Hewlett Packard Enterprise

RESTful API を使用した Hewlett Packard Enterprise サーバー管理

摘要

このガイドでは、Hewlett Packard Enterprise サーバーを管理するための iLO RESTful（iLO 4）および Moonshot iLO Chassis Management Module RESTful API の使用を開始する方法を説明します。

部品番号: 795538-195
2016 年 4 月
第 1 版

本書の内容は、将来予告なしに変更されることがあります。Hewlett Packard Enterprise 製品およびサービスに対する保証については、当該製品およびサービスの保証規定書に記載されています。本書のいかなる内容も、新たな保証を追加するものではありません。本書の内容につきましては万全を期しておりますが、本書中の技術的あるいは校正上の誤り、脱落に対して、責任を負いかねますのでご了承ください。

本書で取り扱っているコンピューターソフトウェアは秘密情報であり、その保有、使用、または複製には、Hewlett Packard Enterprise から使用許諾を得る必要があります。FAR 12.211 および 12.212 に従って、商業用コンピューターソフトウェア、コンピューターソフトウェアドキュメンテーション、および商業用製品の技術データ（Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items）は、ベンダー標準の商業用使用許諾のもとで、米国政府に使用許諾が付与されます。

他社の Web サイトへのリンクは、Hewlett Packard Enterprise の Web サイトの外に移動します。Hewlett Packard Enterprise は、Hewlett Packard Enterprise の Web サイト以外にある情報を管理する権限を持たず、また責任を負いません。

商標

Microsoft®および Windows® は、米国および/またはその他の国における Microsoft Corporation の登録商標または商標です。

Google Inc.© All rights reserved. Google™ は、Google Inc.の商標です。

Linux® は、Linus Torvalds の米国およびその他の国における登録商標です。

本製品は、日本国内で使用するための仕様になっており、日本国外で使用される場合は、仕様の変更を必要とすることがあります。

本書に掲載されている製品情報には、日本国内で販売されていないものも含まれている場合があります。

目次

1 RESTful API の紹介

iLO RESTful API（iLO 4）と Moonshot iLO Chassis Management Module RESTful API は、最新のサーバー管理を有効にするプログラミングインターフェイスです。本書には、RESTful API とやり取りをする方法に関する有用な情報が記載されています。RESTful API は、基本的な HTTP 操作（`GET`、`PUT`、`POST`、`DELETE`、および`PATCH`）を使用して、JSON 形式のリソースを、iLO 4 または Moonshot iLO Chassis Management Module 上の URI との間で送信したり戻したりします。

現代のスクリプティング言語で、RESTful API 向けの単純な REST クライアントを容易に書くことができます。Python などのほとんどの言語では、辞書のような内部データ構造に JSON を変換できるため、データに簡単にアクセスできます。これにより、HPE の HPQLOCFG や CONREP などの中間ツールを使用せずに、直接 RESTful API を呼び出すカスタムコードを書くことができます。

本書は、iLO 4 バージョン 2.30 ファームウェアによる例で更新されました。

2 Redfish 1.0 準拠

RESTful API は、HPE Gen9 サーバー上の iLO 4 2.00 で最初にリリースされました。RESTful API は、新しい Redfish 1.0 DMTF 標準（**http://www.dmtf.org/standards/redfish** を参照してください）対応の開始点としても機能しました。

RESTful API の導入以降、Redfish 標準に数多くの変更が DMTF SPMF メンバーによって行われました。高レベルでは、以下の変更が行われました。

- メタデータを通信するための、データ全体での OData 注釈の使用
- エラー構造のリビジョン
- 特定の JSON プロパティの削除、または名前の変更
- 別のコレクションスキーマの使用
- 異なるセットのエントリーポイント URI（`/redfish/v1/`と`/rest/v1`）の使用

iLO 4 2.30 は既存の RESTful API と下位互換性を保ちながら、Redfish 1.0 に準拠しています。今後、RESTful API は、Redfish を iLO 4 で機能強化した実装になります。Redfish 1.0 標準に準拠しつつ、BIOS 構成などの Hewlett Packard Enterprise 固有の機能で RESTful API は拡張されます。Redfish 標準に準拠するために、ご使用のクライアントコードのアップデートを計画する必要があります。iLO は、（OEM 拡張機能を維持しつつ）Redfish スキーマ定義と一致しないすべてのプロパティを最終的に削除します。

iLO 4 2.30 は、次によって Redfish 1.0 準拠と下位互換を実現します。

1. `/redfish/v1/`と`/rest/v1` 両方でのリソースモデルのミラーリング。
2. デフォルトでは、互換性と Redfish プロパティの両方を返します。
3. Redfish に必要な OData ヘッダーがリクエスト（OData バージョン：4.0）に含まれる場合は、（Hewlett Packard Enterprise の拡張の）Redfish 準拠プロパティだけを返します。

3 HATEOAS を使用して設計された REST API

Representational State Transfer（REST）は、`POST`、`GET`、`PUT`、`PATCH`、および`DELETE` などのHTTP コマンドを使用してリソース上で実行された基本的なCRUD（生成、読み取り、更新、削除、およびパッチ）操作を使用する Web サービスです。RESTful API は HATEOS（Hypermedia as the Engine of Application State）と呼ばれる REST アーキテクチャーを使用して設計されています。このアーキテクチャにより、クライアントは、簡単な固定 URL（`rest/v1`）および iLO データモデルで記載された他のいくつかの最上位 URI を使用して、iLO と情報のやり取りをすることができます。データモデルの残りの部分は、データで明確に識別する「リンク」をたどることで検出できます。これには、クライアントが一連の固定 URL を知る必要がないというメリットがあります。RESTful API を使用してタスクを自動化するスクリプトを作成する場合、この簡単なURL をハードコードして、タスクの実行に必要なREST API URL を検出するスクリプトを設計するだけで済みます。REST および HATEOAS の概念について詳しくは、以下を参照してください。

- **http://ja.wikipedia.org/wiki/REST**
- **http://en.wikipedia.org/wiki/HATEOAS**

RESTful API の主な利点

RESTful API は、iLO 4 と Moonshot iLO Chassis Management Module ベースの Hewlett Packard Enterprise サーバー向けの主な管理インターフェイスになります。この機能セットは、既存の iLO XML API（RIBCL）および IPMI インターフェイスよりも機能が拡大します。RESTful API を使用すると、サーバーの完全なインベントリの取得、電源の制御とリセット、BIOS や iLO の構成、イベントログの取得など、多くの機能を使用できます。

RESTful API は、新しいソフトウェアインターフェースの共通パターンに移行するという、インターネットの傾向に対応したものです。さまざまな業界の多くの Web サービスでは REST API を使用しています。この API の実装が簡単で、使いやすく、以前の技術に比べて拡張性に優れているためです。今や、HPE OneView、OpenStack などの多くのサーバー管理の API は、REST API になりました。Software Defined Infrastructure 全体と同様に、ほとんどの Hewlett Packard Enterprise 管理ソフトウェア製品は、REST API に基づいて作られています。

さらに、RESTful API には、既存および計画されたすべてのサーバーアーキテクチャーで一貫性があるというメリットがあります。同じデータモデルは、従来のラックマウント型サーバー、ブレードに加えて、Moonshot のような新しいタイプのシステムでも動作します。このメリットは、データモデルがクライアントへのサービスの機能を自己記述するように設計されており、始めから柔軟性を考慮した設計になっているためです。

リソースの操作

操作	HTTP 動詞	説明
Create	`POST` リソース URI（ペイロード = リソースデータ）	新しいリソースを作成するか、カスタムアクションを呼び出します。同期`POST` は新しく作成したリソースを返します。
Read	`GET` リソース URI	要求されたリソース表現を返します。
Update	`PATCH` または`PUT` リソース URI（ペイロード = 更新データ）	既存のリソースを更新します。スキーマで`readonly = false` と記述されている`PATCH` プロパティだけを更新できます。
Delete	`DELETE` リソース URI	指定したリソースを削除します。

リターンコード

リターンコード	説明
2*xx*	操作は成功しました。
4*xx*	クライアント側のエラーが、エラーメッセージとともに返されました。
5*xx*	iLO エラーがエラーメッセージとともに返されました。

注記: エラーが発生すると、リターンコード 4*xx* または 5*xx* が示され、`ExtendedError` JSON 応答が返されます。期待するリソースは返されません。

4 RESTful API の使用に関するヒント

iLO RESTful API は、iLO Standard ライセンスで iLO 4 2.00 以降が動作している ProLiant Gen9 サーバーで使用できます。ただし、Advanced のライセンスがないと、データの一部の機能が使用できない場合があります。Moonshot iLO Chassis Management Module 向け RESTful API は、iLO シャーシマネージャー 1.30 以降が動作する Moonshot サーバーで使用できますが、ライセンスは必要ありません。

RESTful API にアクセスするには、Postman REST クライアントプラグイン拡張子または CURL（よく使用されるコマンドライン HTTP ユーティリティ）を持つ Web ブラウザーなどの HTTPS 対応のクライアントが必要です。

RESTful インターフェイスツールと Python の例

必要な要件ではありませんが、RESTful API で RESTful インターフェイスツールを使用できます。このコマンドラインツールは、RESTful API に直接アクセスするよりも優れた抽象度と利便性を提供します。詳細については、**http://www.hpe.com/info/restfulapi**（英語）を参照してください。

また、Hewlett Packard Enterprise は、RESTful API クライアントでの数多くの一般的な操作を実装した Python コードの例を公開しました。このコードは、**https://github.com/HewlettPackard/python-proliant-sdk** でダウンロードできます。場合によっては、本書での例は、Python コードの例を次のような注釈付きで参照する場合があります。

Python：Python コード例の`ex1_functionname()` を参照してください。つまり、python コード例で指定された関数名を参照することを意味します。

Python でクライアントを実装しない場合、これは操作を実行するために必要なロジックを実装する適切な擬似コードとして機能します。

REST API の操作の例

最初に、RESTful API を使用して`GET` 操作を実行しましょう。iLO HTTPS ポート上で、一般的にはポート 443 上で（ただし、以前に別のポートを使用するように構成した場合は、別のポートでもかまいません）、HTTP `GET` を実行します。HTTPS 証明書チャレンジが処理できるように、クライアントの準備をする必要があります。このインターフェイスは無防備な HTTP（ポート 80）では使用できないため、HTTPS を使用する必要があります。

`GET` 操作は、`/rest/v1`（末尾にスラッシュはつけない）のリソースに対して実行されます。

- 適切：`GET` **`https://myilo/rest/v1`**
- 不適切：`GET` **`https://myilo/rest/v1/`**

上記の CURL または Postman REST Client のようなツールでこの最初の`GET` を実行することが最適です。後で独自のスクリプトコードを使用して実行できますが、現在では、ブラウザーを使用して交換された HTTP ヘッダー情報を参照することが有効です。

注記:　このチュートリアルには、iLO RESTful API 内のリソースにアクセスするための疑似コードが含まれています。このコードに適切に従うため、Postman REST クライアントまたは CURL で操作することを強くお勧めします。データ構造がどのように疑似コードと一致しているかが容易に分かります。ここで示す原則と同じ原則が Moonshot iLO Chassis Management Module にも適用されます。

Redfish：Redfish は`/redfish/v1/`へのエントリーポイント URI を変更します。

クライアントが以下の URI にアクセスする場合、スラッシュを含める必要があり、iLO はクライアントに HTTP リダイレクトステータス（308）を返します。

- `/redfish`

- /redfish/
- /redfish/v1

今後、iLO 4 は、/redfish/v1/と等しい URI への 308 リダイレクトのあるすべての/rest/v1/スタイルの URI に対応します。

CURL の例

CURL は、RESTful API に簡単にアクセスできる多くのオペレーティングシステムで使用可能なコマンドラインユーティリティです。CURL は **http://curl.haxx.se/** から入手できます。CURL のすべての例がフラグ–insecure を使用していることに注意してください。これによって、CURL が HTTPS 証明書の検証をバイパスするようにします。実際の使用では、ユーザー提供証明書を使用するように iLO を構成する必要があり、このオプションは不要です。CURL が HTTP リダイレクト応答に従うよう強制する –L オプションを使用することにも注意してください。iLO が、さまざまな項目の URI の場所を変更する場合、その新しい場所をクライアントに通知して自動的に新しいリンクをたどるようにできます。

```
> curl https://myilo/rest/v1 -i --insecure -L
-iは、HTTP応答ヘッダーを返し、--insecureはTLS/SSL認定検証をバイパスし、-Lはリダイレクトに従います
```

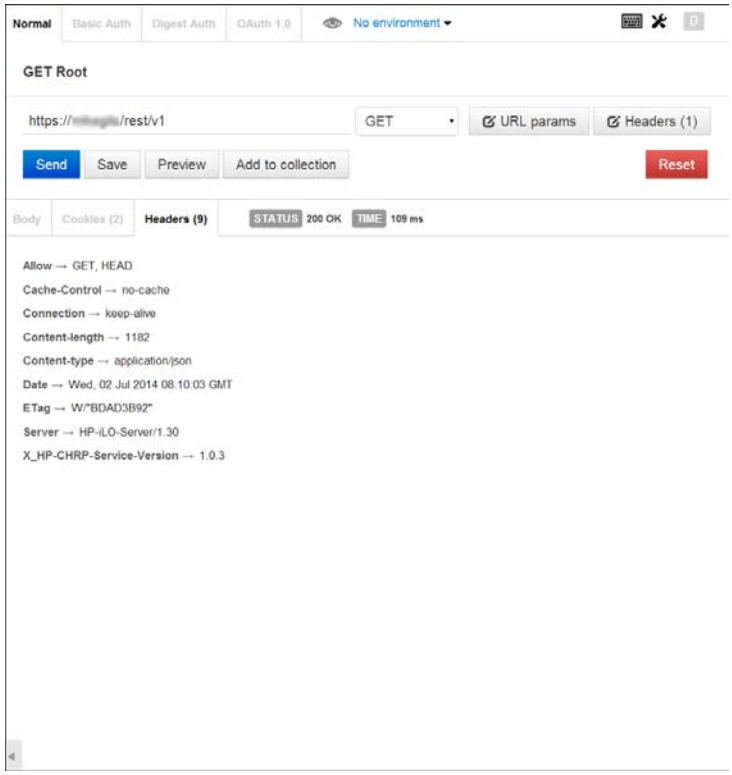
```
{"@odata.context":"/redfish/v1/$metadata#ServiceRoot","@odata.id":"/redfish/v1/","@odata.type":"#ServiceRoot.1.0.0.ServiceRoot",

 "Id":"v1","Name":"HP RESTful Root
Service","Oem":{},"ServiceVersion":"0.9.5","Time":"2014-08-05T13:12:02Z","Type":"ServiceRoot.1.0.0",
"UUID":"627490fe-bded-5d10-8176-0af0927c690e","links":{"AccountService":{"href":"/rest/v1/AccountService"},"Chassis":{"href":"/rest/v1/Chassis"},
"EventService":{"href":"/rest/v1/EventService"},"JSONSchema":{"href":"/rest/v1/Schemas"},"Managers":{"href":"/rest/v1/Managers"},
"Registries":{"href":"/rest/v1/Registries"},"Schemas":{"href":"/rest/v1/Schemas"},"SessionService":{"href":"/rest/v1/SessionService"},
"Sessions":{"href":"/rest/v1/SessionService/Sessions"},"Systems":{"href":"/rest/v1/Systems"},"self":{"href":"/rest/v1"}}}
```

HTTP 応答ヘッダーの例

GET 操作を実行するときに、応答の JSON オブジェクトを受信します。この JSON オブジェクトと上記で返された HTTP ヘッダーを見てみましょう。

最初の行は、内容を確認する応答 HTTP ヘッダーです。それに続くのが、圧縮形式での JSON 応答です。煩雑なように見えても、JSON は高度に構造化されているので、読みやすいドキュメントに「整然とした形でプリント」できます。



注目する最初の細目は、いくつかの応答ヘッダーです。ETag ヘッダーは JSON コンテンツの要約です。ETag ヘッダーの処理は高度なテーマであるため、ETag の文字列値がクライアントによって解析されることを前提としていないという点を除いて、現時点では無視します。これは曖昧な文字列であり、iLO は今後のバージョンでETag 生成アルゴリズムを変更する可能性があります。

2 番目の興味深いヘッダーは、Allow ヘッダーです。Allow は、すべてのGET 操作で RESTful API によって指定され、このリソースで許可された HTTP 操作を示します。/rest/v1 に対してGET のみが許可されていることが分かります。つまり、PATCH、PUT、またはDELETE（変更操作）を使用できないため、読み取り専用のオブジェクトであることを意味します。多くのリソースで、インターフェイスがPATCH をサポートすることがわかります。PATCH は、具体的には（PUT と同様に）既存のオブジェクトを置き換える代わりに、既存のオブジェクトを更新しようとします。これにより一部のオブジェクトにPATCH処理が可能になり、省略したすべてのプロパティを削除する心配はなくなります。

ルート JSON オブジェクトの例

上記の（長さを編集した）JSON 応答の完全なバージョンは次のとおりです。

```
{
  "@odata.context": "/redfish/v1/$metadata#ServiceRoot",
  "@odata.id": "/redfish/v1/",
  "@odata.type": "#ServiceRoot.1.0.0.ServiceRoot",
  "Id": "v1",
  "Name": "HP RESTful Root Service",
  "Oem": {},
  "ServiceVersion": "0.9.5",
  "Time": "2014-08-05T13:12:02Z",
  "Type": "ServiceRoot.1.0.0",
  "UUID": "627490fe-bded-5d10-8176-0af0927c690e",
  "links": {
    "AccountService": {
      "href": "/rest/v1/AccountService"
    },
    "Chassis": {
      "href": "/rest/v1/Chassis"
```

```
    },
    "EventService": {
      "href": "/rest/v1/EventService"
    },
    "JSONSchema": {
      "href": "/rest/v1/Schemas"
    },
    "Managers": {
      "href": "/rest/v1/Managers"
    },
    "Registries": {
      "href": "/rest/v1/Registries"
    },
    "Schemas": {
      "href": "/rest/v1/Schemas"
    },
    "SessionService": {
      "href": "/rest/v1/SessionService"
    },
    "Sessions": {
      "href": "/rest/v1/SessionService/Sessions"
    },
    "Systems": {
      "href": "/rest/v1/Systems"
    },
    "self": {
      "href": "/rest/v1"
    }
  }
}
```


JSON では、プロパティ名の順番は厳密ではないため、iLO は任意の順序で JSON プロパティを返すことがあります。同様に、iLO では送信された JSON の中のプロパティの順序を推測することはできません。これが、RESTful クライアントの最適なスクリプト記述データ構造が、順不同にキーと値を単純に組み合わせた辞書である理由です。このように順序付けされないことも、オブジェクト内に組み込み構造（オブジェクト内にオブジェクトがある）が見られる理由です。これにより、より論理的に整理した関連データをまとめて、見た目を美しくできるため、プロパティ名の競合が発生しなくなり、プロパティ名に非常に長い名前を付けなくても済みます。ステータスのような JSON の同じブロックを、データモデルの多くの場所で使用することもできます。

タイプとリソースバージョン

JSON 応答には、"`ServiceRoot.1.0.0`" の値を持つ`Type` という名前のプロパティがあることに注意してください。Type は、RESTful API で極めて重要です。これは、そのオブジェクトで他のすべてのプロパティが何を意味するかを理解する鍵です。同じタイプのリソースは、同じ**スキーマ定義**に従います（似ている名前のプロパティが同じものを意味するということです）。

Type には、名前（"`ServiceRoot`"）とバージョン情報（"`1.0.0`"）が含まれます。バージョン情報は`major.minor.errata` としてフォーマットされます。今後の iLO ファームウェアのリリースでは、さまざまなリソースに含まれるデータが確実に拡大します。このためクライアントとして、別のバージョンの`Type` を参照する準備を行う必要があります。メジャーバージョン以外は下位互換性があるので、オブジェクトに追加された新しいプロパティは、マイナー`Type` が増えることになります（たとえば、`Chassis.1.0.0` は`Chassis.1.1.0` になります）。メジャーバージョンの変更（たとえば、`Chassis.2.0.0`）への移行は、下位互換性の保証なく公表された変更です。

RESTful API の初期リリースには、タイプバージョン 0.9.5（メジャーバージョンは 0）の多くのリソースが含まれました。ただし、iLO 4 2.30 の時点で多くの種類は、バージョン 1.0.0 に移行中です。この場合、RESTful API データモデルを完成させる以外、0 から 1 へのこのタイ

プの変更は大きな変更ではないため、iLO 4 の以前のバージョンとの下位互換性を維持しました。

Redfish：Redfish では、`Type` が`@odata.type` と交換されました。いくつかの場合、タイプ名の文字列変更も変更しました（たとえば、`PowerMetrics` は`Power` になりました）。Redfish 仕様で指定された OData バージョンヘッダーを含む場合は、応答に Type プロパティがなく`@odata.type` のみであることに注意してください。このヘッダーが無い場合は、両方があります。今後、iLO が非 Redfish コンテンツを削除するときに、タイプインジケーターとして`@odata.type` を残して`Type` が削除されます。

OneView の情報

サーバーが OneView で管理されている場合、OneView のインスタンスに関する情報がこのルートリソースに含まれます。JSON ポインターパス`/rest/v1#/Oem/Hp/Manager/0/IPManager` のプロパティを参照してください。このサブオブジェクトの内容には、OneView サーバーに戻る参照が含まれます。

注記: サーバーが OneView によって管理される場合、設定の多くは OneView の使用に特化して構成されます。RESTful API のプロパティを直接変更する場合に注意をしてください。しばらくの間サーバーがシステムの OneView の表示と同期が取れなくなる可能性があるためです。

リンク

最後に、links プロパティを調べてみましょう。RESTful API では、データのナビゲーションは、仕様にはなく、データ自体に含まれています。このため、長い間仕様の変更を行わなくても異なるサーバーに必要に応じて適合することができます。これは、ナビゲーションがデータ（Web ページなど）に組み込まれているハイパーメディア API と呼ばれます。リンクのプロパティには、他の関連リソースへのポインターが含まれています。リンクオブジェクトの内側は、Systems、Managers、Chassis、Sessions など、定義されたさまざまな関係タイプです。それぞれには、関連のリソースの URI である`href` と呼ばれるプロパティが含まれます。関係のタイプ（たとえば、`Systems` の場合）は、それらの移動先によって、さまざまなリンクを識別します。たとえば、Systems は、`ComputerSystem` リソースのコレクションに移動します。

データモデルのナビゲーション

RESTful API はサーバー上の温度または電源装置など、さまざまな項目への URI のすべてを定義しません。BIOS のバージョン情報が常に特定の URI であるものと仮定することはできません。これはクライアントにとってさらに複雑なことですが、仕様を変更することなく、データモデルが今後のさまざまなサーバーアーキテクチャーに対応するために変更できるようにすることが必要です。たとえば、BIOS のバージョンが`/rest/v1/Systems/1` にあり、クライアントが、バージョンは常にそこにあると想定している場合、インターフェイスが、それぞれ独自の BIOS バージョンで、180 の演算ノードのある Moonshot 上で実装されるときに、クライアントは動かなくなります。仕様で特定機能へのポインターを定義すると、RESTful API の柔軟性がなくなります。このため、厳選した少数の URI だけが公開され、安定した開始点であることが保証されます。クライアントのコードには、データモデルに表示されている URI について何も想定する必要がありません。不明確な文字列としてそれらを処理する必要があります。そうでなければ、クライアントは RESTful API の他の実装と相互運用性がありません。

ご使用のクライアントで URI 解析テンプレートをハードコードしないでください。その代わりに/rest/v1 または下記で固定されたエントリーポイント URI の 1 つから始めて、定義された関係タイプを使用してデータモデルを移動し、興味のあるタイプのインスタンスを見つけて下さい。

Redfish：Redfish は`@odata.id` プロパティを使用して、他のリソースへのリンクを示します。（関連するリソースの）`"Links"` のセクション、または（子のリソースの）ルートオブジェクト自体に、これらのリンクがある場合があります。Redfish は、RESTful API の互換性のため`"links"` に対して`"Links"`（大文字の L）を使用する

ことに注意してください。iLO は、リクエストで OData バージョン HTTP ヘッダーの有無に基づいて`Links` または`links` のいずれかを使用します。

データモデルでの固定のエントリーポイント URI

以下の URI は、データモデルで固定されており、クライアントソフトウェアは RESTful API にアクセスするときに以下のいずれかを使用して起動する可能性があります。

- `/rest/v1`
  ルートリソースです。
- `/rest/v1/Systems`
  演算ノードのコレクションです。
- `/rest/v1/Chassis`
  シャーシのコレクションです。
- `/rest/v1/Managers`
  管理プロセッサー（iLO）のコレクションです。
- `/rest/v1/Sessions`
  セッション管理の API です。
- `/rest/v1/Registries`
  レジストリのコレクションです。
- `/rest/v1/Schemas`
  スキーマのコレクションです。

リンクのいくつかを調べてみましょう。

システム

これは、システムノードのコレクションへのリンクです。システムは演算ノード（原典は DMTF の専門用語）であり、ノードは CPU、メモリ、拡張スロット、電力管理、BIOS バージョンなどを搭載しています。DL または BL サーバーでは、システムコレクションに演算ノードは 1 つですが、Moonshot の場合、集合内に最大で 180 項目がある場合があります。

サーバー（または複数のサーバー）の論理ビューにあるシステムリンクをポータルとして考える必要があります。システムのコレクションは`/rest/v1/Systems` で記載されています。

シャーシ

これは物理ビューです。`Chassis` は物理コンテナと考える必要があります。`Chassis` 関係タイプは、`Chassis` のコレクションへのリンクです。ProLiant DL/ML/BL サーバーでは、1 つのシャーシのコレクションになります。Moonshot`Chassis` のコレクションには、カートリッジとエンクロージャー（カートリッジはシャーシの一種のため）が含まれる可能性があります。

`Chassis` は、多くの場合、電源装置、温度センサーを持ち、物理的な場所があるコンテナです。`Chassis` には論理的なコンピューターノード（システム）が含まれますが、別のシャーシを含めることもできます（他の種類のシャーシであり、そのシャーシにノードが含まれるカートリッジを含む Moonshot のシャーシを考えます）。また、マルチノードシステムや複数のシャーシに分散しているシステムに対応するシステムとシャーシの間には 1 対 1 の関係はありません。

シャーシのコレクションは`/rest/v1/Chassis` で記載されています。

マネージャー

これは、iLO 4 へのリンク、または従来 RIBCL で処理された iLO 管理タスク向けの Moonshot iLO Chassis Management Module 自体です。たとえば、ネットワーク設定、ライセンス管理、iLO のファームウェア管理などです。

シャーシのコレクションは`/rest/v1/Managers` で記載されています。

基本認証の例

以下に、認証が試みられない場合の`GET /rest/v1/Systems` で表示されるエラーを示します。

ExtendedError/ExtendedInfo 応答を見る

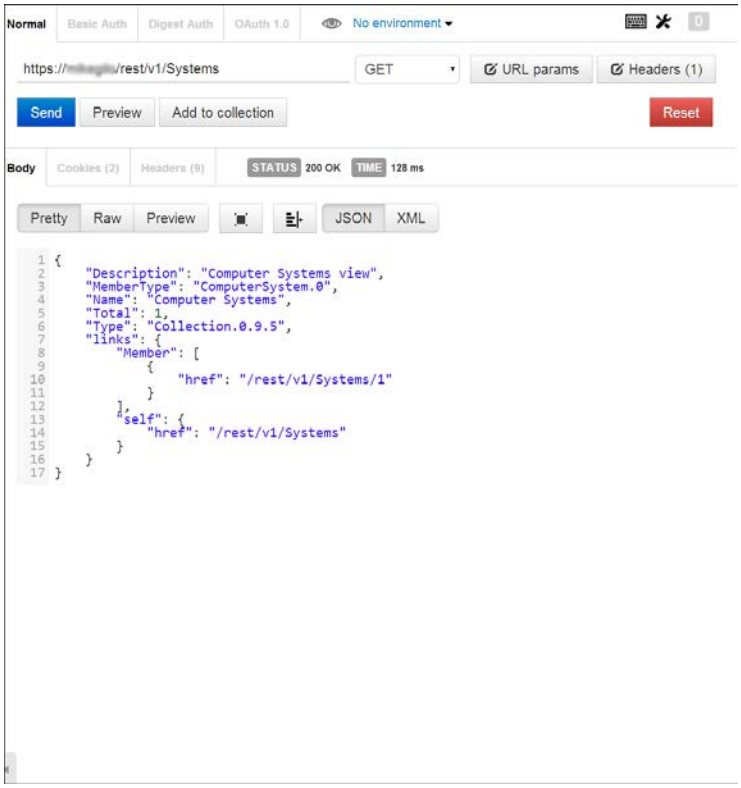
```
{
    "@odata.type": "#ExtendedInfo.ExtendedInfo",
    "Messages": [
        {
            "MessageID": "Base.0.10.NoValidSession",
        }
    ],
    "Type": "ExtendedError.1.0.0",
    "error": {
        "@Message.ExtendedInfo": [
            {
                "MessageId": "Base.0.10.NoValidSession"
            }
        ],
        "code": "iLO.0.10.GeneralError",
        "message": "A general error has occurred. See ExtendedInfo for more information."
    }
}
```

Redfish：上記の応答が互換性と Redfish 応答の組み合わせです。Redfish プロパティは、**強調表示されます**。リクエストに OData バージョンヘッダーが含まれる場合、非 Redfish プロパティは非表示になります。また、応答は互換性タイプ"`ExtendedError`" ですが、Redfish タイプは"`ExtendedInfo`" です。

Postman を使用した基本認証の追加

1. **[Basic Auth]** タブをクリックします。
2. **[Username]** および **[Password]** を入力します。
3. **[Refresh headers]** をクリックします。

   これで、基本認証の Base64 ハッシュが生成されます。
4. **[Send]** をクリックします。



CURL コマンドラインへの基本認証の追加

- `-u username:password` パラメーターを追加します。

例

```
> curl https://myilo/rest/v1/Systems -i --insecure -u username:password -L
```

```
HTTP/1.1 200 OK
Allow: GET, HEAD
Cache-Control: no-cache
Content-length: 437
Content-type: application/json
Date: Tue, 05 Aug 2014 14:39:56 GMT
ETag: W/"9349EA93"
Link: </rest/v1/SchemaStore/en/ComputerSystemCollection.json>; rel=describedby
Server: HP-iLO-Server/1.30
X-Frame-Options: sameorigin
X_HP-CHRP-Service-Version: 1.0.3

{"@odata.context":"/redfish/v1/$metadata#Systems","@odata.id":"/redfish/v1/Systems/"
,"@odata.type":"#ComputerSystemCollection.ComputerSystemCollection","Description":"Computer Systems view",
"MemberType":"ComputerSystem.1","Members":[{"@odata.id":"/redfish/v1/Systems/1/"}],"Members@odata.count":1,
"Name":"Computer Systems","Total":1,"Type":"Collection.1.0.0","links":{"Member":[{"href":"/rest/v1/Syst
ems/1"}],"self":"/rest/v1/Systems"}}
```

セッション管理

ルート `/rest/v1` リソース以外の他のリソースで `GET` を実行すると、リソースのアクセスに必要な認証権限がないことを示す HTTP 401（Forbidden）エラーを受信します。

RESTful API では、1 回限りの操作をする場合、HTTP 基本認証を使用できます。これが、Postman プラグインまたは CURL が大変便利な 1 つの理由です。これらのツールでは、基本認証情報用に適切な base64 エンコードをリクエストに簡単に挿入できます。

ログインとログアウト

さらに複雑なマルチリソース操作の場合、ログインしてセッションを確立する必要があります。ログインする場合、iLO は記載された URI `/rest/v1/Sessions` にセッションマネージャーのオブジェクトを持ちます。セッションを確立するには、以下のように JSON オブジェクトの `POST` をセッションマネージャーに発行する必要があります。

```
POST /rest/v1/Sessions with the required HTTP headers and a body containing

   {
     "UserName": "username",
     "Password": "password"
   }
```

重要: JSON形式の要求本体が含まれるすべてのRESTful API操作に対して、HTTPヘッダー `Content-Type: application/json` を含める必要があります。

セッションが正常に確立された場合、HTTP 201（Created）応答を iLO から受信します。また、さらに 2 つの HTTP ヘッダーがあります。

ヘッダー	値	説明
X-Auth-Token	使用するセッションのトークン（文字列型）です。	ログインセッションの一意の文字列です。セッション内の後に続くすべてのHTTP操作のヘッダーとして、このトークンを含める必要があります。
Location	新しく作成したセッションのリソースの URI です。	iLO は、セッションを記述する新しいセッションリソースを割り当てます。これは、ログアウトするために `DELETE` を発行する必要がある URI です。この位置の URI を失った場合、セッションコレクションのHREF リンクを**移動**することで、URIを見つけることができます。ログアウトを容易に行うために、この URI を保存します。

データモデルのナビゲーション

ログインできるようになると、データモデル全体を移動できるようになります。このモデルはhrefリンクで一緒にリンクされているため、モデル全体を移動できるクライアントを簡単に作成できます。各オブジェクトに対してリンクオブジェクト内のhrefの検索を繰り返すだけです（サブオブジェクトと配列の反復を使用）。

たとえ既知のURIから始めても、データモデルは厳密に言うとツリーではありません。データモデル全体で相互参照される可能性があるため（たとえば、シャーシからシステムへ、およびその反対）、モデルを移動しながらアクセスしたURIの辞書を作成して、無制限に移動が繰り返されることを回避します。URI の取得に `GET` を発行するときに、その URI をこの辞書に追加して、再度その URI を参照する場合は、再度 `GET` は実行しないでください。さらに推奨事項として、辞書のアクセスした URI キーを、たとえば`tolower()` を使って小文字で保存してください。データモデルの URI は、一貫した小文字と大文字の規則を適用する必要があります。しかし、バグが発生し、大文字と小文字の使用に一貫性がない場合、辞書のキーが小文字の文字列に保たれていると、走査アルゴリズムが守られます。

データモデルを移動するとき、多くのオブジェクトのタイプが`Collection`であることが分かります。

コレクション

コレクションは、RESTful API で頻繁に使用されるオブジェクトのタイプです。コレクションの一般のタイプは必ず **Collection** です（つまり、コレクションはすべて同じスキーマに準拠しており、同じプロパティ名を共有します）。コレクションは非常に用途が広く、収集するメンバーのタイプを示す`MemberType`と呼ばれるプロパティによって識別されます。`MemberType`はタイプ名とメジャーバージョンを特定するだけであり、同じコレクションで同じタイプのマイナーバージョンを混在させることができます。

コレクションは非常に柔軟に実装でき、いくつかの異なる方法でメンバーを提供できます。コレクションの2つの基本データ構造は、コレクションのメンバーへのリンク（`/links/Member`

配列）と、Items配列です。コレクションの実装方法によって異なりますが、コレクションから省略されたMembers またはItems のいずれかの配列が分かるかもしれません。

- **フォーム 1**
  子リソースへのリンク
  - /links/Member 配列を格納する。
  - /links/Member 配列に子リソースへの href がある。
  - コレクションの操作（サポートされている場合）。
    GET
    POST 作成用
- **フォーム 2**
  子リソースおよび組み込み項目へのリンク
  - /links/Member 配列を格納する。
  - /links/Member 配列に子リソースへの href がある。
  - Items 配列を格納する。
    それぞれは、完全な MemberType 表現（**項目参照**）。
  - /links/Member 配列に項目要素のフラグがある。
  - コレクションの操作（サポートされている場合）。
    GET
    POST 作成用
- **フォーム 3**
  組み込み項目のみ（読み取り専用）
  - /Links/Member 配列が省略される可能性がある。
  - Items 配列を格納する。
    それぞれは、完全な MemberType 表現。
  - 項目要素のフラグを持つ /Links/Member 配列が省略される可能性がある。
  - コレクションの操作（サポートされている場合）。
    GET

次に、各フォームの説明を記載します。

- フォーム 1 は、メンバー配列の中の href ごとに GET コマンドの使用を繰り返す必要がある、完全に変更可能なコレクションです。これは最も柔軟性と汎用性があるフォームですが、完全に反復するには多くの HTTP 操作が必要であるため、効率が低下します。このフォームのコレクションでは、項目の作成に POST が許可され、子リソースでは項目の削除に DELETE が許可される場合があります。子では、項目の変更に PATCH が許可される場合があります。この形式のコレクションは、メンバーの作成と削除がサポートされ、繰り返しがあまり重要でないセッションとアカウントで使用します。
- フォーム 2 はフォーム 1 の変種であり、コレクションオブジェクト自身のサブセットまたは完全項目表現のいずれか、および子リソースへのリンクを格納するItems配列を含みます。メンバーのサブセットが実装担当者によって選択されている場合、Items配列の最上

位の情報を参照することで、コレクションメンバーを**参照**して、対象の項目に別々の GET を発行することを決めてもかまいません。このため、どの子に GET を発行するかをクライアントが決められることで、拡張性が向上します。Items 配列表現は、項目を省略してスペースを節約するため、MemberType スキーマに完全には準拠している必要はありません。Items 配列では子リソースを置き換えませんが、データを区別するキーを参照できます。

- フォーム 3 は、項目の作成、削除、または変更が必要のない、読み取り専用のコレクションです（たとえば、PCI デバイスやスロット）。子リソースはなく、リンクセクションがない場合があります。Items 配列では、スキーマに準拠したメンバー全体を送信します。利点は、サイズの効率が良いこと、リンクセクションがないこと、別々の GET 操作を使用して繰り返す必要がないことです。これは、読み取り専用の配列として表現されるものに最適です。

RESTful API でコレクションを繰り返すには、Python コード例の collection() を参照してください。これは、yield キーワードを使用して各コレクションメンバーを返す、Python の生成関数です。

コレクションの例

```
> curl https://myilo/rest/v1/Systems -i --insecure -u username:password -L
```


```
{
    "@odata.context": "/redfish/v1/$metadata#Systems",
    "@odata.id": "/redfish/v1/Systems/",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Description": "Computer Systems view",
    "MemberType": "ComputerSystem.1",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/1/"
        }
    ],
    "Members@odata.count": 1,
    "Name": "Computer Systems",
    "Total": 1,
    "Type": "Collection.1.0.0",
    "links": {
        "Member": [
            {
                "href": "/rest/v1/Systems/1"
            }
        ]
    }
}
```


Redfish：Redfish には、汎用の"Collection" タイプはなくなりました。Collections がまったく同じように見えますが、一般的なタイプではありません。代わりに、ComputerSystemCollection と ChassisCollection などがあり、MemberType プロパティは必要ありません。Members アレイが、現在、オブジェクトのルートであり、合計は Members@odata.count で置き換えられるようになりました。Redfish 固有のプロパティが**強調表示され**、クライアントが OData バージョンヘッダーを提供する場合、非 Redfish プロパティは表示されません。Redfish プロパティは、互換性のあるプロパティと同じ情報を、異なる形式で通信します。

コンピューターシステム

ComputerSystems は、データモデルで演算ノードを示します。ProLiant DL ラックマウント型サーバーには単一の演算ノードがある場合がある一方、完全にロードされた Moonshot シャー

シは、180 の演算ノードを含む場合があります。データモードでのすべての ComputerSystem リソースを繰り返す方法の例は次のとおりです。

Python：Python コードの例`ex1_change_bios_setting()` を参照してください。これは、`/rest/v1/Systems` でのコレクションの繰り返し関数を使用した、最上位レベルのループを使用します。

シャーシ

シャーシは、演算リソース用の物理または仮想のコンテナーを表します。たとえば、ProLiant DL ラックサーバーには、サーバーの金属製のコンテナーを表す単一のシャーシがあります。シャーシのリソースでは、サーバーの物理的側面について説明します。データモードですべてのシャーシリソースを繰り返す方法の例は次のとおりです。

注記: **Python**：Python コードの例の`ex29_get_PowerMetrics_Average()` を参照してください。

シャーシのオブジェクトに`GET` を発行すると、属性を参照できます。属性は、`ChassisType`、システムからの共有されるプロパティ（`AssetTag`）、ディメンション情報などです。

`ChassisType` は、さまざまなものに過負荷がかかるため、`Chassis` を別の`Chassis` の中に置くことができます。たとえば、ブレードのシャーシはブレードエンクロージャのシャーシ内に置かれ、さらにラック（シャーシともみなされます）内に置かれます。`contains` と `contained by` の関係は、シャーシオブジェクトのリンクセクション内にあります。

マネージャー

Manager のリソースは、iLO 4 自体を表します。

iLO オブジェクト（`Manager` など）に `GET` を発行すると、属性を参照できます。属性は、`Model`、`Firmware` 情報、さまざまなiLO サービスへのリンクなどです。サービスには、ライセンスサービス、ネットワークサービス、iLO NIC 情報および構成リソースなどがあります。

データモデルですべての Manager リソースを繰り返す方法の例

Python：Python コードの例`ex7_find_iLO_MAC_address()` を参照してください。

注記: `/rest/v1/Managers` でのコレクションの繰り返し関数を使用する最上位レベルのループです。

スキーマ

各リソースタイプは、オブジェクト、許可されたプロパティ、要求されたプロパティ、タイプ、およびその他の情報のフォームを定義するスキーマファイルを備えています。データモデルは、ドラフト 4 の JSON スキーマ基準を採用して定義を行います。詳しくは、**http://json-schema.org** を参照してください。Python スキーマ検証用のパッケージをダウンロードするには、**https://github.com/Julian/jsonschema** を参照してください。パッケージは、対応するスキーマに対して他の言語でリソースを検証する場合にも使用できます。これは、業界標準のツールチェーンを活用する戦略の一部です。

スキーマではリソースの **class** が定義され、クラスの各インスタンスは、利用可能なプロパティのサブセットを実装する場合があります。スキーマ内のすべてのプロパティが、必ずしもすべてのインスタンスに実装されるわけではない点に注意する必要があります。これにより、以下の 1 つまたは複数の結果となる可能性があります。

- プロパティは、特定の Hewlett Packard Enterprise サーバーモデルに適用されない場合があります。

注記: 各プロパティは、`readonly = true` または `readonly = false` として特定されます。これは、プロパティで `PATCH` 操作は実行できますが、`PATCH` を許可する`Allow` ヘッダーがリソースに存在する場合に限られることを示しています。2 つのリソースは同じ`Type` を共有する場合がありますが、`readonly = false` のプロパティに対しても、異なる`Allow` ヘッダー（`PATCH` 操作を有効または無効にするヘッダー）を持つ可能性があります。

メッセージレジストリおよびスキーマの検索方法

RESTful API でデータを定義するスキーマを検索するには、オンラインとオフラインの 2 つの方法があります。オンラインは、スキーマ情報が、iLO への`GET` 操作を実行して使用可能であることを意味します。オフラインは、スキーマファイルは、HPE ProLiant Support Pack DVD で利用可能であることを意味します。

Service Pack for ProLiant のオフラインの.zip ファイル

Service Pack for ProLiant（SPP）には、スキーマと RESTful API のレジストリで構成される複数の ZIP ファイルを含んでいる `hp_restful_api` と呼ばれるフォルダーがあります。

`/Schemas/index.json` または `/Registries/index.json` のどちらかを読み取ることで検索を開始する点を除いて、iLO の RESTful インターフェイスにある構造と同一の構造が、この.zip ファイルの中にあります。これら 2 つの `index.json` ファイルの形式は、`/rest/v1.Schemas/href` で参照される iLO リソースの `SchemaFile.0.9.5` 形式に一致します。

さらに SPP には、サポートされている各 ProLiant サーバーの UEFI BIOS に対して、公開されたスキーマとレジストリの.zip ファイルが含まれます。これらの.zip ファイルの名前は `hp-rest-classes-bios-P89-1.00_07_11_2014.zip` です。ここで、`P89` は ROM のファミリ名であり（各 ProLiant システム）であり、`1.00-07_011_2014` は ROM のバージョンと日付です。

オフラインのスキーマおよびレジストリの.zip ファイルをダウンロードするには、**http://www.hpe.com** を参照してください。

スキーマバージョンの処理：上位および下位互換性

RESTful API 内のリソースは、名前とバージョンでタイプ付けされます（例：“ServiceRoot.1.0.0”）。これは、正しいスキーマだけでなく、正確なバージョンも識別するために十分な情報を提供します。正確なスキーマバージョンがオンラインでもオフラインでも使用できない可能性があるケースがあります（たとえば BIOS アップデートは、iLO がそのスキーマリポジトリで表示するより新しいスキーマを使用する場合があります）。このため、スキーマを検索する際に、クライアントコードで**最も近い**いくつかの形式を実行することが必要です。たとえば、リソースが“HpBaseConfigs.1.1.0” タイプで、唯一の利用可能なスキーマが“HpBaseConfigs.1.0.3” である場合、またはその反対の場合などです。

最良一致を実現するために、以下のガイダンスを提供します。

- スキーマファイル（フォルダー内でオフライン）は、一般的に.json 拡張子付きでスキーマ用に名前が付けられていますが、これはスキーマの正式な名前ではありません。スキーマの正式な名前のベース名は、スキーマオブジェクトのtitle プロパティに含まれます。さらに、Redfish でいくつかのタイプの名称変更のため、スキーマの古い名前がoldtitle プロパティで利用可能な場合があります。次に例を示します。

```
"title": "EthernetInterface.1.0.0",
"oldtitle": "EthernetNetworkInterface.0.92.0",
```

- 名前と一緒に、スキーマのメジャーバージョンでは、スキーマの互換性のあるファミリを定義します。たとえば、EthernetInterface.2.x.y は、必ずしもEthernetInterface.1.x.y と互換性がありません。例外として、バージョン 0 から 1 への移行で、スキーマの初期バージョンが完成したため、バージョン 1 はバージョン 0 スキーマとの上位互換性があると考える必要があります。
- それ以降のマイナーとエラータのバージョン番号は、上位互換性の違いを示しています。たとえば、EthernetInterface1.1.1 はEthernetInterface.1.0.0 のスーパーセットであり、クライアントが 1.0.0 プロパティ定義のみを使用している場合、クライアントは 1.0.0 スキーマと同じくらい簡単に 1.1.1 を使用できます。ただし、クライアントが 1.1.1 で追加されたプロパティを使用する必要がある場合は、新しいデータは 1.0.0 タイプのリソース内に存在しません。

1 つの可能な**最良一致**のスキーマアルゴリズムは、リソースのバージョンとすべての候補スキーマの数学的な違いを算出し、違いが最小のスキーマを選択することです。

5 iLO RESTful API を使用した例

この項では、iLO 向け REST API を使用して特定のタスクを完了する例について説明します。

Python を使用した iLO RESTful API へのアクセス

Python は、REST API 開発でよく使用されている言語で、REST クエリ用の強力なツールが含まれています。`ServiceRoot` リソースへの Python クエリの簡単な例を以下に示します。

```
>>> from httplib import HTTPSConnection
>>> from base64 import b64encode
>>> userid='iloUserID'
>>> passwd='iloPassword'
>>> auth='BASIC ' + b64encode(userid + ":" + passwd)
>>> header = {'Authorization': auth}
>>> conn = HTTPSConnection(host='iloHostName', strict=True)
>>> conn.request('GET', '/rest/v1', headers=header)
>>> conn.getresponse().read()
```

この要求の簡単な例で、含まれている基本的なライブラリおよび要求を正しく形成する方法を示します。SSL や証明書検証プロセスに関連するセキュリティを強化したり緩めたりするために使用できる追加のパラメーターが数多くあります。Httplib ドキュメントには、より堅牢なライブラリ機能の概要が示されています。

Httplib のような組み込みライブラリ以外に、REST 操作を実行するため Python で使用できる一般的なライブラリは他にも多数あります。要求ライブラリは、現在入手可能な最も一般的な HTTP Python ライブラリの 1 つであり、REST API 開発にも適しています。

RESTful API とともに Python を使用する方法について詳しくは、**https://github.com/HewlettPackard/python-proliant-sdk** にある一連の例を参照してください。

BIOS

この項の例は、iLO シャーシマネージャーが動作する Moonshot サーバーには適用できません。

Python：Python コード例の`ex1_change_bios_setting()` を参照してください。ここでは BIOS 設定と PATCHing の新しい値の検索を表しています。

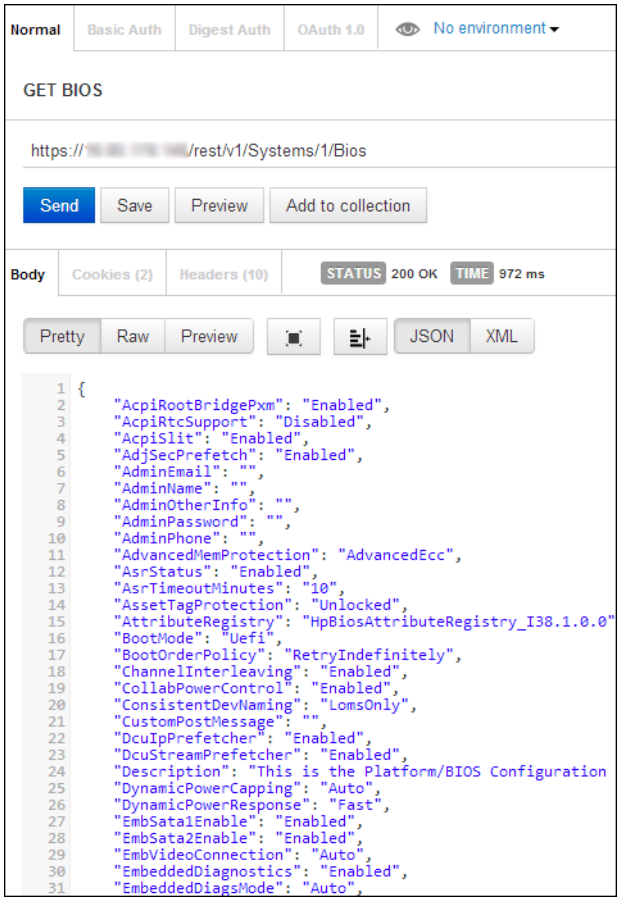
注記: ProLiant Gen9 サーバー以降では、BIOS 設定を変更するために iLO RESTful API を使用できます。

BIOS について

現在の BIOS 構成に `GET` を実行するには、以下の手順を実行します。

iLO RESTful API は、UEFI BIOS 構成を有効にします。BIOS は、現在のアーキテクチャではシステムレベルのエンティティです。BIOS構成へのリンクは、コンピューターシステムのノードオブジェクトからです。

```
SystemCollection = GET /rest/v1/Systems
For each item in SystemCollection.links.Member # this is a collection
  System = GET item.href
  BIOS = GET System.Oem.Hp.links.BIOS.href
```



GET BIOS 操作の結果は、フラットオブジェクトの中にあります。PATCH を使用してプロパティ値を更新することはできません。このリソースでGET 操作を実行するだけでできます。

設定の変更と SettingsResults の理解

BIOS の現在の構成オブジェクトは読み取り専用です。このオブジェクトには、PATCH 操作を実行できる Settings リソース（保留中の設定）へのリンクが含まれます。Settings リソースに GET を実行すると、PATCH コマンドを実行できることが分かります。プロパティを変更し、そのプロパティに PATCH を実行して、Settings URI に戻ることができます。設定の変更は、サーバーがリセットされるまでは反映されません。サーバーがリセットされる前は、現在の設定と保留中の設定を別々に使用できます。サーバーがリセットされた後は、保留中の設定が適用され、メインオブジェクトの SettingsResults プロパティ内にあるすべてのエラーを参照することができます。

別々の 2 つのリソースを使用する利点

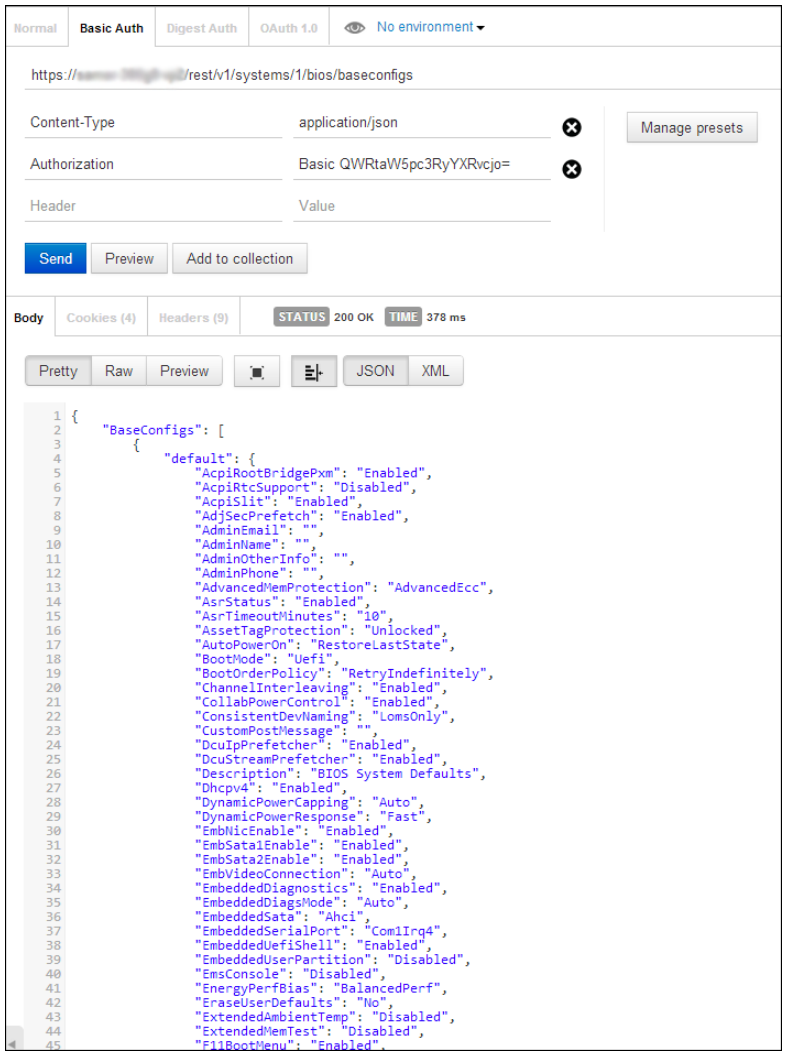
- オフラインコンポーネント（たとえば、BIOS）を有効にして、設定の変更を後で処理できます。
- オフラインのコンポーネントが保留中の設定を処理するまで、再検討のために現在の値と保留中の値を使用可能のままにできます。
- 複雑なジョブキューの必要がありません。

BIOS のデフォルト値の読み込みの例

BIOS の現在の構成オブジェクトには、BIOS のデフォルト設定値をリストする個別の読み取り専用オブジェクト BaseConfigs へのリンクが含まれています。BIOS BaseConfigs リソースに対して GET を実行するには、以下のように記述します。

```
SystemCollection = GET /rest/v1/Systems
For each item in SystemCollection.links.Member # this is a collection
```

```
System = GET item.href
BIOS = GET System.Oem.Hp.links.BIOS.href
BaseConfig = GET BIOS.links.BaseConfigs.href
```



`BaseConfigs` には、デフォルトのセット（または基本構成セット）の配列が格納されています。各基本構成セットには、BIOS のプロパティとそのデフォルト値のリストが含まれています。デフォルトの基本構成セットには、BIOS の製造時のデフォルト値が含まれています。`BaseConfigs` の場合、ユーザーのカスタムデフォルト値の`default.user` のような、他のセットを格納することも可能です。

BIOS リソースと属性レジストリの概要

BIOS リソースの形式は、他のほとんどのリソースとは異なっています。BIOS リソースは、すべてのオブジェクトが準拠しているスキーマタイプに準拠しています。ただし、BIOS 設定はサーバータイプと BIOS のリビジョンで大きく変わります。このため、BIOS 設定の可能なすべてのプロパティを定義する標準的なスキーマを公開することは非常に困難です。さらに、相互設定の依存関係や、json スキーマのメニュー構造など、いくつかの高度な設定を伝えることはできません。したがって、BIOS は**属性レジストリ**を使用します。

属性レジストリの例

BIOS 構成オブジェクトに `GET` を実行すると、`Type` を含む基本構造が残っていることが分かります。ただし、`AttributeRegistry` と呼ばれるプロパティもあります。このプロパティは、レジストリファイルを指しています。このファイルには、メニューの構造情報、別の設定

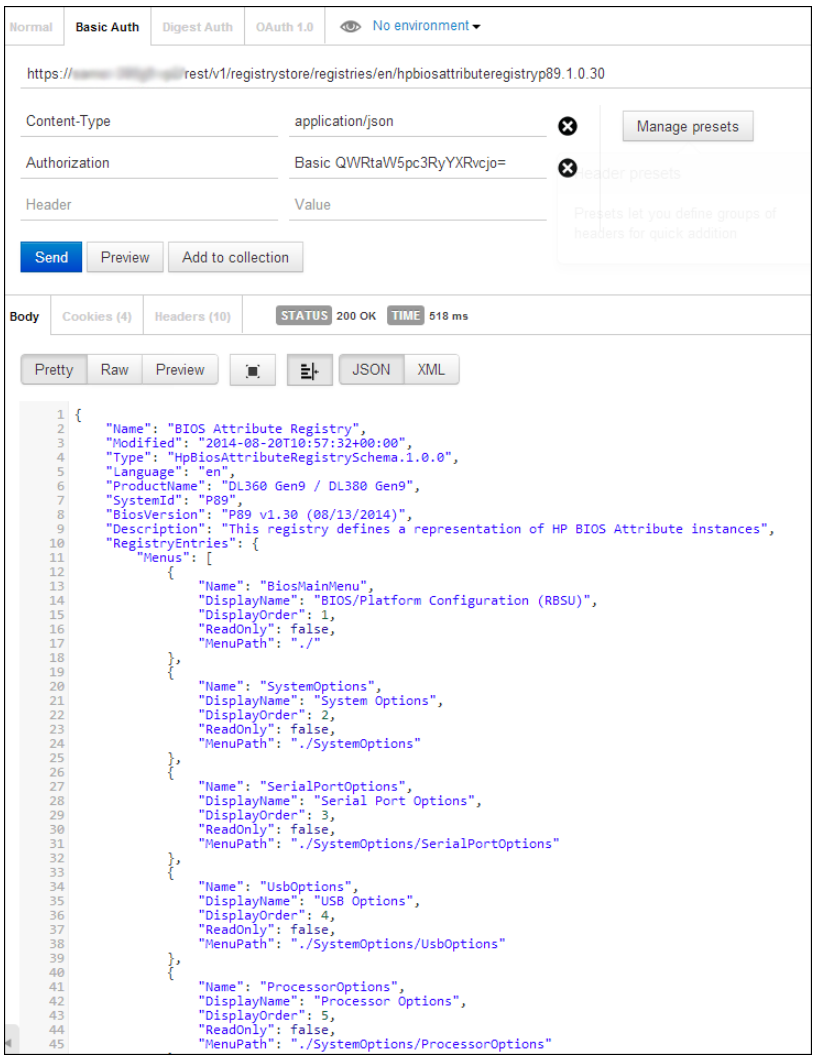
との依存関係、その他の情報など、各 BIOS 設定（たとえば、`PowerProfile`）の一連のデータが格納されています。

BIOS 属性のレジストリリソースを検出するには、以下のように記述します。

```
SystemCollection = GET /rest/v1/Systems
For each item in SystemCollection.links.Member # this is a collection
  System = GET item.href
  BIOS = GET System.Oem.Hp.links.BIOS.href
  AttributeRegistryName = the value of the "AttributeRegistry" property in BIOS object

RegistryCollection = GET /rest/v1/Registries
For each item in RegistryCollection.links.Member # this is a collection
  Registry = GET item.href
  If Registry.Schema beginswith AttributeRegistryName
  For each language in Registry.Location
    If language.Language == "en"  # or choose another
      AttributeRegistry = GET language.Uri.extref
```

BIOS 属性レジストリは、そのサイズにより、JSON リソース（gzip）として圧縮されるため、返される HTTP ヘッダーは`gzip` のコンテンツエンコーディングを示します。REST クライアントは、リソースを解凍する必要があります。Web クライアント（Postman プラグインなど）を実行すると、自動的に解凍されます。



BIOS 属性

以下のような、BIOS 属性（設定）とそのメタデータのリストです。

- BIOS 属性ごとのタイプ（列挙型、文字列、数値、または論理値）。
- 列挙型属性に可能な値。

- 属性と可能な値の表示文字列（レジストリ言語にローカライズ）。
- ヘルプテキストと警告テキスト（各国語版）。
- 場所と表示順序の情報（属性のメニュー階層を含む）。
- 数値属性の最大値、最小値、およびステップ値、文字長の最大値と最小値、文字列属性の正規表現など、値の上限値。
- その他のメタデータ。

BIOS 属性レジストリ

BIOS 属性レジストリには、次の 3 つの最上位レベルの配列が含まれています。

- **Menus**：BIOS 属性メニューとその階層を含む配列。たとえば、ローカルの BIOS 設定に似ているユーザーインターフェイスを構築する場合、または`ProcessorOptions` や `UsbOptions` などの関連するプロパティをグループ分けする場合に使用できます。
- **Dependencies**：このサーバー上の BIOS 属性の依存関係のリストを含む配列。ある BIOS 設定が、別の BIOS 設定の値に基づいて、その値またはその`ReadOnly` プロパティを変更する要因となる可能性のある相互設定の依存関係が含まれます。
- **BaseConfigs**：BIOS 属性の製造時のデフォルト値のリストを含む配列。これは、BaseConfigs リソースの読み取りと、`default` という名前のオブジェクトの解析に相当します。

管理者 BIOS パスワードの更新例

管理者および電源投入時のパスワードを変更するには、それぞれのパスワードの 2 つのプロパティを変更する必要があります。

- 管理者パスワード：`AdminPassword` に新しいパスワード、`OldAdminPassword` に古いパスワードを設定します。
- 電源投入時パスワード：`PowerOnPassword` に新しいパスワード、`OldPowerOnPassword` に古いパスワードを設定します。

古いパスワードを設定しない場合は、古いパスワードプロパティに空白文字列（""）を使用する必要があります。

1. `/rest/v1/Systems` を繰り返して、メンバー ComputerSystem を選択します。

   結果 = *{ilo-ip-address}*`/rest/v1/Systems/1/BIOS`
2. `Bios` と呼ばれる`Oem/Hp/links` でリンクを見つけて、`BiosURI` を書き留めます。
3. `BiosURI` から`BiosObj` に`GET` を実行して、`GET` だけが許可されていることに注目します（これは現在の設定です）。
4. `Settings` と呼ばれる`BiosObj` でリンクを見つけて、この URI を書き留めます。
5. `{"AdminPassword":"@Pa$$w0rd", "OldAdminPassword":""}` に変更された `AdminPassword` と`OldAdminPassword` プロパティを使用して、新しい JSON オブジェクトを作成します。
6. BIOS 設定を更新します。要求本体の更新済み`AdminName` プロパティを送信するだけです。

   `PATCH` *`{ilo-ip-address}`*`/rest/v1/Systems/1/BIOS/Settings`

   サーバーがリセットされたときに、BIOS 設定が検証され、適用されます。

BIOS 設定の更新例

最小で必要なセッション ID の権限は、構成です。

1. `/rest/v1/Systems` を繰り返して、メンバー ComputerSystem を選択します。

   結果 = *{ilo-ip-address}*`/rest/v1/Systems/1/BIOS`

2. `Bios` と呼ばれる`Oem/Hp/links` でリンクを見つけて、`BiosURI` を書き留めます。
3. `BiosURI` から`BiosObj` に`GET` を実行して、`GET` だけが許可されていることに注目します（これは現在の設定です）。
4. `Settings` と呼ばれる`BiosObj` でリンクを見つけて、この URI を書き留めます。
5. 手順 4 の URI を使用して、BIOS 設定を取得します。

   `GET {ilo-ip-address}/rest/v1/Systems/1/BIOS/Settings`
6. `{"AdminName":"Joe Smith"}` に変更された`AdminName` プロパティを使用して、新しい JSON オブジェクトを作成します。
7. BIOS 設定を更新します。要求本体の更新済み`AdminName` プロパティを送信するだけです。

   `PATCH {ilo-ip-address}/rest/v1/Systems/1/BIOS/Settings`
8. BIOS 設定を取得して、`AdminName` プロパティが変更されたことを確認します。

   `GET {ilo-ip-address}/rest/v1/Systems/1/BIOS/Settings`

   サーバーがリセットされたときに、BIOS 設定が検証され、適用されます。

詳細情報

Python：Python コード例の`ex1_change_bios_setting()` を参照してください。ここでは BIOS 設定と PATCHing の新しい値の検索を表しています。

BIOS UEFI セキュアブートの有効化の例

最小で必要なセッション ID の権限は、構成です。

1. /rest/v1/Systems を繰り返して、メンバー ComputerSystem を選択します。`PATCH` 操作を許可しているタイプ`HpSecureBoot` の子リソースを見つけます（この例題でないと複数存在する可能性がありますが、1 番目を選択します）。

   `{ilo-ip-address}/rest/v1/Systems/1/SecureBoot`
2. セキュアブート設定を取得します。

   `GET {ilo-ip-address}/rest/v1/Systems/1/SecureBoot`
3. `{"SecureBootEnable":true}` に変更された`SecureBootEnable` プロパティを使用して、新しい JSON オブジェクトを作成します。
4. セキュアブート設定を更新します。要求本体の更新済み`SecureBootEnable` プロパティを送信するだけです。

   `PATCH {ilo-ip-address}/rest/v1/Systems/1/SecureBoot`

   サーバーがリセットされたときに、ブート設定が検証され、適用されます。

BIOS UEFI 設定をデフォルトに戻す例

BIOS 設定のリソースは、選択されたリソースの BIOS 設定をデフォルトに戻すことができる特別な機能をサポートしています。これを行うには、BIOS 設定オブジェクトの特別なプロパティ`{"BaseConfig": "default"}` で `PATCH` または `PUT` 操作を実行します。これを、他のプロパティセットと組み合わせて、まずデフォルト値を設定してから、1 つの操作ですべての特定の設定をセットすることができます。

注記: `BaseConfig` プロパティは、BIOS または BIOS 設定のリソースにまだ存在しない可能性があります。BIOS のリソースが、デフォルトの設定に戻す機能をサポートしているかどうかを判断するには、BIOS BaseConfigs リソースに `GET` を実行して、`Capabilities` プロパティを参照します。

最小で必要なセッション ID の権限は、構成です。

1. /rest/v1/Systems を繰り返して、メンバーComputerSystem を選択します。PUT 操作を許可しているタイプHpBios の子リソースを見つけます（この例題でないと複数存在する可能性がありますが、1 番目を選択します）。

   {ilo-ip-address}/rest/v1/Systems/1/BIOS/Settings

2. BIOS の UEFI 設定を取得します。

   GET *{ilo-ip-address}*/rest/v1/Systems/1/BIOS/Settings

3. 応答本体でBaseConfig プロパティを{"BaseConfig":"default"} に変更または追加します。
4. BIOS UEFI 設定を更新します。

   PUT *{ilo-ip-address}*/rest/v1/Systems/1/BIOS/Settings

   サーバーがリセットされたときに、BIOS の UEFI 設定はデフォルトに戻されます。

注記:

- リソースタイプHpBaseConfigs を見つけて、BIOS 設定のデフォルト値を参照することも可能です。

  {ilo-ip-address}/rest/v1/Systems/1/BIOS/BaseConfigs

- BaseConfig を他のプロパティ値と組み合わせて、まずすべてをデフォルトにリセットしてから、1 つの操作でいくつかの特定の設定を適用することができます。

iSCSI Software Initiator 構成の例

iSCSI Software Initiator では、ブートソースとして使用するように iSCSI ターゲットデバイスを構成できます。BIOS の現在の構成オブジェクトには、タイプHpiSCSISoftwareInitiator の別個のリソースへのリンクが含まれています。BIOS の現在の構成リソースと iSCSI Software Initiator の現在の構成リソースは読み取り専用です。iSCSI 設定を変更するには、PUT および PATCH 操作を許可するSettings リソースへの別のリンクをたどる必要があります。

iSCSI ターゲット構成は、オブジェクトの配列であるiSCSIBootSources プロパティで表します。それぞれに、1 つのターゲットの設定が含まれます。配列のサイズは、同時に構成可能な iSCSI ブートソースの総数を表します。変更可能なプロパティが多数存在します。たとえば、[1, *N*]（*N* はブートソースの配列サイズです）の範囲で一意の整数に設定できる iSCSIBootAttemptInstance があります。デフォルトでは、すべてのオブジェクトでこの**インスタンス番号**は 0 に設定されています。この値は、iSCSI の構成時に、オブジェクトが無視されることを意味します。

各オブジェクトには、2 つの読み取り専用プロパティ（StructuredBootString および UEFIDevicePath）も含まれます。これらのプロパティは、ターゲットがブートソースとして正常に構成された場合にのみ、値が設定されます。各プロパティについて詳しくは、対応するスキーマを参照してください。

iSCSI イニシエーターの名前は、iSCSIInitiatorName プロパティによって表されます。

もう 1 つの読み取り専用プロパティiSCSINicSources は、iSCSI の現在の構成リソースにだけ表示されます。このプロパティは、iSCSI ブート構成のターゲットとして使用できる NIC インスタンスを表す文字列の配列です。各文字列に対応する NIC デバイスを確認するには、その他の 2 つのリソースを相互参照することをお勧めします。

- タイプHpBiosMapping のリソースは、BIOS の現在の構成リソースのMappings リンクを介して検出できます。そのBiosPciSettingsMappings プロパティには、BIOS 固有のデバイス文字列（NIC ソース文字列など）とCorrelatableID 文字列（BIOS 以外のコンテキストで同じデバイスを参照するために使用）のマッピングの配列が含まれます。
- ComputerSystem リソースの PCIDevices リンクを介してHpServerPciDevices のコレクションを検出できます。NIC インスタンスに対応する特定の PCI デバイスは、通常 UEFIDevicePath に一致するCorrelatableID を検索することによって検出できます。

HpServerPciDevice リソースを検出すると、NIC ソースを判別する助けとなる、人間が読める形式のすべてのプロパティにアクセスできるようになります。

iSCSIBootSources および iSCSIInitiatorName 設定の変更は、PATCH 操作によって実行できます。これは、HpBios 設定を変更する方法と非常に似ています。ただし、すべての BIOS 設定は単一のフラットオブジェクト内に配置されますが、iSCSI 設定は配列やサブシステムにネストされます。PATCH 操作を実行する場合は、変更**しない**ブートソースオブジェクトの代わりに空のオブジェクト（{}）を使用します。

次の例では、2 つの iSCSI ブートソースを構成した後、既存の設定の一部を編集し、3 番目のソースを追加しようとしています。

1. /rest/v1/Systems を繰り返して、メンバーComputerSystem を選択します。PATCH 操作を許可する子のリソースタイプHpiSCSISoftwareInitiator を見つけます。

   {ilo-address}/rest/v1/Systems/1/BIOS/iSCSI/Settings

2. 既存のiSCSIBootSources の配列を調べます。変更したいブートソースを見つけるため、各オブジェクトのiSCSIBootAttemptInstance プロパティを調べる必要があります。

   既存のサンプルリソース：

   ```
   {
              "iSCSIBootSources": [
                  {
                      "iSCSIBootAttemptInstance": 1,
                      ...
                  },
                  {
                      "iSCSIBootAttemptInstance": 2,
                      ...
                  },
                  {
                      "iSCSIBootAttemptInstance": 0,
                      ...
                  },
                  {
                      "iSCSIBootAttemptInstance": 0,
                      ...
                  }
              ],
              ...
          }
   ```

3. iSCSIBootSources プロパティで新しい JSON オブジェクトを作成します。

   ```
   {
              "iSCSIBootSources": [
                  {},
                  {
                      "iSCSIConnectRetry": 2
                  },
                  {
                      "iSCSIBootAttemptInstance": 3,
                      "iSCSIBootAttemptName": "Name",
                      "iSCSINicSource": "NicBootX",
                      ...
                  },
                  {}
              ]
          }
   ```

   変更しないことを示すため、インスタンス1 の位置に空のオブジェクトを使用します。変更するプロパティを含むインスタンス2 の位置にオブジェクトを使用します。省略されたプロパティはすべてそのままで変更されません。

新しいブートソースを追加するには、インスタンス0 の位置を見つけて、すべての新しい設定と、最も重要なことですが、iSCSIBootAttemptInstance の新しい一意の値とが含まれたオブジェクトと交換します。

4. iSCSI Software Initiator の設定を変更します。

   PATCH {ilo-address}/rest/v1/Systems/1/BIOS/iSCSI/Settings

ブート設定

UEFI ブート構造化された名前の文字列

この UEFI ブート構造化された名前の文字列は一意で、システム内の各 UEFI ブートオプションを表します。ソフトウェアは、この仕様で定義されている文字列の固定形式を使用して、デバイスを識別および操作できます。ソフトウェアは、文字列は UEFI BootOrder のブートデバイスごとに一意であると想定できます。

UEFI ブート構造化された名前の文字列は、以下の形式を使用して、'.'文字で区切られたセクションに分かれています。

<DeviceType>.<Location>.<Instance>.<Sub-instance>.<Qualifier>

- **DeviceType**：最初のセクションは、デバイスの種類（たとえば、HD、CD、NIC、および PCI）を示します。
- **Location**：2 番目と 3 番目のセクションは、2 つでデバイスの位置を示します（たとえば、Slot.7 またはEmb.4）。
- **Instance**：3 番目のセクションは、*Location* セクションと一緒に使用して、デバイスの位置を示します（たとえば、スロット番号または内蔵のインスタンス番号）。
- **Sub-instance**：4 番目のセクションはオプションであり、同じインスタンスを使用する複数のブートオプションの場合にサブインスタンス番号として使用されます。たとえば、これは、マルチポート NIC のポート番号にすることができます。
- **Qualifier**：5 番目のセクションはオプションであり、論理プロトコルを示します（たとえば、IPv4、IPv6、および iSCSI）。

構造化されたブート文字列情報はHpServerBootSettings オブジェクトのBootSources[] プロパティおよびHpServerPciDevice オブジェクトのStructuredName プロパティの一部です。

UEFI ブート構造化された名前の文字列の例

表 1 例

HD.Emb.4.2	内蔵 SA コントローラーベイ 4 内のハードディスクドライブの 2 番目のインスタンス
NIC.Slot.7.2.IPv4	PCIe スロット 7 の NIC のポート 2、PXE IPv4 に対して有効
NIC.FlexLOM.1.1.IPv6	内蔵 NIC FlexLOM のポート 1、PXE IPv6 に対して有効
PCI.Slot.6.1	スロット 6 の PCIe カード
HD.FrontUSB.2.2	前面の USB ポート 2 にあるフラッシュドライブの 2 番目のパーティション

表 2 現在サポートされている構造化されたブート文字列の例

デバイスタイプ	位置	インスタンス	サブインスタンス	修飾子	構造化されたブート文字列の例
Smart アレイのハードディスクドライブ	内蔵	ベイ番号	LUN ごとにインクリメント		HD.Emb.1.1
	スロット	スロット番号	LUN ごとにインクリメント		HD.Slot.1.1
Smart アレイコントローラー	内蔵	コントローラーインスタンス	1		RAID.Emb.1.1
	スロット	スロット番号	1		RAID.Slot.1.1
Dynamic Smart アレイコントローラー（ソフトウェアRAID）	内蔵	1	1		Storage.Emb.1.1
Dynamic Smart アレイコントローラー（ソフトウェアRAID）	スロット	コントローラーインスタンス	1		Storage.Slot.1.1
SATA ハードディスクドライブ	内蔵	SATA ポート番号 1			HD.Emb.1.1
SATA コントローラー	内蔵	コントローラーインスタンス	1		SATA.Emb.1.1
その他のすべてのストレージコントローラー（FC、SAS など）	内蔵	1	1		Storage.Emb.1.1
	スロット	スロット番号	1		Storage.Slot.1.1
ネットワークアダプター	LOM	NIC 番号 1 番目の NIC は 1 2 番目の NIC は 2	ポート番号	IPv4 または IPv6 または iSCSI または FCoE	NIC.LOM.1.2.IPv4 NIC.LOM.1.2.IPv6
	FlexibleLOM	FlexibleLOM 番号 1 番目の FlexLOM は 1 2 番目の FlexLOM は 2	ポート番号	IPv4 または IPv6 または iSCSI または FCoE	NIC.FlexLOM.2.1.IPv4 NIC.FlexLOM.2.1.IPv6
	スロット	スロット番号	ポート番号	IPv4 または IPv6 または iSCSI または FCoE	NIC.Slot.3.2.Ipv4
ファイバーチャネルアダプター	スロット	スロット番号	ポート番号	IPv4 または IPv6 または iSCSI または FCoE	PCI.Slot.3.1
OS ブートエントリー（「Windows ブートマネージャー」など）	スロット 内蔵		インクリメント		HD.Emb.1.2 HD.Slot.1.2

表 2 現在サポートされている構造化されたブート文字列の例 (続き)

デバイスタイプ	位置	インスタンス	サブインスタンス	修飾子	構造化されたブート文字列の例
USB キー	前面の USB	USB ポート番号	LUN ごとにインクリメント		HD.FrontUSB.1.1
	背面の USB	USB ポート番号	LUN ごとにインクリメント		HD.RearUSB.1.1
	内部 USB	USB ポート番号			HD.InternalUSB.1.1
	iLO 仮想メディア				HD.Virtual.1.1
ISO イメージ	iLO 仮想メディア				CD.Virtual.2.1
仮想インストールディスク（VID）	内蔵ストア	USB ポート番号			HD.VirtualUSB.1.1
内蔵ユーザーパーティション	内蔵ストア	USB ポート番号			HD.VirtualUSB.2.1
USB CD/DVD	前面の USB	USB ポート番号			CD.FrontUSB.1.1
	背面の USB	USB ポート番号			CD.RearUSB.1.1
	内部 USB	USB ポート番号			CD.InternalUSB.1.1
SD カード	SD スロット	USB ポート番号			HD.SD.1.1
フロッピー	前面の USB 背面の USB	USB ポート番号			FD.FrontUSB.1.1 FD.RearUSB.1.1
内蔵 UEFI シェル	内蔵	1	1		Shell.Emb.1.1
UEFI アプリケーション（ROM ファームウェアに組み込み） （診断、システムユーティリティなど）	内蔵	1	インクリメント		App.Emb.1.1 App.Emb.1.2 App.Emb.1.3
ファイル	URL	さまざまな URL 1 ずつ増加	1		File.URL.1.1
HPE RAM ディスクデバイス	RAM Memory。読み出し専用メモリ	1	ポート番号		RAMDisk.Emb.1.1
デバイスパスのある特別な USB デバイスクラス： UsbClass（0xFFFF、 0xFFFF、0xFF、 0xFF、0xFF）	システム内の任意の USB デバイス	1			Generic.USB.1.1
空のスロット、デバイスなし	スロット	スロット番号	1		PCI.Slot.2.1
Unknown device	内蔵 スロット	スロット番号または 1	インクリメント		Unknown.Slot.1.1 Unknown.Unknown.1.1

表 2 現在サポートされている構造化されたブート文字列の例 (続き)

デバイスタイプ	位置	インスタンス	サブインスタンス	修飾子	構造化されたブート文字列の例
	不明な位置				
NVMe	スロット	スロット番号	NVMe ドライブ番号（この番号はバス列挙のシーケンスに基づきます）。		NVMe.Slot.1.1
NVMe	内蔵	ベイ番号	1（各ドライブベイに 1 つの NVMe ドライブが存在します）。		NVMe.Emb.1.1

UEFI ブート順序の変更の例

BIOS の現在の構成オブジェクトには、UEFI ブート順序の現在の構成をリストする、タイプ `HpServerBootSettings` の個々の読み取り専用リソースへのリンクが含まれます。これは、システムが UEFI ブートモードで構成されているときのシステムブート順序です。UEFI ブート順序の現在の構成リソースには、UEFI ブートソースの配列である`BootSources` プロパティが含まれます。その配列内の各オブジェクトは、そのブートソースを特定する他のプロパティの中で一意な`StructuredBootString` を持ちます。

UEFI ブート順序リスト自体は、ブートソースの順序付けされた配列である、個々の `PersistentBootConfigOrder` プロパティで表されており、それぞれは、その `StructuredBootString` で参照されています。さらに、`DesiredBootDevices` プロパティは、`BootSources` プロパティにリストされない可能性がある目的のブートソースの、順序付けされた個々のリストを示します。これは、まだ構成されていない（および BIOS で検出された）可能性のある、SCSI、FC LUN、または iSCSI の特定ターゲットからの起動を構成するのに有効です。

BIOS の現在の構成リソースと同様、UEFI ブート順序の現在の構成リソースは読み取り専用です（許可ヘッダーで `PATCH` が許可された操作としてリストされていないことからもわかります）。UEFI ブート順序を変更するには、保留中の UEFI ブート順序設定が含まれる、`PATCH` 操作を実行できる個別の Settings リソースへのリンクをたどり、その Settings リソースの `PersistentBootConfigOrder` および（または）`DesiredBootDevices` プロパティを更新する必要があります。設定は次の再起動まで保留され、UEFI ブート順序の現在の構成リソースの`SettingsResults` プロパティに結果が反映されます。

前提条件

- 必要な最小セッション ID の権限：構成

UEFI ブート順序の変更の例

1. `/rest/v1/Systems` を繰り返して、メンバー`ComputerSystem` を選択します。`PATCH` 操作を許可しているタイプ`HpServerBootSettings` の子リソースを見つけます（この例題でないと複数存在する可能性がありますが、1 番目を選択します）。

   {ilo-ip-address}`/rest/v1/Systems/1/BIOS/Boot/Settings`

2. UEFI ブート順序を取得します。

   `GET` *{ilo-ip-address}*`/rest/v1/Systems/1/BIOS/Boot/Settings`

3. `PersistentBootConfigOrder` プロパティで新しい JSON オブジェクトを作成し、ブート順序を変更します。

UEFI ブート順序の変更の例

4. UEFI ブート順序を変更します。要求本体の更新済み`PersistentBootConfigOrder` プロパティを送信するだけです。

   `PATCH {ilo-ip-address}/rest/v1/Systems/1/BIOS/Boot/Settings`

サーバーがリセットされたときに、新しいブート順序が検証され、使用されます。

BIOS 管理者パスワードに関する留意事項

BIOS 管理者パスワードを有効にした場合、BIOS 設定の`PATCH` または`PUT` 操作に特別な情報を提供する必要があります。`PATCH` または`PUT` 操作は、以下の HTTP ヘッダーに追加する必要があります。

HTTP ヘッダー名	値
X-HPRESTFULAPI-AuthToken	管理者パスワードの SHA256 の 16 進ダイジェスト（大文字）からなる文字列。Python では、`hashlib.sha256(bios_password.encode()).hexdigest().upper()` です。

すべての BIOS とブート順序の設定の出荷時のデフォルト設定へのリセットの例

すべての BIOS とブート順序の設定の出荷時のデフォルト設定へのリセット

1. `/rest/v1/Systems` を繰り返して、メンバー`ComputerSystem` を選択します。`PATCH` 操作を許可しているタイプ`HpBios` の子リソースを見つけます（この例題でないと複数存在する可能性がありますが、1 番目を選択します）。

   `{ilo-ip-address}/rest/v1/Systems/1/BIOS/Settings`

2. BIOS とブート順序の設定を取得します。

   `GET {ilo-ip-address}/rest/v1/Systems/1/BIOS`

3. `RestoreManufacturingDefaults` プロパティで新しい JSON オブジェクトを作成し、値を yes に変更します。
4. BIOS とブート順序の設定をリセットします。要求本体の更新済み`RestoreManufacturingDefaults` プロパティを送信するだけです。

   `PATCH {ilo-ip-address}/rest/v1/Systems/1/BIOS`

電源

サーバーの電源制御は、シャーシレベルの制御ではなく、システムノードレベルのエンティティです。たとえば、マルチノードシャーシ内の 1 つのノードをオンにできます。電源を制御するには、コンピューターシステムのノードオブジェクトで HTTP 操作を実行します。コンピューターシステムのノードオブジェクトについて詳しくは、「コンピューターシステム」(19 ページ)を参照してください。

インターフェイスでのいくつかの操作は、正確な RESTful の `GET`、`PUT`、`POST`、`DELETE`、または `PATCH` ではありません。これらはカスタム操作と呼ばれ、特定の要求のペイロードを含む HTTP `POST` を使用して実行されます。通常、実行するアクションが、そのタイプで利用可能なプロパティで十分に表現されないときに、アクションが定義されます。たとえば、電源ボタンが読みにくいため、電源ボタンのステータスに `GET` を実行できません。この場合、電源ボタンを押すことがアクションです。

アクションは、実行するアクションの名前を示す `Action` プロパティ、およびゼロまたは複数のパラメーターのプロパティを持つ `POST` 操作です。

サーバーのリセットの例

Python：Python コードの例`ex2_reset_server()` を参照してください。

前提条件

- 必要な最小セッション ID の権限：構成

サーバーのリセットの例

1. `/rest/v1/Systems` を繰り返して、`POST` 操作を許容するメンバー`ComputerSystem` を選択します。

   {ilo-ip-address}`/rest/v1/Systems/1`
2. iLO に送信する Action オブジェクトを作成します。

   `{"Action":"Reset","ResetType":"ForceRestart"}`
3. `Action` および`ResetType` プロパティを`{"Action":"Reset","ResetType":"ForceRestart"}` に変更します。
4. サーバーをリセットします。

   `POST` *{ilo-ip-address}*`/rest/v1/Systems/1`

サーバーはリセットされ、再起動されます。

6 iLO RESTful API のエラーメッセージとレジストリ

エラーメッセージは、iLO RESTful API の複数の場面で表示されます。

- HTTP 操作にすぐに応答する場面。
- データモデルの`SettingsResult`。BIOS などの他のプロバイダーがある時点で設定を処理した場所、およびこのモデルのステータスを通信したい場所です。

すべてのエラーケースでは、`Type` が`ExtendedError.0.9.5` の`ExtendedError`（Redfish では`ExtendedInfo`）という基本的なエラーの JSON 構造を使用します。`ExtendedError` で最も重要なプロパティは、メッセージレジストリへの検索キーを含む文字列の`MessageID` です。

`MessageID` により、エラーについて説明する多くのテキストをコードから除くことで、iLO サービスを小さく保つことができます。実際には、iLO は`ExtendedError` 応答を提供しています。この応答では、`MessageID` が別のファイルから詳細情報を検索できるだけの情報を提供します。

たとえば、iLO ライセンスサービスに `POST` を実行して iLO ライセンスをインストールする場合に、誤ったキー文字列を指定すると、iLO の応答は以下のようなエラーが表示されます。

```
HTTP response 400
{
  "Type": "ExtendedError.0.9.5",
  "MessageID": "iLO.1.0.InvalidLicenseKey"
}
```

HTTP 応答 400 は、エラーへの標準的な RESTful API 応答です。上記の例は、分かりやすいエラーですが、エラーによっては理解しにくいものもあります。さらに意味のあるエラーメッセージを表示するには、次のコンポーネントの文字列 `iLO.0.9.InvalidLicenseKey` を解析します。

- `iLO.0.9`：参照するメッセージレジストリのベース名です。一致するレジストリファイルを探します。
- `InvalidLicenseKey`：メッセージレジストリの検索キーです。

この検索では、次のような結果が返されます。

```
"InvalidLicenseKey":{
  "Description": "The supplied license key is not valid.",
  "Message": "The supplied license key is not valid.",
  "Severity": "Warning",
  "NumberofArgs": 0,
  "ParamTypes": [],
  "Resolution": "Provide a valid license key."
}
```

多くのエラーメッセージは、パラメーターを返すこともできます。これらのパラメーターをレジストリ内の文字列に組み込んで、エラーメッセージのインスタンスに合わせて調整された詳細なメッセージを作成する場合があります。

Redfish：Redfish には代替エラー応答があります。Redfish 1.0 仕様を参照して、`ExtendedInfo` スキーマを調べてください。

7 クライアントの作成と正しくない前提の回避のためのベストプラクティス

RESTful API 用のクライアントを開発するときは、保証されていない前提に基づいてコードを記述しないようにしてください。システムとファームウェアのバージョンによって実装が異なる可能性があるため、またコードが一貫して動作するようにするために、このような前提を回避することが重要です。

API アーキテクチャー

RESTful API は、意図的にハイパーメディア API になっています。これは、将来のハードウェア実装への適合を難しくする、データモデルに対する限定的な前提を組み込まないようにするためです。ハイパーメディア API は、リソース間のリンク経由でデータモデルを検出可能にすることで、これらの前提を回避します。

クライアントは、静的なままの場合と同様に URI とやり取りすべきではありません。特定の最上位 URI のみ（このサンプルコードの任意の URI）、静的であると想定できます。

既知の最上位 URI を除き、すべての URI は、データモデルの href リンクをたどることで動的に検出される必要があります。クライアントはコレクションのリソースメンバーの URI に関して想定すべきではありません。たとえば、コレクションメンバーの URI は、必ずしも/rest/v1/.../collection/1 または 2 とは限りません。Moonshot では、システムのコレクションメンバーは/rest/v1/Systems/C1N1 である可能性があります。

データモデルを横断した URI の検索

データモデルのリソースはお互いにリンクしていますが、リソース間の相互リンク参照が原因で、クライアントはリソースモデルがツリーであると想定しない可能性があります。そうではなく、グラフなので、データモデルのすべてのクロールは、訪問済みのリソースを記録して、トラバーサルの無限ループを回避する必要があります。

別のリソースへの参照は、リソースのどこで発生する場合でも、`href`（Redfish では `@odata.id`）と呼ばれる任意のプロパティです。

データモデルの外部にあるリソースへの外部参照は、"extref"と呼ばれるプロパティで参照されます。extref によって参照されるリソースは、API の規約に従っていると想定すべきではありません。

タイプ名とバージョン

各リソースには`Type` プロパティがあり、その値の形式は`Typename.x.y.z` です。

- x = メジャーバージョン。スキーマへの大きな変更によってインクリメント。
- y = マイナーバージョン。スキーマへの（大きな変更ではなく）付加的な変更によってインクリメント。
- z = エラータ。大きくない変更 。たとえば代替の説明テキスト、スペル修正などです。

HTTP POST での新しいリソースの作成

POST 操作でリソースを作成すると（たとえば、アカウントまたはセッションの作成）、正常な応答に、新しく作成したリソースのリソース URI を示す`Location` HTTP ヘッダーが含まれます。POST には、JSON 応答本体に、新しく作成したオブジェクトの表現が含まれる場合と、含まれない場合があります。応答本体を想定しないで、それをテストしてください。`ExtendedError` オブジェクトの場合もあります。

HTTP リダイレクト

すべてのクライアントが HTTP リダイレクト（308、301 など）を正しく処理する必要があります。iLO 4 はデータモデルの一部のエイリアスの手段として、またデータモデルを Redfish 指定の URI（たとえば、`/redfish/...`）に移行する手段として、リダイレクションを使用します。

将来：非同期タスク

将来、一部の操作は非同期タスクを開始する可能性があります。この場合、クライアントは、必要に応じて HTTP 202 を認識および処理する必要があります。`Location` ヘッダーはタスク情報とステータスを持つリソースを指し示します。

スキーマと実装

`/rest/v1/Schemas` で入手できる JSON スキーマは、リソースの内容を制御しますが、以下の点に注意してください。

- スキーマのすべてのプロパティがすべての実装で実装されるとは限りません。
- 一部のプロパティは、null と別の型（文字列型、整数型など）の両方を許可するように設計されています。

堅牢なクライアントコードは、関心のあるプロパティの存在と型の両方を確認し、期待値を満たしていない場合、正しい手順で失敗する必要があります。

クライアントに関する追加情報

クライアントは、以下の項目について常に準備する必要があります。

- 実装されていないプロパティ（たとえば、プロパティは特定のケースで適用されません）
- プロパティの値がシステム状態が原因で現在不明な場合に、場合によっては null 値
- 200 OK 以外の HTTP ステータスコード。その他の情報がない HTTP 500 Internal Server Error にコードが対応できるか。
- URI は大文字と小文字が区別されません。
- HTTP ヘッダー名は大文字と小文字が区別されません。
- JSON プロパティおよび Enum 値は大文字と小文字が区別されます。
- クライアントはサービスが返す HTTP ヘッダーの任意のセットに寛容である必要があります。

8 RESTful イベントとイベントサービス

iLO では、iLO 4 2.30 以降、REST データの変更時、または特定のアラートの発生時に通知を受信するよう登録できる新しいイベントサブスクリプションサービスが導入されました。これらの通知は、選択した URI への HTTPS POST 操作という形式を取ります。

イベントサービスは、/rest/v1/EventService にあるデータモデルに配置されます。このリソースには、サブスクリプションのコレクション（EventSubscriptions と呼ばれ、/rest/v1/EventService/EventSubscriptions に配置されます）へのリンクが含まれています。

イベントの登録の例

イベントを受信するには、iLO のネットワークにアクセス可能な HTTPS サーバーを URI で指定する必要があります。この URI は、iLO によって開始される HTTPS POST 操作のターゲットとして指定します。

タイプListenerDestination に準拠した（例を参照）JSON オブジェクトを作成し、これを/rest/v1/EventService/EventSubscriptions のEventSubscriptions リンクによって示されるコレクションに POST します。HTTP 201 Created 応答を受信した場合は、新しいサブスクリプションが追加されています。iLO はこの段階ではあて先 URI をテストしません。そのため、指定された URI が無効な場合は、イベントが発生して宛先への接続が失敗するまでフラグ付けされません。

例 1 EventSubscriptions コレクションへの POST ペイロードの例（ListenerDestination）

```
{
  "Destination": "https://myeventreciever/eventreceiver",
  "EventTypes": [
    "ResourceAdded",
    "ResourceRemoved",
    "ResourceUpdated",
    "StatusChange",
    "Alert"
  ],
  "HttpHeaders": {
    "Header”:”HeaderValue"
  },
  "TTLCount": 1440,
  "TTLUnits": "minutes",
  "Context": "context string",
  "Oem": {
    "Hp": {
      "DeliveryRetryIntervalInSeconds": 30,
      "RequestedMaxEventsToQueue": 20,
      "DeliveryRetryAttempts": 5,
      "RetireOldEventInMinutes": 10
    }
  }
}
```

上記の内容の多くは、ユーザーのニーズとセットアップによってまったく異なります。

- **Destination** は、iLO のネットワークにアクセス可能な HTTPS URI である必要があります。
- この例の**EventTypes** にはすべてが含められていますが、配列からタイプを削除することもできます。
- **HttpHeaders** には、イベント POST 操作に必要な任意の HTTP ヘッダーを指定できます。サブスクリプションは、承認済み iLO ユーザーへの GET を介して読み取ることができます。

- **Context** には、任意の文字列を指定できます。

各プロパティについて詳しくは、**ListenerDestination** スキーマを参照してください。指定された TTL 情報の後、サブスクリプションは自動的に有効期限が切れるため、更新する必要があります。

9 トラブルシューティング

サードパーティの REST Web クライアントを使用した RESTful API のリセット

症状

ProLiant Gen9 サーバーでは、システムのブート時に RESTful API エラーを検出する可能性があり、RESTful API を使用して BIOS 設定を構成できなくなります。さらに、次の一貫性のあるエラーメッセージが、システムのブート（POST）中に表示される可能性があり、Integrated Management Log に記録されます。

```
335 RESTful API Error- RESTful API PUT request failed (HTTP: Status
Code = 404)
```

iLO ファームウェア v2.20 以降では、REST API をリセットできます。サードパーティの REST Web クライアント、RESTful インターフェイスツール、または、HPE 内蔵 UEFI シェルの `restclient` コマンドを使用して、RESTful API を介してこれを行います。

操作

1. リクエスト本体に次の JSON を付けて、URI`<ilo-ip>/rest/v1/managers/1` のリソースへの POST 操作を実行します。

   ```
   --------------- Copy Start ----------------
   {
       "Action":"ClearRestApiState",
       "Target":"/Oem/Hp"
   }
   ---------------- Copy End -----------------
   ```

2. サーバーを再起動します。

RESTful インターフェイスツールを使用した iLO RESTful API のリセット

症状

ProLiant Gen9 サーバーでは、システムのブート時に RESTful API エラーを検出する可能性があり、RESTful API を使用して BIOS 設定を構成できなくなります。さらに、次の一貫性のあるエラーメッセージが、システムのブート（POST）中に表示される可能性があり、Integrated Management Log に記録されます。

```
335 RESTful API Error- RESTful API PUT request failed (HTTP: Status
Code = 404)
```

iLO ファームウェア v2.20 以降では、REST API をリセットできます。サードパーティの REST Web クライアント、RESTful インターフェイスツール、または、HPE 内蔵 UEFI シェルの `restclient` コマンドを使用して、RESTful API を介してこれを行います。

原因

操作

1. RESTful インターフェイスツールをダウンロードし、インストールします。このツールの使用について詳しくは、**http://www.hpe.com/info/resttool**（英語）を参照してください。
2. テキストファイルに次の JSON をコピーして貼り付け、`hprest_tool_clear_api.json` として保存します。

   ```
   --------------- Copy Start ----------------
   {
       "path": "/rest/v1/managers/1",
   ```

```
    "body": {
        "Action":"ClearRestApiState",
        "Target":"/Oem/Hp"
    }
}
--------------- Copy End -----------------
```

3. hprest ツールを起動します。

   ```
   hprest
   ```

4. iLO にログインします。

   ```
   hprest> login <ilo-ip>
   ```

5. hprest_tool_clear_api.json ファイルを指す、次のコマンドを実行します。

   ```
   hprest> rawpost hprest_tool_clear_api.json
   ```

6. サーバーを再起動します。

内蔵 UEFI シェル restclient コマンドを使用した iLO RESTful API のリセット

症状

ProLiant Gen9 サーバーでは、システムのブート時に RESTful API エラーを検出する可能性があり、RESTful API を使用して BIOS 設定を構成できなくなります。さらに、次の一貫性のあるエラーメッセージが、システムのブート（POST）中に表示される可能性があり、Integrated Management Log に記録されます。

```
335 RESTful API Error- RESTful API PUT request failed (HTTP: Status
Code = 404)
```

iLO ファームウェア v2.20 以降では、REST API をリセットできます。サードパーティの REST Web クライアント、RESTful インターフェイスツール、または、HPE 内蔵 UEFI シェルの restclient コマンドを使用して、RESTful API を介してこれを行います。

原因

操作

1. 拡張した内蔵 UEFI シェルを入力します。詳しくは、**http://www.hpe.com/servers/proliant/uefi**（英語）で『UEFI シェルユーザーガイド』を参照してください。
2. 次の JSON を ASCII テキストファイルにコピーして貼り付け、FAT フォーマットの USB メディア上に clear_api.json として保存します。

   ```
   --------------- Copy Start ----------------
   {
     "Action":"ClearRestApiState",
     "Target":"/Oem/Hp"
   }
   --------------- Copy End ------------------
   ```

3. USB メディアをサーバーに接続します。
4. サーバーの電源を入れ、内蔵 UEFI シェルから起動します。
5. UEFI シェルプロンプトには、USB メディアに対応するファイルシステムを検索する partitions コマンドを使用します。例：FS0、FS1 など。
6. ファイルシステムに切り替えるには、ファイルシステムの名前を入力します（たとえば、shell>FS0:など）。
7. 次のコマンドを実行します。

   ```
   Fs0:> restclient –m POST –uri “/rest/v1/managers/1” –i clear_api.json
   ```

8. サーバーを再起動します。

iLO SSH CLI を使用した iLO RESTful API のリセット

症状

ProLiant Gen9 サーバーでは、システムのブート時に RESTful API エラーを検出する可能性があり、RESTful API を使用して BIOS 設定を構成できなくなります。さらに、次の一貫性のあるエラーメッセージが、システムのブート（POST）中に表示される可能性があり、Integrated Management Log に記録されます。

```
335 RESTful API Error- RESTful API PUT request failed (HTTP: Status
Code = 404)
```

iLO ファームウェア v2.20 以降では、REST API をリセットできます。サードパーティの REST Web クライアント、RESTful インターフェイスツール、または、HPE 内蔵 UEFI シェルの `restclient` コマンドを使用して、RESTful API を介してこれを行います。

原因

操作

1. iLO で SSH 接続を開いて、管理者権限を持つアカウントを使用してログインします。詳しくは、**http://www.hpe.com/jp/servers/ilo** にある『HPE iLO 4 スクリプティング/コマンドラインガイド』を参照してください。
2. CLI プロンプトで、`oemhp_clearRESTAPIstate` コマンドを実行します。このコマンドが完了するまでに数秒かかる場合があることに注意してください。
3. サーバーを再起動します。

Alert EventType への登録時の、JSON の異常な文字

症状

Alert EventType に登録するとき、JSON ファイルの最後に異常あるいは余分な文字が含まれます。

原因

進行中のイベントリスナーによる解析が、ヘッダーの`Content-length` 値を超えています。

操作

10 Gen9 サーバー用 iLO RESTful API 機能

次の表は、iLO 4 2.30 を実行している Gen8 サーバーと比較して、iLO 4 2.30 を実行している Gen9 サーバーで利用可能な追加の REST API の機能を示します。

# UEFI 拡張メモリの詳細	
/rest/v1/Systems/{item}/Memory/{item}	HpMemory.1.0.0 #/Manufacturer
# UEFI 拡張 PCI デバイスの詳細	
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/ClassCode
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/DeviceID
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/DeviceInstance
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/DeviceSubInstance
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/SegmentNumber
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/StructuredName
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/SubclassCode
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/SubsystemDeviceID
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/SubsystemVendorID
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/UEFIDevicePath
/rest/v1/Systems/{item}/PCIDevices/{item}	HpServerPciDevice.1.0.0 #/VendorID
# UEFI PCI スロットの詳細	
/rest/v1/Systems/{item}/PCISlots/{item}	HpServerPCISlot.1.0.0 #/UEFIDevicePath
# BIOS 関連のリソース	
/rest/v1/Systems/{item}/SecureBoot	HpSecureBoot.1.0.0
/rest/v1/Systems/{item}/bios	HpBios.1.2.0
/rest/v1/Systems/{item}/bios/Settings	HpBios.1.2.0
/rest/v1/Systems/{item}/bios/iScsi/Settings	HpiSCSISoftwareInitiator.1.1.0
/rest/v1/Systems/{item}/bios/Mappings	HpBiosMapping.1.2.0
/rest/v1/Systems/{item}/bios/iScsi/BaseConfigs	HpBaseConfigs.0.10.0
/rest/v1/Systems/{item}/bios/iScsi	HpiSCSISoftwareInitiator.1.1.0
/rest/v1/Systems/{item}/bios/Boot/Settings	HpServerBootSettings.1.2.0
/rest/v1/Systems/{item}/bios/BaseConfigs	HpBaseConfigs.0.10.0
# PCI デバイスのファームウェアバージョン	
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/{PCIDeviceID}[]/ImageSizeBytes
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/{PCIDeviceID}[]/Key
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/{PCIDeviceID}[]/Location
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/{PCIDeviceID}[]/Updateable
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/{PCIDeviceID}[]/VersionString

# Gen9 の新しいファームウェアコンポーネント	
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/SASProgrammableLogicDevice[]/Location
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/SASProgrammableLogicDevice[]/VersionString
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/StorageBattery[]/Location
/rest/v1/Systems/{item}/FirmwareInventory	FwSwVersionInventory.1.0.0 #/Current/StorageBattery[]/VersionString
/rest/v1/Chassis/{item}	HpServerChassis.1.0.0 #/Oem/Hp/Firmware/SASProgrammableLogicDevice/Current/VersionString

11 サポートと他のリソース

Hewlett Packard Enterprise サポートへのアクセス

- ライブアシスタンスを受けるには、Web サイト「Contact Hewlett Packard Enterprise Worldwide」に移動します。

  http://www.hpe.com/assistance

- ドキュメントとサポートサービスにアクセスするには、Hewlett Packard Enterprise サポートセンターの Web サイトに移動します。

  http://www.hpe.com/support/hpesc

ご用意いただく情報

- テクニカルサポートの登録番号（該当する場合）
- 製品名、モデルまたはバージョン、シリアル番号
- オペレーティングシステム名およびバージョン
- ファームウェアバージョン
- エラーメッセージ
- 製品固有のレポートおよびログ
- 増設した製品またはコンポーネント
- 他社製品またはコンポーネント

アップデートへのアクセス

- 一部のソフトウェア製品では、その製品のインターフェイスを介してソフトウェアアップデートにアクセスするためのメカニズムが提供されます。製品のドキュメントを確認し、推奨されるソフトウェアアップデートの方法を特定します。
- 製品のアップデートをダウンロードするには、以下のいずれかに移動します。
  - Hewlett Packard Enterprise サポートセンターの **[メールニュース配信登録]**ページ：

    http://www.hpe.com/support/e-updates-ja

  - Software Depot の Web サイト：

    http://www.hpe.com/support/softwaredepot

- お客様の資格を表示したりアップデートしたり、契約や保証をお客様のプロファイルにリンクしたりするには、Hewlett Packard Enterprise サポートセンターの **[More Information on Access to Support Materials]** ページに移動します。

  http://www.hpe.com/support/AccessToSupportMaterials

重要: 一部のアップデートにアクセスするには、Hewlett Packard Enterprise サポートセンターからアクセスするときに製品の資格が必要になる場合があります。関連する資格を使って HP パスポートをセットアップしておく必要があります。

Web サイトおよびドキュメント

表 3 Web サイト

Web サイト	リンク
Hewlett Packard Enterprise Information Library	**http://www.hpe.com/info/enterprise/docs**
UEFI	**http://www.hpe.com/info/ProLiantUEFI/docs**
HPE Service Pack for ProLiant	**http://www.hpe.com/servers/spp/documentation**
HPE iLO 4	**http://www.hpe.com/info/ilo/docs**
HPE iLO University ビデオ	**http://www.hpe.com/info/ilo/videos**（英語）
HPE Systems Insight Manager	**http://www.hpe.com/jp/hpsim**
HPE Onboard Administrator	**http://www.hpe.com/info/oa**（英語）
HPE VMware Vibs Depot	**http://vibsdepot.hpe.com/**（英語）
Hewlett Packard Enterprise Information Library	**http://www.hpe.com/info/enterprise/docs**
Hewlett Packard Enterprise サポートセンター	**http://www.hpe.com/support/hpesc**
Contact Hewlett Packard Enterprise Worldwide	**http://www.hpe.com/assistance**
サブスクリプションサービス/サポートのアラート	**http://www.hpe.com/support/e-updates-ja**
Software Depot	**http://www.hpe.com/support/softwaredepot**
カスタマーセルフリペア	**http://www.hpe.com/support/selfrepair**
Insight Remote Support	**http://www.hpe.com/info/insightremotesupport/docs**
HP-UX 用の Serviceguard ソリューション	**http://www.hpe.com/info/hpux-serviceguard-docs**
Single Point of Connectivity Knowledge（SPOCK）のストレージ互換性マトリックス	**http://www.hpe.com/storage/spock**（英語）
ストレージのホワイトペーパーおよび分析レポート	**http://www.hpe.com/storage/whitepapers**

表 4 ドキュメント

ドキュメント
『iLO RESTful API Data Model Reference』
『HPE iLO 4 スクリプティング/コマンドラインガイド』
『HPE iLO 4 リリースノート』
『UEFI システムユーティリティユーザーガイド』
『HPE ProLiant Gen9 トラブルシューティングガイド、ボリューム I：トラブルシューティング』
『HPE iLO 連携ユーザーガイド 』
『HPE Service Pack for ProLiant クイックスタートガイド』

カスタマーセルフリペア

Hewlett Packard Enterprise カスタマーセルフリペア（CSR）プログラムでは、ご使用の製品をお客様ご自身で修理することができます。CSR 部品を交換する必要がある場合、お客様のご都合のよいときに交換できるよう直接配送されます。一部の部品は CSR の対象になりません。Hewlett Packard Enterprise もしくはその正規保守代理店が、CSR によって修理可能かどうかを判断します。

リモートサポート（HPE 通報サービス）

リモートサポートは、保証またはサポート契約の一部としてサポートデバイスでご利用いただけます。リモートサポートは、インテリジェントなイベント診断を提供し、ハードウェアイベントを Hewlett Packard Enterprise に安全な方法で自動通知します。これにより、ご使用の製品のサービスレベルに基づいて、迅速かつ正確な解決が行われます。ご使用のデバイスをリモートサポートに登録することを強くおすすめします。

デバイスサポートについて詳しくは、次の Web サイトを参照してください。

http://www.hpe.com/info/insightremotesupport/docs

A 保証および規制に関する情報

安全、環境、および規定に関する情報については、『サーバー、ストレージ、電源、ネットワーク、およびラック製品の安全と準拠に関する情報』（**http://www.hpe.com/support/Safety-Compliance-EnterpriseProducts**）を参照してください。

保証情報

HPE ProLiant と x86 サーバーおよびオプション

http://www.hpe.com/support/ProLiantServers-Warranties

HPE エンタープライズサーバー

http://www.hpe.com/support/EnterpriseServers-Warranties

HPE ストレージ製品

http://www.hpe.com/support/Storage-Warranties

HPE ネットワーク製品

http://www.hpe.com/support/Networking-Warranties

規定に関する情報

Belarus Kazakhstan Russia marking

EAC

Manufacturer and Local Representative Information

Manufacturer information:

- Hewlett Packard Enterprise Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.

Local representative information Russian:

- **Russia**:

  ООО «Хьюлетт Паккард Энтерпрайз», Российская Федерация, 125171, г. Москва, Ленинградское шоссе, 16А, стр.3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  ИООО «Хьюлетт-Паккард Бел», Республика Беларусь, 220030, г. Минск, ул. Интернациональная, 36-1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ТОО «Хьюлетт-Паккард (К)», Республика Казахстан, 050040, г. Алматы, Бостандыкский район, проспект Аль-Фараби, 77/7, Телефон/факс: + 7 727 355 35 52

Local representative information Kazakh:

- **Russia**:

  ЖШС "Хьюлетт Паккард Энтерпрайз", Ресей Федерациясы, 125171,
  Мәскеу, Ленинград тас жолы, 16А блок 3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  «HEWLETT-PACKARD Bel» ЖШС, Беларусь Республикасы, 220030, Минск қ.,
  Интернациональная көшесі, 36/1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ЖШС «Хьюлетт-Паккард (К)», Қазақстан Республикасы, 050040, Алматы к.,
  Бостандық ауданы, Әл-Фараби даңғ ылы, 77/7, Телефон/факс: +7 727 355 35 52

Manufacturing date:

The manufacturing date is defined by the serial number.

CCSYWWZZZZ (serial number format for this product)

Valid date formats include:

- YWW, where Y indicates the year counting from within each new decade, with 2000 as the starting point; for example, 238: 2 for 2002 and 38 for the week of September 9. In addition, 2010 is indicated by 0, 2011 by 1, 2012 by 2, 2013 by 3, and so forth.
- YYWW, where YY indicates the year, using a base year of 2000; for example, 0238: 02 for 2002 and 38 for the week of September 9.

Turkey RoHS material content declaration

Türkiye Cumhuriyeti: EEE Yönetmeliğine Uygundur

Ukraine RoHS material content declaration

Обладнання відповідає вимогам Технічного регламенту щодо обмеження використання деяких небезпечних речовин в електричному та електронному обладнанні, затвердженого постановою Кабінету Міністрів України від 3 грудня 2008 № 1057