstrainedCompositionCapabilities",
    "Name": "Capabilities for the Zone",
    "Name@Redfish.RequiredOnCreate": true,
    "Name@Redfish.SetOnlyOnCreate": true,
    "Description@Redfish.OptionalOnCreate": true,
    "Description@Redfish.SetOnlyOnCreate": true,
    "HostName@Redfish.OptionalOnCreate": true,
    "HostName@Redfish.UpdatableAfterCreate": true,
    "Boot@Redfish.OptionalOnCreate": true,
    "Boot": {
        "BootSourceOverrideEnabled@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideEnabled@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.OptionalOnCreate": true,
        "BootSourceOverrideTarget@Redfish.UpdatableAfterCreate": true,
        "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "Usb",
            "Hdd"
        ]
    },
    "Processors@Redfish.RequiredOnCreate": true,
    "Processors": {
        "@odata.type": "#ProcessorCollection.ProcessorCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#Processor.v1_1_0.Processor",
                "@Redfish.RequestedCountRequired": true,
                "ProcessorType@Redfish.RequiredOnCreate": true,
                "TotalCores@Redfish.RequiredOnCreate": true,
                "Model@Redfish.OptionalOnCreate": true,
                "InstructionSet@Redfish.OptionalOnCreate": true,
                "AchieveableSpeedMHz@Redfish.OptionalOnCreate": true
            }
        ]
    },
    "Memory@Redfish.RequiredOnCreate": true,
    "Memory": {
        "@odata.type": "#MemoryCollection.MemoryCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#Memory.v1_1_0.Memory",
                "@Redfish.RequestedCountRequired": true,
```

```
                "MemoryType@Redfish.RequiredOnCreate": true,
                "MemoryDeviceType@Redfish.OptionalOnCreate": true,
                "CapacityMiB@Redfish.RequiredOnCreate": true,
                "SpeedMHz@Redfish.OptionalOnCreate": true,
                "DataWidthBits@Redfish.OptionalOnCreate": true,
                "BusWidthBits@Redfish.OptionalOnCreate": true
            }
        ]
    },
    "SimpleStorage@Redfish.OptionalOnCreate": true,
    "SimpleStorage": {
        "@odata.type": "#SimpleStorageCollection.SimpleStorageCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#SimpleStorage.v1_2_0.SimpleStorage",
                "@Redfish.RequestedCountRequired": true,
                "Devices@Redfish.RequiredOnCreate": true,
                "Devices": {
                    "@Redfish.RequestedCountRequired": true,
                    "CapacityBytes@Redfish.RequiredOnCreate": true
                }
            }
        ]
    },
    "Storage@Redfish.OptionalOnCreate": true,
    "Storage": {
        "@odata.type": "#StorageCollection.StorageCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#Storage.v1_3_0.Storage",
                "@Redfish.RequestedCountRequired": true,
                "StorageControllers@Redfish.OptionalOnCreate": true,
                "StorageControllers": [
                    {
                        "@Redfish.RequestedCountRequired": true,
                        "SupportedControllerProtocols@Redfish.RequiredOnCreate": true
                    }
                ],
                "Drives@Redfish.RequiredOnCreate": true,
                "Drives": [
                    {
                        "@odata.type": "#Drive.v1_2_0.Drive",
                        "@Redfish.RequestedCountRequired": true,
                        "CapacityBytes@Redfish.RequiredOnCreate": true
                    }
                ]
            }
        ]
```

```
    },
    "EthernetInterfaces@Redfish.OptionalOnCreate": true,
    "EthernetInterfaces": {
        "@odata.type": "#EthernetInterfaceCollection.EthernetInterfaceCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#EthernetInterface.v1_3_0.EthernetInterface",
                "@Redfish.RequestedCountRequired": true,
                "SpeedMbps@Redfish.RequiredOnCreate": true,
                "FullDuplex@Redfish.OptionalOnCreate": true
            }
        ]
    },
    "NetworkInterfaces@Redfish.OptionalOnCreate": true,
    "NetworkInterfaces": {
        "@odata.type": "#NetworkInterfaceCollection.NetworkInterfaceCollection",
        "Members@Redfish.RequiredOnCreate": true,
        "Members": [
            {
                "@odata.type": "#NetworkInterface.v1_1_0.NetworkInterface",
                "@Redfish.RequestedCountRequired": true,
                "NetworkPorts@Redfish.RequiredOnCreate": true,
                "NetworkPorts": {
                    "@odata.type": "#NetworkPortCollection.NetworkPortCollection",
                    "Members@Redfish.RequiredOnCreate": true,
                    "Members": [
                        {
                            "@odata.type": "#NetworkPort.v1_1_0.NetworkPort",
                            "@Redfish.RequestedCountRequired": true,
                            "ActiveLinkTechnology@Redfish.RequiredOnCreate": true,
                            "SupportedLinkCapabilities@Redfish.OptionalOnCreate": true,
                            "SupportedLinkCapabilities": {
                                "LinkSpeedMbps@Redfish.RequiredOnCreate": true
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

The structure of the capabilities object for a constrained composition is an expanded object that represents the resource the client can compose. In the above example, the properties `Processors` and `Memory` are expanded, and showing `ProcessorCollection` resource and `MemoryCollection` resource representations respectively. These expanded objects do not contain extraneous information required under normal circumstances, such as `@odata.id`, in order to reduce the information to only what the client requires to form the composition request.

4.3.4 Constructing the composition request

In a constrained composition request, the request includes structures for the devices or other subordinate resources of the desired composed resource. For `ComputerSystem` resources, this includes processors, memory, storage and network interfaces. Each structure includes the annotation `@Redfish.RequestedCount`, which specifies the requested amount of a resource.

Based on the capabilities example in the previous step, the following structures can contain an enumeration annotation and may required sub-properties.

- The processors request
  - The `@Redfish.RequestedCount` is required.
  - The `ProcessorType` and `TotalCores` properties are required.
- The memory request
  - The `@Redfish.RequestedCount` is required.
  - The `MemoryType` and `CapacityMiB` properties are required.
- The storage request
  - The `@Redfish.RequestedCount` is required.
  - Either the `SimpleStorage` or `Storage` property may be present, or neither.
  - If present, `CapacityBytes` property is required.
- The network interface request
  - The `@Redfish.RequestedCount` is required.
  - Either `EthernetInterfaces` or `NetworkInterfaces` property may be present, or neither.
  - If present, `SpeedMbps` or `LinkSpeedMbps` properties are required.

For example, the following requests 4 CPUs and 2 FPGAs for processors. Other properties in the request can further characterize the desired processors.

The composition request should be kept simple in order to increase the probability of a successful composition. Overly-constrained requests are less likely to be fulfilled.

Example request to describe a desired set of processors:

```
{
    "Processors": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "ProcessorType": "CPU",
                "TotalCores": 16
            },
            {
```

```
                "@Redfish.RequestedCount": 2,
                "ProcessorType": "FPGA",
                "TotalCores": 16
            }
        ]
    }
}
```

4.3.5 Create the composed resource

1. Using the constructed request from the previous step, perform a `POST` operation on target resource collection.
2. The `Location` HTTP header in the service response contains the URI of the composed resource.

Example client request:

```
POST /redfish/v1/Systems HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
OData-Version: 4.0

{
    "Name": "My Computer System",
    "@Redfish.ZoneAffinity": "1",
    "PowerState": "On",
    "BiosVersion": "P79 v1.00 (09/20/2013)",
    "Processors": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "@Redfish.AllowOverprovisioning": true,
                "ProcessorType": "CPU",
                "ProcessorArchitecture": "x86",
                "InstructionSet": "x86-64",
                "MaxSpeedMHz": 3700,
                "TotalCores": 8,
                "TotalThreads": 16
            },
            {
                "@Redfish.RequestedCount": 4,
                "@Redfish.AllowOverprovisioning": false,
                "ProcessorType": "FPGA",
                "ProcessorArchitecture": "x86",
                "InstructionSet": "x86-64",
                "MaxSpeedMHz": 3700,
                "TotalCores": 16
```

```
            }
        ]
    },
    "Memory": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "MaxTDPMilliWatts": [ 12000 ],
                "CapacityMiB": 8192,
                "DataWidthBits": 64,
                "BusWidthBits": 72,
                "ErrorCorrection": "MultiBitECC",
                "MemoryType": "DRAM",
                "MemoryDeviceType": "DDR4",
                "BaseModuleType": "RDIMM",
                "MemoryMedia": [ "DRAM" ]
            }
        ]
    },
    "SimpleStorage": {
        "Members" : [
            {
                "@Redfish.RequestedCount": 6,
                "Devices": [
                    {
                        "CapacityBytes": 322122547200
                    }
                ]
            }
        ]
    },
    "EthernetInterfaces": {
        "Members": [
            {
                "@Redfish.RequestedCount": 1,
                "SpeedMbps": 1000,
                "FullDuplex": true,
                "NameServers": [
                    "names.redfishspecification.org"
                ],
                "IPv4Addresses": [
                    {
                        "SubnetMask": "255.255.252.0",
                        "AddressOrigin": "Dynamic",
                        "Gateway": "192.168.0.1"
                    }
                ]
            }
        ]
    }
}
```

Example service response:

```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
Location: /redfish/v1/Systems/NewSystem2

{
    "@odata.id": "/redfish/v1/Systems/NewSystem2",
    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
    "Id": "NewSystem",
    "Name": "Sample Composed System",
    "PowerState": "On",
    "BiosVersion": "P79 v1.00 (09/20/2013)",
    "Processors": {
        "@odata.id": "/redfish/v1/Systems/NewSystem2/Processors",
    },
    "Memory": {
        "@odata.id": "/redfish/v1/Systems/NewSystem2/Memory",
    },
    "SimpleStorage": {
        "@odata.id": "/redfish/v1/Systems/NewSystem2/SimpleStorage",
    },
    "EthernetInterfaces": {
        "@odata.id": "/redfish/v1/Systems/NewSystem2/EthernetInterfaces",
    },
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock0"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock2"
            }
        ]
    },
    <Other ComputerSystem properties>
}
```

The previous client request shows a constrained composition request to create a new composed system found at `/redfish/v1/Systems`. In the request, the client is specifying a new `ComputerSystem` resource with 4 CPUs, 4 FPGAs, 4GB of memory, 6 322GB local drives, and a 1GB Ethernet interface. The usage of the annotation `@Redfish.AllowOverprovisioning` permits the Redfish service to supply more resources than what was requested. The usage of the annotation `@Redfish.ZoneAffinity` indicates the client wants the components for the composition to be all selected from the resource zone that contains the value `"1"` for the `Id` property.

In the previous service response, the service responded with a successful `201 Created` response, and indicated that the new composed system can be found at `/redfish/v1/Systems/NewSystem2`.

4.4 Modify a composed resource

To modify a composed resource, a client can perform a `PATCH` operation to request changes to the composed resource or use actions to perform modifications.

4.4.1 PATCH method for modifying

To modify a composed resource with a `PATCH` operation, the client specifies modifications to the `ResourceBlocks` array property inside the `Links` property of the composed resource. The "PATCH on array properties" clause in the Redfish Specification describes methods for inserting, removing, and appending to array properties.

Example client request:

```
PATCH /redfish/v1/Systems/NewSystem HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
OData-Version: 4.0

{
    "Links": {
        "ResourceBlocks": [
            {},
            {},
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/NetworkBlock8"
            }
        ]
    }
}
```

The previous example will preserve the existing resource blocks in the composed resource for array elements 0 and 1, and it will add the `NetworkBlock8` resource block to array element 2.

4.4.2 Actions for modifying

Composed resources that support using actions for modification will have them advertised in the `GET` response for the resource.

Example `ComputerSystem` resource with modification actions:

```
{
    "@odata.id": "/redfish/v1/Systems/Composed",
```

```
    "Id": "Composed",
    "Name": "Sample Composed System",
    "SystemType": "Composed",
    "Links": {
        "ResourceBlocks": [
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock1"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3"
            },
            {
                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock4"
            }
        ]
    },
    "Actions": {
        "#ComputerSystem.AddResourceBlock": {
            "target": "/redfish/v1/Systems/Composed/Actions/ComputerSystem.AddResourceBlock"
        },
        "#ComputerSystem.RemoveResourceBlock": {
            "target": "/redfish/v1/Systems/Composed/Actions/ComputerSystem.RemoveResourceBlock"
        }
    },
    ...
}
```

In the previous example, the system `Composed` supports two actions: `ComputerSystem.AddResourceBlock` and `ComputerSystem.RemoveResourceBlock`. A client is able to modify the system by performing `POST` requests to the URI specified by the `target` properties.

Example `AddResourceBlock` request:

```
POST /redfish/v1/Systems/Composed/Actions/ComputerSystem.AddResourceBlock HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
OData-Version: 4.0

{
    "ResourceBlock": {
        "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/NetworkBlock8"
    },
    "ResourceBlockETag": "6e83d4d5f1b8d93fed866876a220c0ab",
    "ComputerSystemETag": "31171b07d6e2733c8368c54ab1857456"
}
```

In the previous example, the client is requesting to add resource block `NetworkBlock8` to the system

`ComposedSystem`. It also uses the optional parameters `ResourceBlockETag` and `ComputerSystemETag` to help protect the usage of the system and resource block in multi-client scenarios so that the action is not carried out of if the specified ETags do not match the state of the resources.

4.5 Delete a composed resource

The client can retire or decompose a composed resource by performing a `DELETE` operation on the composed resource.

Example client request:

```
DELETE /redfish/v1/Systems/NewSystem HTTP/1.1
```

The previous example will request that the composed system called `NewSystem` be retired. When this happens, this will free the resource blocks being used by the system so that they can be used in future compositions.

5 Free pool and active pool

The `FreePool` and `ActivePool` properties in the `CompositionService` resource contain links to collections of resource blocks that represent the resource blocks that are free, the free pool, and resource blocks that are in use, the active pool, respectively. The members of these collections will contain resource blocks found in the collection referenced by the `ResourceBlocks` property. However, special semantics are applied to help clients understand which resource blocks are available to them and which ones are currently in use.

Resource blocks that are participating in at least one composition will be found in the active pool and resource blocks that are not participating in any compositions will be found in the free pool. As composed resources are created, modified, and deleted, the members of these collections will move between the free and active pools. For example, if resource block `Drive37` is not participating in any compositions, it will be found in the free pool. Later, if a request is made to add `Drive37` to a composed system with one of the methods to modify a composed resource, it will be removed from the free pool and can be found in the active pool.

Services are required to enforce user-specific privileges when showing the contents of the free pool and active pool. This is to ensure users in different domains can only see their available resource blocks and never the resource blocks of other users. For example, "customer A" could be allocated 10 resource blocks named `Block00` to `Block09` and "customer B" could be allocated 10 resource blocks named `Block10` to `Block19`. If "customer A" queries the free pool or active pools, they will only see resource blocks `Block00` to `Block09` in responses and never the resource blocks allocated to "customer B".

Example free pool response for "customer A":


```
{
    "@odata.type": "#ResourceBlockCollection.ResourceBlockCollection",
    "@odata.id": "/redfish/v1/CompositionService/FreePool",
    "Name": "Free Pool Resource Block Collection",
    "Members@odata.count": 5,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block00"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block06"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block07"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block08"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block09"
```

```
        }
    ]
}
```

Example active pool response for "customer A":

```
{
    "@odata.type": "#ResourceBlockCollection.ResourceBlockCollection",
    "@odata.id": "/redfish/v1/CompositionService/ActivePool",
    "Name": "Active Pool Resource Block Collection",
    "Members@odata.count": 5,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block01"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block02"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block03"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block04"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block05"
        }
    ]
}
```

Example free pool response for "customer B":

```
{
    "@odata.type": "#ResourceBlockCollection.ResourceBlockCollection",
    "@odata.id": "/redfish/v1/CompositionService/FreePool",
    "Name": "Free Pool Resource Block Collection",
    "Members@odata.count": 7,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block11"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block12"
        },
        {
```

```
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block13"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block14"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block15"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block18"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block19"
        }
    ]
}
```

Example active pool response for "customer B":

```
{
    "@odata.type": "#ResourceBlockCollection.ResourceBlockCollection",
    "@odata.id": "/redfish/v1/CompositionService/ActivePool",
    "Name": "Active Pool Resource Block Collection",
    "Members@odata.count": 3,
    "Members": [
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block10"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block16"
        },
        {
            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/Block17"
        }
    ]
}
```

In the previous examples, 5 resource blocks that belong to "customer A" are in the free pool and 5 resource blocks that belong to "customer A" are in the active pool. These responses do not show resource blocks that belong to other users. Note that the members of the collections all refer back to `/redfish/v1/CompositionService/ResourceBlocks`.

6 Compose action

`#CompositionService.Compose` action allows users to perform bulk operations that are specified in a manifest provided by the user. The manifest contains one or more composition requests.

The following parameters are defined for the compose action:

Name	Description
`RequestFormat`	The type of payload in the action request. `Manifest` is the only value allowed.
`RequestType`	How to process the payload in the action request. See below.
`Manifest`	The set of operations to in the request. See Manifest format.
`ReservationId`	The identifer of the reservation to apply. See Reservations.

The `RequestType` parameter dictates how to process the contents of the manifest. Users can specify one of the following values:

- `Preview` : The service responds with what it would do if it applied the requested set of operations, but does not make any changes to the service or hardware. This can be useful if the client wants to verify the outcome is desirable or it needs to adjust the requested operations.
- `PreviewReserve` : The same as `Preview` , but also reserves resources that would be consumed when the operations are enacted. This is to ensure another user doesn't claim the resources in the meantime. See Reservations.
- `Apply` : Applies the requested set of operations. Can optionally include the `ReservationId` parameter if applying a set of operations from a previous `PreviewReserve` request.

The response of the action contains the outcome of the operations, or a preview of the outcome of the operations. See Response format.

6.1 Reservations

The compose action allows a user to reserve resources consumed for the requested set of operations prior to enacting them. This is to allow a user to inspect the outcome of the request to ensure it meets their criteria while also preventing other users from consuming the reserved resources. This is invoked when the user specifies `PreviewReserve` as the `RequestType` parameter. The action response when this type of request is made will contain a `ReservationId` property, which represents the reservation identifier for this request. The value of `ReservationId` is used in the subsequent request to apply the reservation, if the user is satisfied with the proposed outcome of the request.

The `CompositionReservations` property in the composition service contains the URI to a resource collection with all reservations the service is tracking. Each `CompositionReservation` resource contains:

- `ReservationTime` : The time when the reservation was created.
- `Client` : The client that created the reservation.
- `ReservedResourceBlocks` : The resource blocks reserved by the reservation.
- `Manifest` : A copy of the manifest provided by the user when making the reservation.

To cancel a reservation, perform a `DELETE` operation on the appropriate `CompositionReservation` resource. This will free the a reserved resources for others to consume.

The `ReservationDuration` property in the composition service controls how long reservations can remain outstanding. When a reservation's life exceeds this time, the service automatically cancels the reservation.

Once a user is satisfied with the results of the `PreviewReserve` request, they can enact the reservation with the compose action by:

- Specifying `Apply` as the `RequestType` .
- Providing the reservation identifier from the response of the `PreviewReserve` request as the `ReservationId` parameter.

When a reservation is applied, its respective `CompositionReservation` resource is also deleted.

6.2 Manifest format

The `Manifest` parameter in the compose action is a JSON object that describes a set of operations to perform. It consists of the following properties:

Property	Description
`Description`	A description of the manifest.
`Timestamp`	The date and time when the manifest was created.
`Expand`	Indicates if responses are expanded to show subordinate resources.
`Stanzas`	An array of objects containing the operations to perform. Each stanza represents one operation.

Each object in the `Stanzas` property contains the following properties:

Property	Description
`StanzaType`	The type of stanza. The value dictates the contents of the `Request` property.

Property	Description
`OEMStanzaType`	The OEM-defined type of stanza. This property is only used if `StanzaType` contains `OEM` to further describe the OEM usage of the `Request` property.
`StanzaId`	A unique identifier for the stanza.
`Request`	The request body for the operation. The contents depend on the value of the `StanzaType` property.

The `StanzaType` property can be one of the following values:

Value	Contents of `Request`
`ComposeSystem`	A `ComputerSystem` resource that describes a composed system to create. This could contain either a specific composition or constrained composition request.
`DecomposeSystem`	A reference object, which only consists of an `@odata.id` property, that references the URI of a composed system to decompose.
`ComposeResource`	A `ResourceBlock` resource that describes a composed resource block to create from other resource blocks. This could contain either a specific composition or constrained composition request.
`DecomposeResource`	A reference object, which only consists of an `@odata.id` property, that references the URI of a composed resource block to decompose.
`OEM`	The contents are OEM-specific, and further described by the `OEMStanzaType` property.

6.3 Response format

The response for the compose action contains the following properties:

Name	Description
`RequestFormat`	The type of payload specified in the original action request. `Manifest` is the only value allowed.
`RequestType`	The type of request specified in the original action request.
`Manifest`	The set of operations provided in the original action request. See Manifest format.
`ReservationId`	The identifer of the reservation that was created if `RequestType` contains `PreviewReserve`. See Reservations.

The contents of `Manifest` also include a `Response` property in each object of the `Stanzas` array. This contains the response body of each operation. The `StanzaType` property in each object dictates the contents of the `Response` object.

Value	Contents of `Request`
`ComposeSystem`	A `ComputerSystem` resource that represents the composed system that was created or an error response if the request could not be satisfied.
`DecomposeSystem`	A `ComputerSystem` resource that represents the composed system that was decomposed or an error response if the request could not be satisfied.
`ComposeResource`	A `ResourceBlock` resource that represents the composed resource block that was created or an error response if the request could not be satisfied.
`DecomposeResource`	A `ResourceBlock` resource that represents the composed resource block that was decomposed or an error response if the request could not be satisfied.
`OEM`	The contents are OEM-specific, and further described by the `OEMStanzaType` property.

6.4 Examples

The following examples show the common variations of invoking the `#CompositionService.Compose` action.

6.4.1 Preview

This example shows a request to preview the outcome of the requested operations. Only one operation is requested: a specific composition to create a composed system from resource blocks `CB0` and `DB2`. The response shows if the request were to be applied, it would successfully create a new composed system named `ComposedSys1`.

Action request:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "Preview",
    "Manifest": {
        "Description": "Specific composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "Name": "Sample Composed System",
                    "Links": {
                        "ResourceBlocks": [
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                            },
```

```
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                        }
                    ]
                }
            }
        }
    ]
}
}
```

Action response:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "Preview",
    "Manifest": {
        "Description": "Specific composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "Links": {
                        "ResourceBlocks": [
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                            },
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                            }
                        ]
                    }
                },
                "Response": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1",
                    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
                    "Id": "ComposedSys1",
                    "Name": "Computer system composed from Compute1",
                    "Processors": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/Processors"
                    },
                    "Memory": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/Memory"
                    },
                    "EthernetInterfaces": {
```

```
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/EthernetInterfaces"
                },
                "Storage": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Storage"
                },
                "Links": {
                    "ResourceBlocks": [
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                        }
                    ]
                }
            }
        }
    ]
}
```

6.4.2 Preview and reserve

This example shows a request to preview the outcome of the requested operations and reserve the resources that would be consumed. Only one operation is requested: a specific composition to create a composed system from resource blocks `CB0` and `DB2`. The response shows if the request were to be applied, it would successfully create a new composed system named `ComposedSys1` and it reserved the resources with the reservation named `Compute1-0415247`.

Action request:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "PreviewReserve",
    "Manifest": {
        "Description": "Specific composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "Name": "Sample Composed System",
                    "Links": {
                        "ResourceBlocks": [
```

```
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                        }
                    ]
                }
            }
        ]
    }
}
```

Action response:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "PreviewReserve",
    "ReservationId": "Compute1-0415247",
    "Manifest": {
        "Description": "Specific composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "Links": {
                        "ResourceBlocks": [
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                            },
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                            }
                        ]
                    }
                },
                "Response": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1",
                    "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
                    "Id": "ComposedSys1",
                    "Name": "Computer system composed from Compute1",
                    "Processors": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/Processors"
                    },
```

```
                    "Memory": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/Memory"
                    },
                    "EthernetInterfaces": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/EthernetInterfaces"
                    },
                    "Storage": {
                        "@odata.id": "/redfish/v1/Systems/ComposedSys1/Storage"
                    },
                    "Links": {
                        "ResourceBlocks": [
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                            },
                            {
                                "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```


6.4.3 Apply from reservation

This example builds upon the previous example. The user is satisfied with the response from the previous example and makes a request to apply the requested operations to the reserved resources by providing the reservation identifier `Compute1-0415247`.

Action request:


```
{
    "RequestFormat": "Manifest",
    "RequestType": "Apply",
    "ReservationId": "Compute1-0415247"
}
```


Action response:


```
{
    "RequestFormat": "Manifest",
    "RequestType": "Apply",
```

```
"Manifest": {
    "Description": "Specific composition example",
    "Timestamp": "2023-06-22T10:35:16+06:00",
    "Expand": "None",
    "Stanzas": [
        {
            "StanzaType": "ComposeSystem",
            "StanzaId": "Compute1",
            "Request": {
                "Links": {
                    "ResourceBlocks": [
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                        }
                    ]
                }
            },
            "Response": {
                "@odata.id": "/redfish/v1/Systems/ComposedSys1",
                "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
                "Id": "ComposedSys1",
                "Name": "Computer system composed from Compute1",
                "Processors": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Processors"
                },
                "Memory": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Memory"
                },
                "EthernetInterfaces": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/EthernetInterfaces"
                },
                "Storage": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Storage"
                },
                "Links": {
                    "ResourceBlocks": [
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB0"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB2"
                        }
                    ]
                }
            }
        }
    ]
```

```
    }
}
```

6.4.4 Apply without reservation

This example shows a request to apply the requested operations from available resources without using a reservation. Only one operation is requested: a constrained composition to create a composed system with 4 CPUs, 4 FPGAs, 4GB of memory, 6 322GB local drives, and a 1GB Ethernet interface. The response shows shows a new composed system was created with the name `ComposedSys1`.

Action request:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "Apply",
    "Manifest": {
        "Description": "Constrained composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "@Redfish.ZoneAffinity": "1",
                    "PowerState": "On",
                    "BiosVersion": "P79 v1.00 (09/20/2013)",
                    "Processors": {
                        "Members": [
                            {
                                "@Redfish.RequestedCount": 4,
                                "@Redfish.AllowOverprovisioning": true,
                                "ProcessorType": "CPU",
                                "ProcessorArchitecture": "x86",
                                "InstructionSet": "x86-64",
                                "MaxSpeedMHz": 3700,
                                "TotalCores": 8,
                                "TotalThreads": 16
                            },
                            {
                                "@Redfish.RequestedCount": 4,
                                "@Redfish.AllowOverprovisioning": false,
                                "ProcessorType": "FPGA",
                                "ProcessorArchitecture": "x86",
                                "InstructionSet": "x86-64",
                                "MaxSpeedMHz": 3700,
```

```
                "TotalCores": 16
            }
        ]
    },
    "Memory": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "MaxTDPMilliWatts": [ 12000 ],
                "CapacityMiB": 8192,
                "DataWidthBits": 64,
                "BusWidthBits": 72,
                "ErrorCorrection": "MultiBitECC",
                "MemoryType": "DRAM",
                "MemoryDeviceType": "DDR4",
                "BaseModuleType": "RDIMM",
                "MemoryMedia": [ "DRAM" ]
            }
        ]
    },
    "SimpleStorage": {
        "Members" : [
            {
                "@Redfish.RequestedCount": 6,
                "Devices": [
                    {
                        "CapacityBytes": 322122547200
                    }
                ]
            }
        ]
    },
    "EthernetInterfaces": {
        "Members": [
            {
                "@Redfish.RequestedCount": 1,
                "SpeedMbps": 1000,
                "FullDuplex": true,
                "NameServers": [
                    "names.redfishspecification.org"
                ],
                "IPv4Addresses": [
                    {
                        "SubnetMask": "255.255.252.0",
                        "AddressOrigin": "Dynamic",
                        "Gateway": "192.168.0.1"
                    }
                ]
            }
        ]
```

```
                    }
                }
            }
        ]
    }
}
```

Action response:

```
{
    "RequestFormat": "Manifest",
    "RequestType": "Apply",
    "Manifest": {
        "Description": "Constrained composition example",
        "Timestamp": "2023-06-22T10:35:16+06:00",
        "Expand": "None",
        "Stanzas": [
            {
                "StanzaType": "ComposeSystem",
                "StanzaId": "Compute1",
                "Request": {
                    "@Redfish.ZoneAffinity": "1",
                    "PowerState": "On",
                    "BiosVersion": "P79 v1.00 (09/20/2013)",
                    "Processors": {
                        "Members": [
                            {
                                "@Redfish.RequestedCount": 4,
                                "@Redfish.AllowOverprovisioning": true,
                                "ProcessorType": "CPU",
                                "ProcessorArchitecture": "x86",
                                "InstructionSet": "x86-64",
                                "MaxSpeedMHz": 3700,
                                "TotalCores": 8,
                                "TotalThreads": 16
                            },
                            {
                                "@Redfish.RequestedCount": 4,
                                "@Redfish.AllowOverprovisioning": false,
                                "ProcessorType": "FPGA",
                                "ProcessorArchitecture": "x86",
                                "InstructionSet": "x86-64",
                                "MaxSpeedMHz": 3700,
                                "TotalCores": 16
                            }
                        ]
                    },
                    "Memory": {
```

```
                "Members": [
                    {
                        "@Redfish.RequestedCount": 4,
                        "MaxTDPMilliWatts": [ 12000 ],
                        "CapacityMiB": 8192,
                        "DataWidthBits": 64,
                        "BusWidthBits": 72,
                        "ErrorCorrection": "MultiBitECC",
                        "MemoryType": "DRAM",
                        "MemoryDeviceType": "DDR4",
                        "BaseModuleType": "RDIMM",
                        "MemoryMedia": [ "DRAM" ]
                    }
                ]
            },
            "SimpleStorage": {
                "Members" : [
                    {
                        "@Redfish.RequestedCount": 6,
                        "Devices": [
                            {
                                "CapacityBytes": 322122547200
                            }
                        ]
                    }
                ]
            },
            "EthernetInterfaces": {
                "Members": [
                    {
                        "@Redfish.RequestedCount": 1,
                        "SpeedMbps": 1000,
                        "FullDuplex": true,
                        "NameServers": [
                            "names.redfishspecification.org"
                        ],
                        "IPv4Addresses": [
                            {
                                "SubnetMask": "255.255.252.0",
                                "AddressOrigin": "Dynamic",
                                "Gateway": "192.168.0.1"
                            }
                        ]
                    }
                ]
            }
        },
        "Response": {
            "@odata.id": "/redfish/v1/Systems/ComposedSys1",
            "@odata.type": "#ComputerSystem.v1_20_1.ComputerSystem",
```

```
                "Id": "ComposedSys1",
                "Name": "Computer system composed from Compute1",
                "PowerState": "On",
                "BiosVersion": "P79 v1.00 (09/20/2013)",
                "Processors": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Processors"
                },
                "Memory": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/Memory"
                },
                "SimpleStorage": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/SimpleStorage"
                },
                "EthernetInterfaces": {
                    "@odata.id": "/redfish/v1/Systems/ComposedSys1/EthernetInterfaces"
                },
                "Links": {
                    "ResourceBlocks": [
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/CB5"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/FPGA3"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/FPGA6"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/FPGA7"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/FPGA8"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/FPGA9"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB8"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB10"
                        },
                        {
                            "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DB16"
                        }
                    ]
                }
            }
        }
    ]
```

```
    }
}
```

7 Appendix A: References

- "Composable System" and "Bladed Partitions" mockups: http://redfish.dmtf.org/redfish/v1
- `CompositionService` schema: http://redfish.dmtf.org/schemas/v1/CompositionService_v1.xml
- `ResourceBlock` schema: http://redfish.dmtf.org/schemas/v1/ResourceBlock_v1.xml
- `Zone` schema: http://redfish.dmtf.org/schemas/v1/Zone_v1.xml
- `CollectionCapabilities` schema: http://redfish.dmtf.org/schemas/v1/CollectionCapabilities_v1.xml

8 Appendix B: Change log

Version	Date	Description
1.3.0	2023-10-04	Made many changes for style consistency, grammar, and general clarity.
		Clarified how `ResourceBlockType` is an array and can represent a mix of devices.
		Removed statement about constrained composition users not needing to know about resource zones, which isn't really true.
		Added missing workflow step for constrained composition for determining the capabilities of the service.
		Added documentation for free pool and active pool usage.
		Added documentation for the `#CompositionService.Compose` action.
1.2.0	2018-12-11	Added documentation for usage of `@Redfish.ResourceBlockLimits` term.
		Added text in the constrained composition section to link to the appendix.
		Added expandable resources section.
		Added new methods for modifying composed resources.
1.1.0	2018-08-23	Added documentation for constrained composition requests.
		Updated modeling section to cover new properties added in DSP8010 2018.1 and 2018.2.
		Added guidance for implementers on different conditions to avoid when annotating properties in the capabilities object.
		Added recommended flow diagrams for the `CompositionState` property within a resource block.
1.0.0	2017-06-30	Initial release.