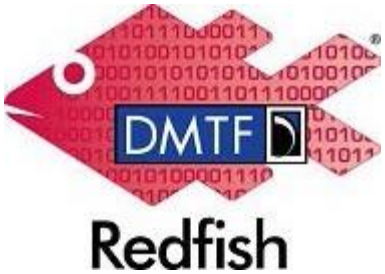

文書識別番号:DSP2050

日付:2018 年 8 月 3 日

バージョン:1.1.0

Redfish コンポーザビリティ・ホワイト・ペーパー

文書の種類:情報提供

文書の状態:公開

文書の言語:ja-JP

DMTF は、企業やシステムの管理および相互運用性を推進することに力を注いでいる、業界のメンバーから成る非営利団体である。メンバー、およびメンバー以外でも、出典を正しく表示することを条件に、DMTF の仕様と文書を複製することができる。DMTF の仕様は時折改定されることがあるため、特定のバージョンおよび公開日に、常に注意を払う必要がある。

本標準または標準案の特定の要素を実装することは、仮出願済特許権を含む第三者の特許権（本書では「特許権」と呼ぶ）の対象となることもある。DMTF は本標準のユーザーに対し、上記権利の存在について何ら表明するものではなく、上記第三者の特許権、特許権者または主張者の、いずれかまたはすべてを認識、公開、または特定する責任を負わない。また、上記権利、特許権者、主張者の不完全または不正確な特定、公開に対しても責任を負わない。DMTF は、いかなる相手に対して、いかなる方法または環境、またいかなる法論理においても、上記の第三者特許権を認識、公開、または特定しないことに対し何ら責任を負わず、上記第三者の標準に関する信頼性、またはその製品、プロトコル、試験方法論に組み込まれた標準に関しても何ら責任を負わない。DMTF は、上記標準の実装が知見できるか否かにかかわらず、上記標準を実装するいかなる相手に対しても、また、いかなる特許権者または主張者に対しても、何ら責任を負わない。また、DMTF は、公開後に標準が撤回または修正されることにより生じるコストや損失に対し何ら責任を負わず、また、標準を実装するいかなる相手からも、上記実装に対して特許権者が起こす、いずれかまたはすべての侵害の主張から何ら損害を受けず、免責されるものとする。

第三者が保有する特許権であって、DMTF 標準の実装に関連するかまたは影響を与える可能性があると特許権者が考え、すでに DMTF に通知済みである特許権に関する情報については、サイト http://www.dmtf.org/about/policies/disclosures.php を参照のこと。

このドキュメントの標準言語は英語である。英語以外の言語への翻訳が許可されている。

目次

前書き

Redfish コンポーザビリティ・ホワイト・ペーパーは、DMTF の Redfish Forum により準備されたものである。

DMTF は、企業やシステムの管理および相互運用性を推進することに力を注いでいる、業界のメンバーから成る非営利団体である。DMTF に関する情報については、http://www.dmtf.org を参照のこと。

謝辞

DMTF は、この文書に対する以下の個人の貢献に謝意を表する。

- 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 Inc.
- Paul von Behren - Intel Corporation

1. はじめに

世界がソフトウェア・デファインド・パラダイムへと変化していく中、データセンターにおいてそのシフトに対応するためにハードウェア管理機能を進化させる必要がある。ハードウェアの構成要素が分解されて提供されるような状況になると、管理ソフトウェアはトレイ、モジュール、半導体といったハードウェアの個々の要素を 1 つに結合して、構成済み(コンポーズド)論理システムを作成する能力を必要としている。これらの論理システムは、従来からある標準的なラックマウント・システムのように機能する。これにより、ユーザーは動的にハードウェアを構成して、ワークロードのニーズを満たすことができる。さらに、ユーザーは物理的に機器を移動することなく、論理システムにより多くのコンピューティング能力を追加するなどにより、システムのライフサイクル管理を行うことができる。

Redfish は、柔軟性、拡張可能性、および相互運用性を提供するよう設計された進化するハードウェア管理標準である。Redfish には、構成可能(コンポーザブル)なハードウェアを表現するために使用するデータ・モデルと、クライアントが構成されたコンポジションを管理するためのインターフェースが含まれる。本書は、実装者およびクライアントが Redfish コンポーザビリティ・データ・モデルについて、ならびにコンポジション(構成)要求を作成する方法について理解できるよう支援するものである。

2. コンポーザビリティのためのモデリング

Redfish サービスがコンポーザビリティをサポートする場合、サービス・ルート・リソースには `CompositionService` プロパティが含まれる。Composition Service 内で、クライアントは、新しい論理システムに構成され得るすべてのコンポーネントの目録(リソース・ブロック)、異なるコンポーネントの構成におけるバインディング制限を含む記述子(リソース・ゾーン)、およびクライアントにコンポジション要求を作成する方法を通知する注釈(コレクション機能)を探し出す。以下のセクションでは、Redfish サービスによりこれらがどのように報告されるかについて詳細を示す。

2.1. コンポジション・サービス

コンポジション・サービスは、コンポーザビリティに関連するものすべてのトップ・レベルのリソースである。これには、`Status` や `ServiceEnabled` といったステータスと制御のインジケータ・プロパティが含まれる。これらはさまざまな Redfish サービス・インスタンス上にある共通プロパティである。

コンポジション・サービスには、`AllowOverprovisioning` プロパティが含まれる。これは、クライアントで要求されたよりも多くのリソースを許可できるようにするための制約ありコンポジション要求を、サービスが受け入れるか否かを示すために使用する。

コンポジション・サービスには、`AllowZoneAffinity` プロパティが含まれる。これは、クライアントで特定のリソース・ゾーンからのリソース・ブロックを使用して特定のコンポジション要求を満たせるようにするための制約のあるコンポジション要求を、サービスが受け入れるか否かを示すために使用する。

コンポジション・サービスには、それぞれ `ResourceBlocks` プロパティと `ResourceZones` プロパティを介した、リソース・ブロックおよびリソース・ゾーンのコレクションへのリンクも含まれる。リソース・ブロックについては「リソース・ブロック」セクション、リソース・ゾーンについては「リソース・ゾーン」セクションで説明する。

コンポジション・サービス・リソースの例:

```
{
   "@odata.context":   "/redfish/v1/$metadata#CompositionService.CompositionService",
   "@odata.type":  "#CompositionService.v1_1_0.CompositionService",
   "@odata.id":  "/redfish/v1/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"
   }
}
```

2.2. リソース・ブロック

リソース・ブロックはコンポジション要求の最も最下位の構成ブロックである。リソース・ブロックには、リソース・ブロック・インスタンスについてのステータスと制御の情報が含まれる。また、リソース・ブロック・インスタンス内にあるコンポーネントのリストも含まれる。たとえば、あるリソース・ブロックに 1 つのプロセッサーと 4 つの DIMM が含まれる場合、そのうち 1 つのみが必要な場合であっても、これらのコンポーネントすべては同じコンポジション要求の一部となる。構成要素が完全に分離されたシステムでは、クライアントは各リソース・ブロックに 1 つのコンポーネント・インスタンスを見つける可能性が高い。リソース・ブロックおよびそのコンポーネントは、コンポジションに属するようになるまでは、システムのソフトウェアが使用できる状態ではない。たとえば、あるリソース・ブロックにドライブ・インスタンスが含まれる場合、そのリソース・ブロックを使用できるようにするコンポジション要求が作成されるまで、ドライブはどのコンピューター・システムにも属さないことになる。

プロパティ `ResourceBlockType` には、クライアントによるリソース・ブロックの迅速な識別を支援するために使用できる、リソース・ブロック上のコンポーネント・タイプに関する分類情報が格納されている。各 `ResourceBlockType` は、そのリソース・ブロック内に含まれることになる固有のスキーマ要素と関連付けられている。たとえば、値 `Storage` がこのプロパティ内に見つかった場合、クライアントは、個々のコンポーネント・リソースを詳しく調べなくても、この特定のリソース・ブロックに、ストレージ関連のデバイス(ストレージ・コントローラーやドライブなど)が含まれることがわかる。値 `Compute` には特別の意味がある。これは、コンピューティング・サブシステムとして連携して機能するバウンド・プロセッサーとメモリー・コンポーネントを持つリソース・ブロックを表すために使用する。値 `Expansion` も、特定のリソース・ブロックが時間の経過と共に異なるタイプのデバイスを持つ場合があることを示す、特殊なインジケータである。たとえば、リソース・ブロックにプラグイン・カードが含まれ、ユーザーがいつでもコンポーネントを置き換える可能性があるような場合である。

プロパティ `CompositionStatus` は、以下のいくつかのプロパティを含むオブジェクトである。

- `CompositionState` は、コンポジションでの使用に関するリソース・ブロックの状態をクライアントに通知するために使用する。
- `Reserved` は、クライアントが使用できる書き込み可能なフラグ。このリソース・ブロックがあるクライアントにより識別されており、当該クライアントがコンポジションのために使用する予定であることを伝える。あるコンポジションのためのリソースを識別しようとしている 2 番目のクライアントが、`Reserved` フラグが true に設定されているのを見ると、2 番目のクライアントはそれを割り当て済みと見なし、使用しないようになる。2 番目のクライアントはさらに処理を行うため次のリソース・ブロックへ移動することになる。Redfish サービスは `Reserved` フラグについていかなる種類の保護も提供しない。どのクライアントでもその状態を変更でき、正しく処理する責任はクライアントに委ねられる。
- `SharingCapable` は、リソース・ブロックが複数のコンポジションに同時に参加できることを示すフラグである。
- `SharingEnabled` は、リソース・ブロックが複数のコンポジションに同時に参加することを許可されているかどうかを示す書き込み可能フラグである。
- `MaxCompositions` は、リソース・ブロックが同時に参加できるコンポジションの最大数を示すために使用する。
- `NumberOfCompositions` は、リソース・ブロックが現在参加しているコンポジション数を示すために使用する。

`Processors`、`Memory`、および `Storage` 配列などのさまざまなコンポーネント・タイプへのいくつかのリンク配列が存在する。これらのリンクは、最終的にはリソース・ブロック内の個々のコンポーネントにアクセスする。これらのコンポーネントは、コンポジション要求が作成された後に新しいコンポジションで利用可能になる。`ComputerSystems` 配列は、リソース・ブロックに 1 つ以上のコンピューター・システム全体が含まれる場合に使用する。これにより、一連のより小さいコンピューター・システムから、単一の構成済みコンピューター・システムを作成する能力がクライアントに付与される。

Links プロパティには、関連するリソースへの参照が含まれる。Chassis 配列には、リソース・ブロック内のリソースを含むシャーシ・インスタンスが含まれる。ComputerSystems 配列には、コンポジションの一部としてリソース・ブロックを消費するコンピューター・システム・インスタンスが含まれる。Zones 配列には、リソース・ブロックを含むリソース・ゾーンへのリンクが含まれる。

リソース・ブロックのリソースの例:

```
{
   "@odata.context":   "/redfish/v1/$metadata#ResourceBlock.ResourceBlock",
   "@odata.type":  "#ResourceBlock.v1_2_0.ResourceBlock",
   "@odata.id":   "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3",
   "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": 1
   },
   "Processors": [],
   "Memory": [],
   "Storage": [
      {
         "@odata.id":  "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock3/Storage/Block3NVMe"
      }
   ],
"  Links": {
      "ComputerSystems": [
         {
            "@odata.id":  "/redfish/v1/Systems/ComposedSystem"
         }
      ],
      "Chassis": [
         {
            "@odata.id":  "/redfish/v1/Chassis/ComposableModule3"
         }
      ],
      "Zones": [
         {
            "@odata.id":  "/redfish/v1/CompositionService/ResourceZones/1"
         },
         {
            "@odata.id":  "/redfish/v1/CompositionService/ResourceZones/2"
         }
      ]
   }
}
```

上記の例では、リソース・ブロックは Storage タイプで、単一のストレージ・エンティティが含まれる。CompositionStatus から、リソース・ブロックが現在少なくとも 1 つのコンポジションの一部であり、より多くのコンポジションで使用可能であることがわかり、Links セクションで、それがコンピューター・システム ComposedSystem によって使用されていることがわかる。

2.2.1. CompositionState のための推奨される状態の図

クライアントが構成済みのリソースを作成または削除する要求を行うと、リソース・ブロックは `CompositionStatus` オブジェクト内の `CompositionState` プロパティによって示されるように、異なる状態を遷移する。図 1 は、共有可能でない 1 つのリソース・ブロックが関係する `CompositionState` の推奨される状態の図を示している。図 2 は、共有可能である 1 つのリソース・ブロックが関係する `CompositionState` の推奨される状態の図を示している。図に示されていないが、クライアント要求は、何かの電源が入っていないなど事前条件チェックで失敗する可能性があり、状態が変更されない場合もある。

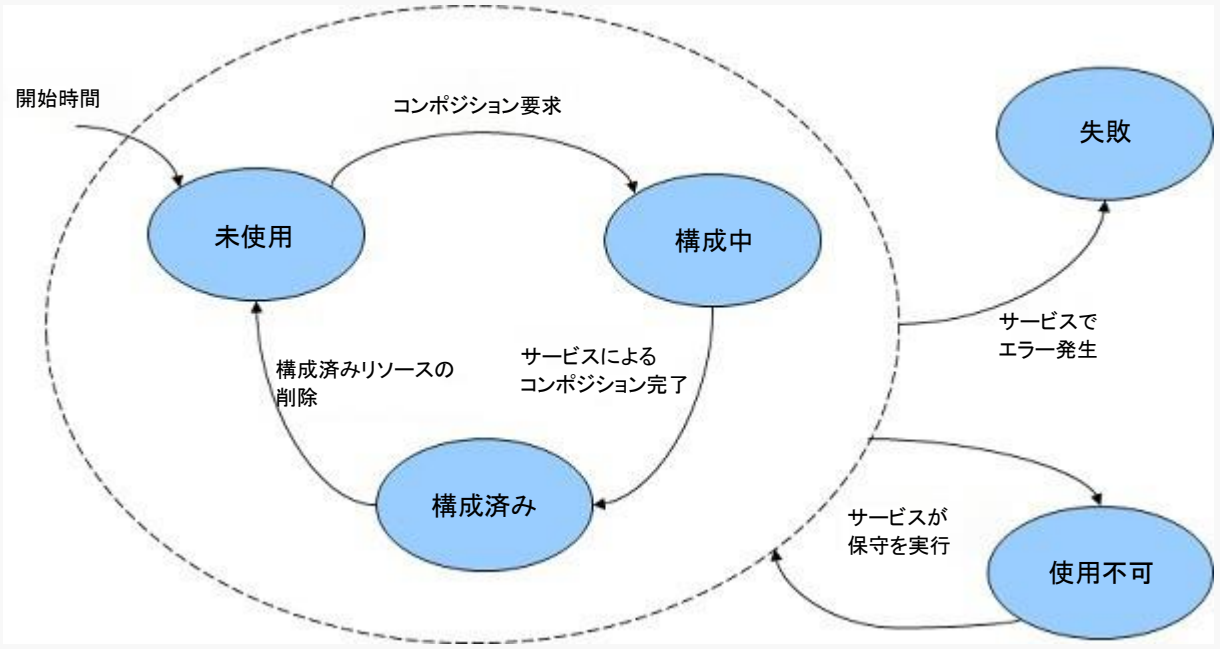

図 1

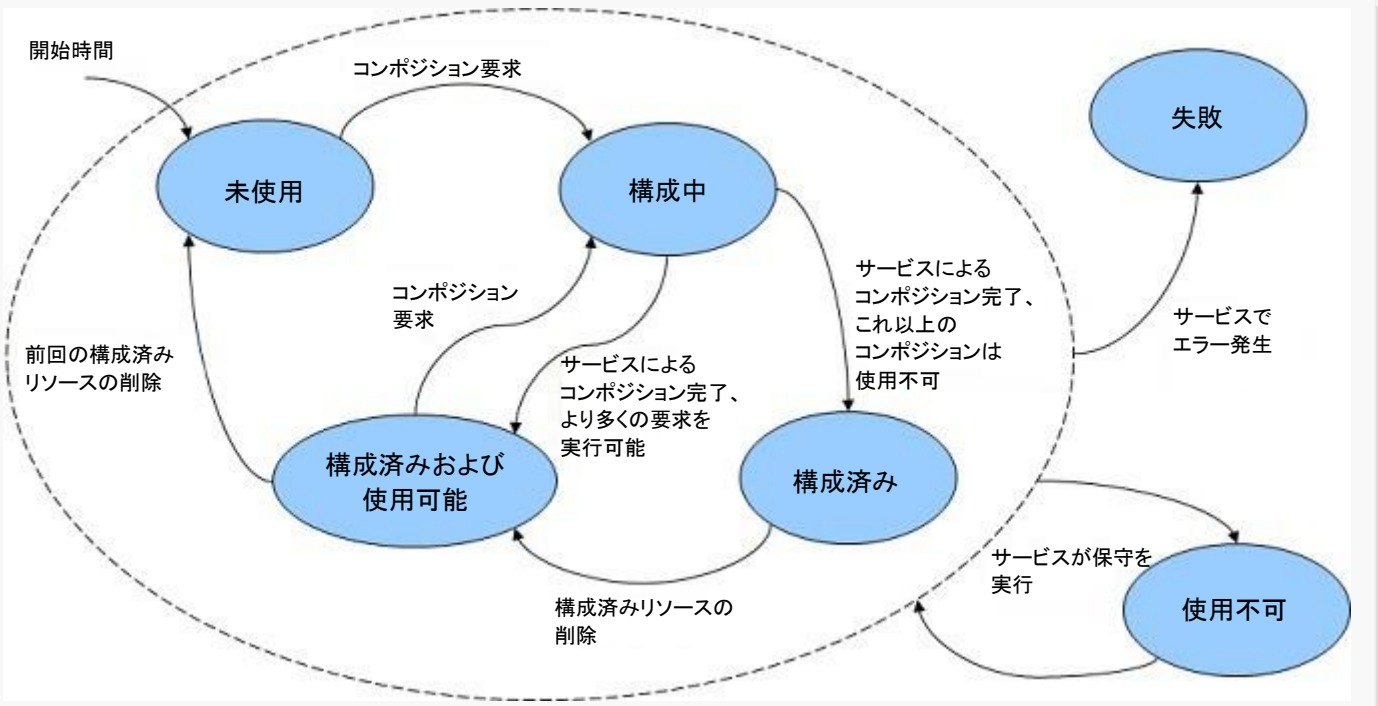

図 2

2.3. リソース・ゾーン

リソース・ゾーンは、サービスにより報告されるリソース・ブロックの異なるコンポジションの制限をクライアントに対して記述する。同じリソース・ゾーンで報告されたリソース・ブロックは、一緒に構成することが許可される。これにより、特定の実装に対して設定された異なる制限を認識するためにクライアントがトライ&エラーのロジックを実行する必要がなくなる。また、リソース・ゾーンはコレクション機能の注釈を利用して、各リソース・ゾーンが構成できる対象を記述する。この点は、「コレクション機能」セクションでより詳細に説明する。

リソース・ゾーンのリソースの例:


```
{
   "@odata.context":  "/redfish/v1/$metadata#Zone.Zone",
   "@odata.type": "#Zone.v1_1_0.Zone",
   "@odata.id":  "/redfish/v1/CompositionService/ResourceZones/1",
   "Id": "1",
   "Name": "Resource Zone 1",
   "Status": {
      "State": "Enabled",
      "Health": "OK"
   },
   "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"
         }
      ]
   },
   "@Redfish.CollectionCapabilities":  {
      "@odata.type":  "#CollectionCapabilities.v1_1_0.CollectionCapabilities",
      "Capabilities": [
         {
            "CapabilitiesObject": {
               "@odata.id":  "/redfish/v1/Systems/Capabilities"
            },
            "UseCase":  "ComputerSystemComposition",
            "Links": {
               "TargetCollection": {
                  "@odata.id": "/redfish/v1/Systems"
               }
            }
         }
      ]
   }
}
```


上記の例では、リソース・ブロック `ComputeBlock1`、`DriveBlock3`、`DriveBlock4`、`DriveBlock5`、`DriveBlock6`、および `DriveBlock7` はすべて同じリソース・ゾーン内に存在する。また、リソース・ゾーンのコレクション機能は、このリソース・ゾーンがコレクション `/redfish/v1/Systems` のコンピューター・システムを作成できることを示している。

2.4. コレクション機能

コレクション機能は、リソース・ゾーンおよびリソース・コレクション自体に存在する。これは、コレクション機能が、コンポーザビリティのコンテキスト外のことに適用可能であるためである。コレクション機能は、応答する側の`@Redfish.CollectionCapabilities` の注釈によって識別できる。この注釈は、指定されたユース・ケースに基づいて特定のコレクションに作成（POST）操作の要求内容を作成する方法をクライアントに通知するために使用する。この結果新しいメンバーが当該コレクションに追加される。

2.4.1. コレクション機能の注釈

コレクション機能の注釈内に、`Capabilities` という単一のプロパティが存在する。これは、特定のリソース・ゾーンまたはリソース・コレクションのすべての機能を識別するための配列である。`Capabilities` 配列の各インスタンス内に、特定の機能を説明するオブジェクトが存在する。

プロパティ `CapabilitiesObject` には、ペイロード形式を記述する基礎となるオブジェクト・インスタンスに対する URI が含まれる。この点については、次のセクションで詳細に説明する。

プロパティ `UseCase` は、特定の作成（POST）操作のコンテキストについてクライアントに通知するために使用する。下の表は、コンポーザビリティによって使用される `UseCase` の異なる値を示している。各値は、要求のコンポジション・タイプに加えて、構成されるリソースの特定の種類に対応している。

UseCase の値	構成されるリソース	コンポジション・タイプ
`ComputerSystemComposition`	`ComputerSystem`	指定
`ComputerSystemConstrainedComposition`	`ComputerSystem`	制約あり

`Links` オブジェクト内のプロパティ `TargetCollection` には、所与の機能を受け入れるリソース・コレクションの URI が含まれる。クライアントは、`CapabilitiesObject` のコンテンツによって記述されるように、この URI に対して作成（POST）操作を実行できる。

コレクション機能の注釈の例：

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

上記の注釈には、2 つの機能が含まれる。最初の機能オブジェクトでは、`UseCase` プロパティは、この機能が一連の固有のリソース・ブロックから 1 つの新しいコンピューター・システムを作成するための作成（POST）要求を生成する方法を記述することを示している。さらに、`TargetCollection` プロパティは、クライアントがリソース・コレクション`/redfish/v1/Systems` に対して要求を行えることを示している。ク

ライアントによって作成される新しいリソース・インスタンスは、そのコレクション内に存在するようになる。2 番目の機能オブジェクトでは、UseCase プロパティは、この機能が一連の制約から 1 つの新しいコンピューター・システムを作成するための作成（POST）要求を生成する方法を記述することを示している。

2.4.2. コレクション機能オブジェクト

コレクション機能オブジェクトは、クライアントが作成できる新しいリソースのスキーマに従う。たとえば、オブジェクトが新しいコンピューター・システム・インスタンスを作成する要求を形成する方法を説明している場合、オブジェクトのタイプは ComputerSystem.vX_Y_Z.ComputerSystem となる。ここで、vX_Y_Z はサービスによってサポートされる ComputerSystem のバージョンである。

オブジェクト自体に、クライアントが作成（POST）操作の本体で使用できる注釈付きのプロパティが含まれる。また、それにはオプションのプロパティおよび、新しいリソースの作成後にプロパティに追加される可能性のある制限がリストされる。下の表は、コレクション機能オブジェクト内のプロパティで使用される異なる注釈を示している。.

プロパティの注釈	説明
@Redfish.RequiredOnCreate	クライアントは、作成（POST）要求の本体に特定のプロパティを提供する必要がある
@Redfish.OptionalOnCreate	クライアントは、作成（POST）要求の本体にプロパティを提供できる
@Redfish.SetOnlyOnCreate	クライアントでプロパティに特定の値が必要である場合、作成（POST）要求の本体で提供される必要がある。このプロパティはリソースの作成後の「読み取り専用」プロパティとなる可能性が高い
@Redfish.UpdatableAfterCreate	クライアントは、リソースの作成後にプロパティの更新を許可されている
@Redfish.AllowableValues	クライアントは、特定のプロパティに対する作成（POST）要求の本体で指定された値のいずれかを使用することを許可されている

上の表の一部の注釈の用語は、誤って使用すると互いに競合する可能性がある。これは、用語の定義による競合する論理セマンティクスが原因である場合がある。サービスで、コレクション機能オブジェクトに以下のようなタイプの競合がないことを確認する必要がある。

- @Redfish.RequiredOnCreate および@Redfish.OptionalOnCreate の両方でプロパティに注釈を付記しないこと。1 つのプロパティを、必須とオプションの両方に指定することはできない。
- @Redfish.SetOnlyOnCreate および@Redfish.UpdatableAfterCreate の両方でプロパティに注釈を付記しないこと。1 つのプロパティは、これらのうちどちらかにのみ指定可能である。

オブジェクトには、クライアントに対する別のタイプのペイロード・ルールを記述するオブジェクト・レベルの注釈も含めることができる。下の表は、コレクション機能オブジェクト内のオブジェクト・レベルで使用される異なる注釈を示している。

オブジェクトの注釈	説明
@Redfish.RequestedCountRequired	クライアントが@Redfish.RequestedCount を使用して要求ペイロード内の対応するオブジェクトに注釈を追加し、クライアントが要求しているオブジェクト・インスタンス数を示す必要があることを示している

コレクション機能オブジェクトの例:

```
{
   "@odata.context":  "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
   "@odata.type":  "#ComputerSystem.v1_4_0.ComputerSystem",
   "@odata.id":  "/redfish/v1/Systems/Capabilities",
   "Id": "Capabilities",
   "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"
      ]
   },
   "Links@Redfish.RequiredOnCreate":  true,
   "Links": {
      "ResourceBlocks@Redfish.RequiredOnCreate":  true,
      "ResourceBlocks@Redfish.UpdatableAfterCreate":  true
   }
}
```

上記の例では、`Name`、`Links`、および `Links` 内の `ResourceBlocks` が `Redfish.RequiredOnCreate` の注釈でマークされている。その他すべてのプロパティは `Redfish.OptionalOnCreate` の注釈が付記されている。しかし、`Name` および `Description` の両方が `Redfish.SetOnlyOnCreate` で注釈が付記されており、それらは新しいリソースの作成後に変更できないことを意味している。

3. コンポジション・タイプ

Redfish コンポーザビリティ・データ・モデルは、サービス実装者がニーズに基づいて異なるコンポジション・タイプを報告する上で柔軟性を提供する。サービスは、コレクション機能の注釈にある `UseCase` プロパティに基づいて、コンポジション要求のタイプをクライアントに通知する。現在の Redfish コンポーザビリティ・モデルでは、指定コンポジション(Specific Composition)というタイプを定義している。

3.1. 指定コンポジション

指定コンポジションにより、クライアントは事前定義されたリソース・ブロックおよびリソース・ゾーンを介して構成されたリソースを作成し、そのライフサイクルを管理できる。リソース・ブロックはリソース・ゾーン内の自己完結型エンティティであるため、クライアントはコンポジション要求で指定されたのリソース・ブロックを選択することができる。

バインディング規則に従ってリソース・ブロックを選択し、作成（POST）要求内で指定リソース・ブロックの詳細を提供する例については、「構成済みリソースの作成」セクションに示している。

指定コンポジションの例として適合する別の業界標準サーバー設計は、ブレード・パーティションのモックアップで定義されている。この例では、分離されたハードウェア・シャーシで構成される複数ブレードのエンクロージャは、共にバインドして、パーティション・サーバーと呼ばれるものを作成できる。これらのパーティションは、指定コンポジションを使用して構成できる。Redfish サービスはエンクロージャー内で各ブレードをリソース・ブロックとして実装し、`ResourceBlockType` を `Compute` と `Storage` のどちらかに設定して、クライアントが複数のリソース・ブロックを組み合わせて、構成済みのコンピューター・システム、つまりパーティション・サーバーを作成できるようにする。

指定コンポジションの作成 (POST) 要求の本体の例:

```
{
   "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. 制約ありコンポジション

制約ありコンポジションでは、クライアントはコンポジションに組み込むコンポーネントの数と特性を指定することでコンポジションを要求できる。リソース・ブロックの選択は、クライアントからコンポジション・サービスに委任される。制約ありコンポジションでは、クライアントはリソース・ゾーンを理解する必要はない。

4. 付録

4.1. コンポジション要求を行うクライアントのワークフロー

クライアントがコンポジション要求を行うには、指定コンポジションのワークフローと、制約ありコンポジションのワークフローの 2 つのワークフローが存在する。

以下に、クライアントが Redfish コンポジション・モデルを使用して構成済みシステムを作成および管理する際に、使用することが予想される操作を示す。以下の例では、クライアントが有効な Redfish Session または Basic Authentication ヘッダーを持つことを前提とする。

4.1.1. Redfish サービスがコンポジションをサポートするかどうかの識別

アプリケーション・コードは常にルート(`/redfish/v1/`)から開始する。

サービス・ルート・リソースの読み込み

1. `CompositionService` プロパティを探し出す
2. そのプロパティにより指定される URI で GET を実行する
3. true となる `ServiceEnabled` 属性の値を確認する

一般的なフローの図:

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

4.1.2. 指定コンポジション・ワークフロー

クライアントは、リソース・ブロックおよびリソース・ゾーン・コレクションを読み取ることで、コンポジション・サービスにより報告されるコンポジション・モデルを理解する必要がある。この関係は、Redfish サービスでサポートする、報告された `UseCase` を実行するために使用する。これについては、後半の「構成済みリソースの作成」セクションで説明する。

4.1.2.1 リソース・ブロックの読み込み

1. コンポジション・サービス URI で GET を実行する
2. `ResourceBlocks` プロパティを探し出す
3. その URI 上で GET を実行して、すべてのリソース・ブロックのリストを取得する
4. 特定のリソース・ブロックの詳細にアクセスするには、`Members` 配列内の特定のエントリに対してリストされた関連付けられた URI 上で GET を実行する
5. 各リソース・ブロック内の `CompositionStatus` プロパティは、コンポジション要求内のリソース・ブロックが利用できるかどうかを識別する
    - クライアントはコンポジション要求で使用するリソース・ブロックを決定する際にこの点に留意する必要がある
    - `CompositionStatus` プロパティに含まれるものによっては、特定のリソース・ブロックはコンポジションに現時点で使用できない可能性がある

リソース・ブロック・コレクションのサンプル:


```
[
   "@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" }
   ]
}
```

4.1.2.2. リソース・ゾーンの読み込み

1. コンポジション・サービス URI で GET を実行する
2. `ResourceZones` プロパティを探し出す
3. その URI 上で GET を実行して、すべてのリソース・ゾーンのリストを取得する
4. 特定のリソース・ゾーンの詳細にアクセスするには、`Members` 配列内の特定のエントリに対してリストされた関連付けられた URI 上で GET を実行する

リソース・ゾーン・コレクションのサンプル:

```
[
   "@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" }
   ]
}
```

4.1.2.3 各リソース・ゾーンの機能の読み込み

1. `Members` 配列の各エントリに存在する URI を使用して、各リソース・ゾーンで GET を実行する
2. 各リソース・ゾーンで`@Redfish.CollectionCapabilities` の注釈を探し出す
    - `UseCase` プロパティは、後でクライアントが作成するコンポジション・タイプを決定した時に使用する
    - `TargetCollection` プロパティは、後でコンポジション要求を行う際に使用する

リソース・ゾーン機能のサンプル:

```
[
   "@odata.context":  "/redfish/v1/$metadata#Zone.Zone",
   "@odata.type": "#Zone.v1_1_0.Zone",
   "@odata.id":  "/redfish/v1/CompositionService/ResourceZones/1",
   "Id": "1",
   "Name": "Resource Zone 1",
   "Status": {},
   "Links": {},
"@Redfish.CollectionCapabilities":  {
      "@odata.type":  "#CollectionCapabilities.v1_1_0.CollectionCapabilities",
      "Capabilities": [
         {
            "CapabilitiesObject": { "@odata.id": "/redfish/v1/Systems/Capabilities" },
            "UseCase":"ComputerSystemComposition",
            "Links": {
               "TargetCollection": { "@odata.id": "/redfish/v1/Systems" },
               "RelatedItem": [
                  { "@odata.id": "/redfish/v1/CompositionService/ResourceZones/1" }
               ]
            }
         }
      ]
   }
}
```

4.1.2.4　各機能のオブジェクトの読み込み

1. 各機能の `CapabilitiesObject` プロパティにリストされた URI で GET を実行する

指定コンポジションの機能オブジェクトのサンプル


```
[
   "@odata.context":  "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
   "@odata.type":  "#ComputerSystem.v1_4_0.ComputerSystem",
   "@odata.id":  "/redfish/v1/Systems/Capabilities",
   "Id": "Capabilities",
   "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"
      ]
   },
   "Links@Redfish.RequiredOnCreate":  true,
   "Links": {
      "ResourceBlocks@Redfish.RequiredOnCreate":  true,
      "ResourceBlocks@Redfish.UpdatableAfterCreate":  true
   }
}
```


4.1.2.5.　構成済みリソースの作成

クライアントは次の手順で指定コンポジション要求を構築する。

1. 上記の例で説明するように、コレクション URI で GET を実行して、特定のリソース・ゾーンに属するすべてのリソース・ブロックをリストする
    - リソース・ブロックを読み込む際、`CompositionStatus` プロパティに留意する
    - `CompositionStatus` プロパティに含まれるものによっては、特定のリソース・ブロックはコンポジションに現時点で使用できない可能性がある
2. （オプション）コンポジション要求に識別された各リソース・ブロックを予約する
    - `Reserved` が true に設定された各リソース・ブロックで、PATCH を実行する
    - この手順は、複数のクライアントがコンポジション要求を行う可能性があるシナリオで実行する必要がある
3. 指定コンポジションの `UseCase` のニーズを特定する
    - 目的のリソース・ゾーンで GET を実行する
    - `@Redfish.CollectionCapabilities` 注釈内で一致する `UseCase` の値を探し出す
        - たとえば、リソース・ブロックの指定のリストから新しいコンピューター・システムを構成しようとしている場合は、`ComputerSystemComposition` の値を探し出す
    - プロパティ `CapabilitiesObject` 内で見つかる URI で GET を実行する
    - `RequiredOnCreate` の注釈の付いたプロパティすべてをマークダウンする
        - これらは、コンポジション要求の一部として渡される必要のあるプロパティである
    - `TargetCollection` URI をマークダウンする
        - これは、新しいコンポジションの作成（POST）要求が行われる場所である
4. `RequiredOnCreate` の注釈の付いたプロパティすべてを使用して、`TargetCollection` URI に送信される作成（POST）要求の本体を構築する
    - 上記の例の手順 4 では、`Links` にある `Name` および `ResourceBlocks` のみが必須である
    - Redfish サービスは、要求の一部としてその他のプロパティを受けれることができるため、後でそれらを更新する必要はない
5. サービス応答内の `Location` HTTP ヘッダーには、構成されたリソースの URI が格納されている

一般的なフローの図:

```
Client   |                                                                  | Redfish Service
         |---> GET /redfish/v1/CompositionService/ResourceZones/1 ----------> |
         |<--- { ..., "UseCase": "ComputerSystemComposition", ... } <---------- |
         |                                                                  |
         |---> GET /redfish/v1/Systems/Capabilities -------------------------> |
         |     {  ...,                                          <---------- |
         |        "Name@Redfish.RequiredOnCreate":  true,                     |
         |        "ResourceBlocks@Redfish.RequiredOnCreate":  true,           |
         |        ...                                                         |
         |<--- }                                                              |
         |                                                                    |
         |           ( << Identify which Resource Blocks to use >> )          |
         |                                                                    |
         |-> GET /redfish/v1/CompositionService/ResourceBlocks/ComputeBlock2 -> |
         |<--- { ..., "CompositionState": "Unused", "Reserved": false ... } <-- |
         |                                                                    |
         |-> PATCH /redfish/v1/CompositionService/ResourceBlocks/ComputeBlock2  |
         |  { "CompositionStatus": { "Reserved": true } } -------------------->  |
```

クライアント要求の例:

```
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" }
      ]
   }
}
```

サービス応答の例:

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

上記のクライアント要求の例には、`/redfish/v1/Systems` にあるコンピューター・システム・コレクションに対してクライアントが実行する指定コンポジション要求を示している。この要求で、クライアントはリソース・ブロック `ComputeBlock0` および `DriveBlock2` を使用して、新しいコンピューター・システムを作成する。上記のサービス応答の例では、サービスは成功を示す 201 応答で応答し、新しいコンピューター・システムが `/redfish/v1/Systems/NewSystem` に存在することを示している。

4.1.3. 制約ありコンポジション・ワークフロー

以下に、クライアントが Redfish コンポジション・モデルを使用して構成済みシステムを作成および管理する際に、使用することが予想される操作を示す。以下の例では、クライアントが有効な Redfish Session または Basic Authentication ヘッダーを持つことを前提とする。

4.1.3.1. 機能のオブジェクトの読み込み

`UseCase` プロパティに値 `ComputerSystemConstrainedComposition` が含まれる `CapabilitiesObject` プロパティにリストされた URI で、GET を実行する。

制約ありコンポジションの機能オブジェクトのサンプル:


```
{
   "@odata.context":  "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
   "@odata.type":  "#ComputerSystem.v1_4_0.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
                     }
                  }
               ]
            }
         }
      ]
   }
}
```

上記の例では、制約ありコンポジションの機能オブジェクトの一般的な構造は、クライアントが構成できるリソースを表す拡張されたオブジェクトであることに注意を払う必要がある。このケースでは、プロパティ `Processors` および `Memory` が拡張されており、それぞれプロセッサー・コレクションとメモリー・コレクションの表現を示している。これらの拡張されたオブジェクトには、`@odata.id` といった、通常の環境で必須とされる無関係の情報は含まれていない。これは、コンポジション要求を形成するためにクライアントが必要とする情報のみを含むよう、情報を減らすためである。

4.1.3.2. コンポジション要求の作成

制約ありコンポジション要求では、要求にはプロセッサー、メモリー、ストレージ、およびネットワーク・インターフェースの構造が含まれる。各構造には、`@Redfish.RequestedCount` という注釈が含まれる。この注釈はリソースの要求される量を指定する。

前のセクションの機能の例に基づいて、以下の構造には列挙型注釈と、必須であるサブプロパティを含めることができる。

- プロセッサーの要求
  - `@Redfish.RequestedCount` が必須
  - `ProcessorType` および `TotalCores` プロパティが必須
- メモリーの要求
  - `@Redfish.RequestedCount` が必須
  - `MemoryType` および `CapacityMiB` プロパティが必須
- ストレージの要求
  - `@Redfish.RequestedCount` が必須
  - `SimpleStorage` または `Storage` プロパティのどちらかが存在するか、またはどちらも存在しない
  - 存在する場合、`CapacityBytes` プロパティが必須
- ネットワーク・インターフェースの要求
  - `@Redfish.RequestedCount` が必須
  - `EthernetInterfaces` または `NetworkInterfaces` プロパティのどちらかが存在するか、またはどちらも存在しない
  - 存在する場合、`SpeedMbps` または `LinkSpeedMbps` プロパティが必須

たとえば、以下のプロセッサー要求では、4 つの CPU と 2 つの FPGA を要求する。要求内のその他のプロパティは、プロセッサー・リソースの追加の属性を指定する。

コンポジション成功の可能性を高めるため、コンポジション要求はシンプルにする必要がある。過度に制約のある要求は、実現する可能性が低くなる。

一連のプロセッサーを記述する要求ペイロードのサンプル:

```
{
   "Processors": {
      "Members": [
         {
            "@Redfish.RequestedCount": 4,
            "ProcessorType": "CPU",
            "TotalCores": 16
         },
         {
            "@Redfish.RequestedCount": 2,
            "ProcessorType": "FPGA",
            "TotalCores": 16
         }
      ]
   }
}
```

4.1.3.3. コンポジション要求の実行

`TargetCollection` URI に要求を POST する。

クライアント要求の例:

```
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",
   "Description": "Description of server",
   "@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"
               }
            ]
         }
      ]
   }
}
```

サービス応答の例:

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

上記のクライアント要求の例には、`/redfish/v1/Systems` にあるコンピューター・システム・コレクションに対してクライアントが実行するコンポジション要求を示している。この要求で、クライアントは 4 つの CPU、4 つの FPGA、4GB のメモリー、6 つの 322GB のローカル・ドライブ、および 1GB の Ethernet インターフェースで構成される 1 つの新しいコンピューター・システムを要求している。注釈 `@Redfish.AllowOverprovisioning` の設定により、Redfish サービスが要求されたよりも多くのリソースを供給することが許可される。注釈`@Redfish.ZoneAffinity` の使用法を見ると、コンポジションのコンポーネントすべてを、`ID` プロパティの値が`"1"`のリソース・ゾーンから選択するよう、クライアントが指定していることを示している。

上記のサービス応答の例では、サービスは成功を示す 201 応答で応答し、新しいコンピューター・システムが `/redfish/v1/Systems/NewSystem2` に存在することを示している。

4.1.4. 構成済みリソースの更新

Redfish サービスが既存のコンポジションの更新をサポートしている場合、クライアントは PUT/PATCH を介してすでに作成したコンポジションを更新できる。これは、構成されたリソースにある ResourceBlocks 配列を更新することで実行できる。PATCH を使用する場合、Redfish 仕様で説明するように同じ配列セマンティクスが適用される。

クライアント要求の例:

```
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" }
      ]
   }
}
```

上記の例では、配列要素 0 および 1 に構成されたリソース内の既存のリソース・ブロックを保持し、配列要素 2 に `NetworkBlock8` リソース・ブロックを追加することになる。

4.1.5. 構成済みリソースの削除

クライアントは、DELETE を使用してすでに構成されたリソースを削除または分解できる。

クライアント要求の例:

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

上記の例では、`NewSystem` という名前の構成済みシステムを削除するよう要求する。削除が行われると、このシステムによって使用されていたリソース・ブロックが解放され、将来のコンポジションで使用できるようになる。ただし、各リソース・ブロックの `CompositionStatus` 内にある `Reserved` フラグは同じ状態のままになる。クライアントがそのリソース・ブロックの使用を停止した場合、クライアントは `Reserved` フラグを false に設定する必要がある。

4.2. 参照

- 「構成可能なシステム」と「ブレード・パーティション」のモックアップ: http://redfish.dmtf.org/redfish/v1
- コンポジション・サービスのスキーマ: http://redfish.dmtf.org/schemas/v1/CompositionService_v1.xml
- リソース・ブロックのスキーマ: http://redfish.dmtf.org/schemas/v1/ResourceBlock_v1.xml
- リソース・ゾーンのスキーマ: http://redfish.dmtf.org/schemas/v1/Zone_v1.xml
- コレクション機能のスキーマ: http://redfish.dmtf.org/schemas/v1/CollectionCapabilities_v1.xml

4.3. 変更のログ

バージョン	日付	説明
1.1.0	2018 年 8 月 3 日	制約ありコンポジション要求の文書を追加。
		DSP8010 2018.1 および 2018.2 で追加された新しいプロパティに対応するためモデリングのセクションを更新。
		機能オブジェクトでプロパティに注釈を付記する際に避けるべき異なる条件について実装者に対するガイダンスを追加。
		リソース・ブロック内の CompositionState プロパティに対して推奨されるフロー図を追加。
1.0.0	2017 年 6 月 30 日	初期リリース。