インターシステムズ製品における FHIR のサポート

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

インターシステムズ製品における FHIR のサポート
InterSystems Version 2023.1 2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

目次

テーブル一覧

1

インターシステムズの FHIR コンポーネント

インターシステムズ製品には、以下の FHIR® テクノロジが搭載されています。

FHIR サーバ

FHIR サーバは、FHIR 要求を受信および処理する一方、FHIR リソースを内部リポジトリに格納したり、リポジトリから取得できるアーキテクチャを活用しています。インターシステムズ製品では、FHIR サーバ向けのすぐに使用可能なソリューションによってリソース・リポジトリがストレージとして使用されます。製品のライセンスによっては、FHIR サーバをリソース・リポジトリと共にインストールできないことがあります。その場合は、FHIR 相互運用アダプタを使用して、FHIR 要求を受信し、処理する必要があります。FHIR サーバを使用すると、サーバの内部アーキテクチャに到達するまで、要求を相互運用プロダクションを通じてルーティングできますが、これは必須ではありません。相互運用プロダクションを使用しない FHIR サーバの方が、はるかに高速になる可能性があります。

FHIR 相互運用アダプタ

アプリケーションで FHIR 要求を受信し、処理する必要があるものの、リソースを内部ストレージに格納したり、内部ストレージから取得する必要がない場合は、FHIR サーバではなく、FHIR 相互運用アダプタを使用するのが最適な方法です。FHIR 相互運用アダプタは、FHIR サーバの内部アーキテクチャをインストールせずに、FHIR 要求の処理に必要なコンポーネントをインストールします。FHIR 相互運用アダプタは、常に相互運用プロダクションを使用して要求を処理します。

変換

インターシステムズ製品では、相互運用プロダクションから、または ObjectScript アプリケーションから直接呼び出すことができる事前定義済みの一連の変換を使用して、HL7v2 などの FHIR 以外の標準で捕捉された医療データを FHIR に変換するために使用できます。FHIR を入力として取り、別の相互運用性標準に変える変換も、用意されています。これらの変換の中核となるのは、FHIR とインターシステムズの臨床データ形式である SDA との間での変換機能です。

FHIR クライアント

インターシステムズのテクノロジでは、FHIR クライアントは相互運用ビジネス・ホストまたは ObjectScript アプリケーションです。ObjectScript アプリケーションは、外部 FHIR サーバのエンドポイントなのか、同じインターシステムズ製品内の FHIR アーキテクチャなのかに関係なく、FHIR エンドポイントへの要求を行います。FHIR クライアント・クラスは、FHIR サーバ上で FHIR 相互作用やオペレーションを実行するための簡単なメソッドを提供します。

Amazon HealthLake アダプタ

インターシステムズ製品には、相互運用プロダクションから Amazon HealthLake データ・ストア内の FHIR リソースを取得、作成、削除、および更新できる受信アダプタと送信アダプタが付属します。

FHIRPath

FHIRPath は、簡単な構文を使用して、FHIR リソースをナビゲートし、データを評価したり、そのフィールドから抽出できるようにする言語です。インターシステムズ製品では、リソースの評価に使用できる FHIRPath の関数と演算のサブセットを用意しています。

バルク FHIR コーディネータ

インターシステムズの製品には、HL7® FHIR® バルク・データ要求でクライアントと FHIR エンドポイントとの対話処理を仲介するバルク FHIR コーディネータが用意されています。一連の構成を入力できます。構成のそれぞれで FHIR エンドポイントを特定し、承認のタイプやファイルの場所など、バルク FHIR 相互作用で使用する各種パラメータを定義します。

2

FHIR について

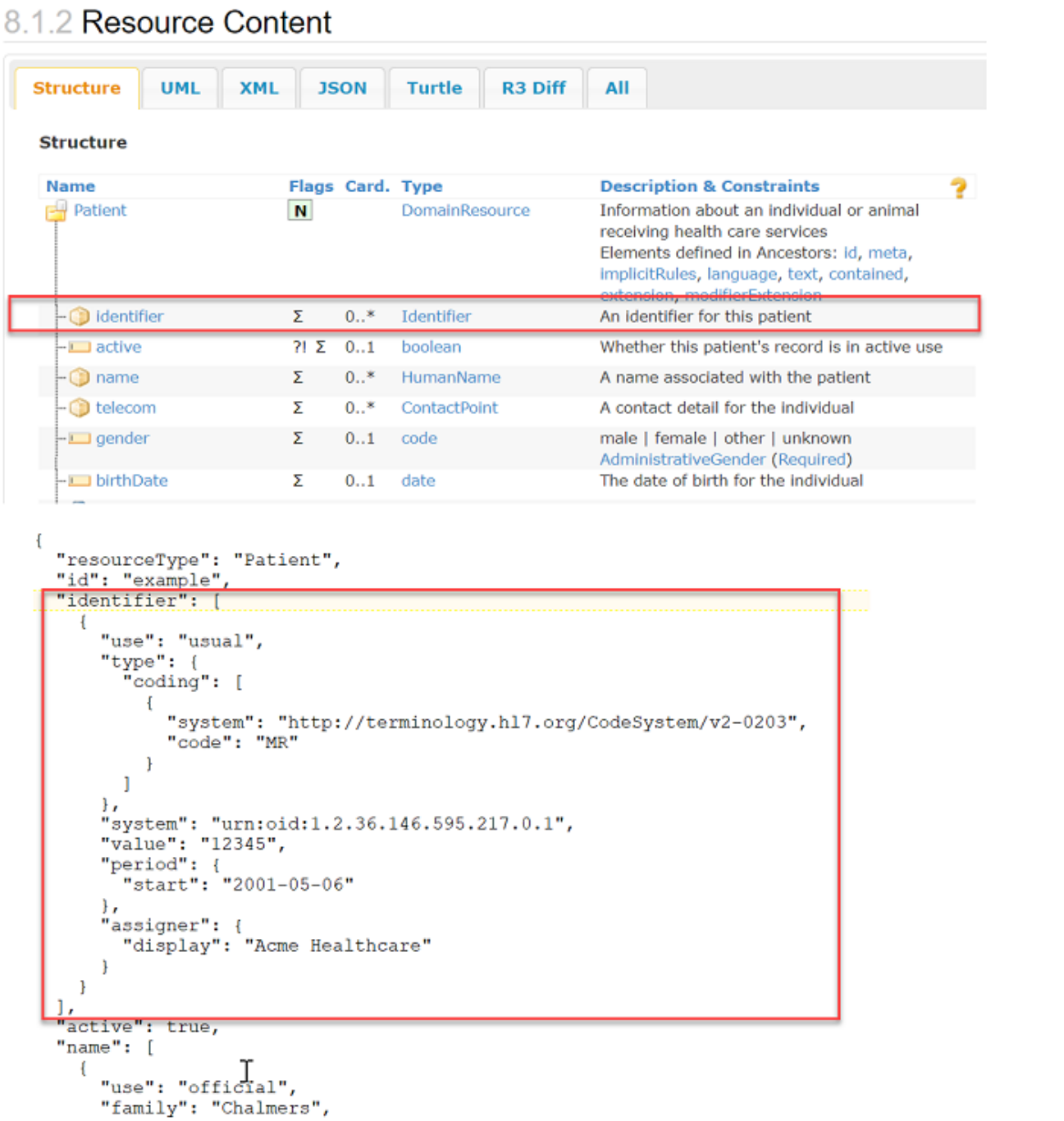
FHIR® (Fast Healthcare Interoperability Resources) は、HL7 の医療情報の相互運用性標準であり、合意されたデータ・モデルを使用して、複数のシステム間での医療情報の交換を可能にします。FHIR のこれらのデータ・モデルは単純でわかりやすく、人とコンピュータのどちらにとっても読み取り可能です。また、これらを組み合わせることにより堅牢となり、複雑な医療情報の伝達にも対応できます。

以下に、FHIR の重要な概念について簡単に説明します。これらの概念の詳細は、正式な FHIR 仕様を参照してください。

2.1 FHIR リソース

FHIR はリソースの概念をベースに構築されています。このリソースとは、JSON や XML として表現されるデータの離散単位です。例えば、1 人の患者に関するすべてのデータを Patient リソースとしてカプセル化する一方で、ある医師の往診情報を Encounter リソースで取得できます。この Encounter リソースには、通常、該当の医師が診察した患者の Patient リソースへの参照が含まれるため、患者のデータを Encounter リソース自体に含める必要はありません。リソースは、RESTful API を使用して個別に格納および取得できるため、FHIR が必要とする帯域幅や計算リソースは、他の相互運用性標準と比べて少なくて済みます。リソースを JSON として表現できるため、FHIR データの変換をより一層軽量にできます。

基本の FHIR 仕様には、サポート対象のすべてのリソースのページが用意されています。例えば、最新バージョンの FHIR の Patient リソースは、hl7.org/fhir/patient.html にあります。リソースに属しているデータ・フィールドやそれらのフィールドのデータ型など、リソースに関する主な情報は、仕様ページの Resource Content セクションに記載されています。このセクションには、各リソース・フィールドについて説明する [Structure] タブが含まれています。FHIR リソースから始める場合は、特定のリソース例をこの構造と比較すると便利です (サンプルのリソースは、仕様内の各リソース・ページの [Examples] タブで入手できます)。Patient リソースの構造の一部は次のようになります。



リソースの [Structure] タブで使用される記号およびアイコンの説明は、"Resource Formats" を参照してください。

FHIR はまた、標準の要素自体の定義にもリソースを使用します。Conformance リソースとして知られるこのメタデータは、リソースの有効なフィールド、FHIR サーバからリソースを取得するのに使用できる検索パラメータ、特定の医療環境内で使用されるコードなどを定義します。

基本の FHIR 仕様に現在記載されているリソースのリストは、"Resource Index" を参照してください。

2.2 FHIR の適応

FHIR は、特定の医療環境や実装に適応することを目的としており、この目的を達成するために、FHIR 標準を拡張したり制約する簡単な方法を提供します。FHIR は 80/20 ルールに従っているとよく言われます。つまり、基本の FHIR 仕様では、医療環境におけるニーズがその 80% を、カスタムの制約や拡張が残りの 20% を占めています。多くの場合、FHIR サーバは標準の公開済みの "実装ガイド" に準拠します。実装ガイドには、特定のエコシステムに関する FHIR の完全な実装が説明されています。例えば、US Core Implementation Guide では、米国の医療環境で FHIR を使用するための標準が定められています。もちろん、医療環境は、独自のニーズを満たすため、基本の FHIR 仕様、US Core Implementation Guide、またはその他の実装ガイドを拡張することができます。

FHIR の適応の中心には、FHIR プロファイルがあり、これにより、特定のリソースが拡張または制約されます。例えば、US Core Implementation Guide には、Patient リソースの独自のプロファイルや、Observation リソースの別のプロファイルなどが含まれています。技術レベルでは、各プロファイルは StructureDefinition Conformance リソースによって定義されます。FHIR 仕様によると、"プロファイリング" という用語は、これらの StructureDefinition を使用して特定の実装向けにリソースを構成する行為に使用されます。

FHIR の適応には、リソース・プロファイル以上のものを含めることができます。例えば、実装ガイドには、医療環境に固有のコードや検索パラメータを含めることができます。プロファイルと同様、これらのアセットは、ValueSet や SearchParameter などの Conformance リソースを使用して定義されます。

プロファイルおよびその他の Conformance リソースの一貫したコレクションは、FHIR パッケージとして知られています。パッケージの内容は大きく異なる可能性があります。実装ガイド全体を含めることも、単一のカスタム・プロファイルのみを含めることもできます。インターシステムズ製品では、パッケージを FHIR エンドポイントに追加することにより、特定の医療エコシステムをサポートするよう FHIR サーバを構成できます。

2.3 RESTful API

FHIR は、従来の医療用途の相互運用性標準のようなメッセージング・フレームワークやドキュメントベースのフレームワークで使用できますが、革新的なのは、RESTful API 呼び出しを使用して医療データを使用できることです。FHIR クライアントは、GET や POST などの HTTP 動詞を使用することで、必要に応じて FHIR リソースを格納、削除、更新、および取得できます。FHIR クライアントがリソースに対して実行できるこれらのアクションは、相互作用として知られています。RESTful API とサポートされている相互作用の詳細は、FHIR 仕様の "RESTful API" を参照してください。

FHIR ではまた、FHIR クライアントが各種オペレーションを使用して、FHIR サーバで機能を実行することもできます。クライアントはサーバ上で機能を呼び出すため、このようなオペレーションは RESTful 呼び出しというよりも RPC 呼び出しに似ています。例えば、標準の `$validate` オペレーションは、リソースがプロファイルに準拠しているかどうかを確認する機能をサーバ上で呼び出します。医療環境では、カスタムのオペレーションを実装して、FHIR クライアントの要求に応じて、多様なアクションを実行できます。

2.4 FHIR リソースの検索

検索は非常に強力な FHIR 相互作用です。医療データは個々のリソースとして保存されるため、FHIR クライアントは複雑なクエリを使用して、無関係のデータを解析することなく、必要なデータのみを取得できます。これらのクエリは GET HTTP 動詞を使用して実行されます。また、検索パラメータを使用して、特定の条件に合うリソースに結果を絞り込むことができます。最も簡単な方法では、検索パラメータを指定せずに検索を実行し、特定のタイプのすべてのリソースを取得できます。例えば、以下の RESTful API 呼び出しでは、すべての Patient リソースが取得されます。

```
GET http://myFHIREndpointURL/Patient
```

? 文字を使用して、API 呼び出しに検索パラメータを追加できます。例えば、検索で name 検索パラメータを使用して、名前フィールドに指定の値を持つ Patient リソースを検索できます。このような Patient リソースを取得する API 呼び出しは次のようになります。

```
GET http://myFHIREndpointURL/Patient?name=Smith
```

& 文字を使用して、複数の検索パラメータを連鎖させることができます。例えば、以下の API 呼び出しでは、患者の性別を追加することにより、結果をさらに絞り込めます。

```
GET http://myFHIREndpointURL/Patient?name=Smith&gender=male
```

FHIR 仕様には、強力かつ複雑なクエリの実行に使用できるその他の多くの標準検索パラメータが含まれます。詳細は、FHIR 仕様の ″Search″ を参照してください。特定のリソースの検索パラメータは、仕様のリソースのページで確認できます。

3

FHIR サーバ:概要

多くの実装では、すぐに使用可能な FHIR® サーバを使用できます。このサーバは、リソース・リポジトリを使用して、リソースをインターシステムズのデータベースに格納したり、このデータベースから取得します。この FHIR サーバは、相互運用プロダクションを使用することも、まったく新しいバックエンドを開発することもなく、カスタマイズできます。リソース・リポジトリを使用するサーバでサポートされる FHIR 相互作用の詳細は、"サポートされる相互作用とオペレーション" を参照してください。

FHIR 要求は、サーバのインフラストラクチャに到達するまで、相互運用プロダクションを通じてルーティングできますが、これは必須ではありません。相互運用プロダクションを使用しない FHIR サーバの方が、はるかに高速になる可能性があります。

あまりよくあることではありませんが、完全にカスタムのバックエンドを持つ FHIR サーバを構築することもできます。この実装では、リソース・リポジトリで使用されるものと同じ内部アーキテクチャを活用しますが、独自の FHIR 処理ロジックを開発する必要があります。

ご使用のインターシステムズ製品に FHIR サーバをインストールするライセンスがない場合は、FHIR 相互運用アダプタを使用して、相互運用プロダクション経由で FHIR 要求を受信および処理できます。

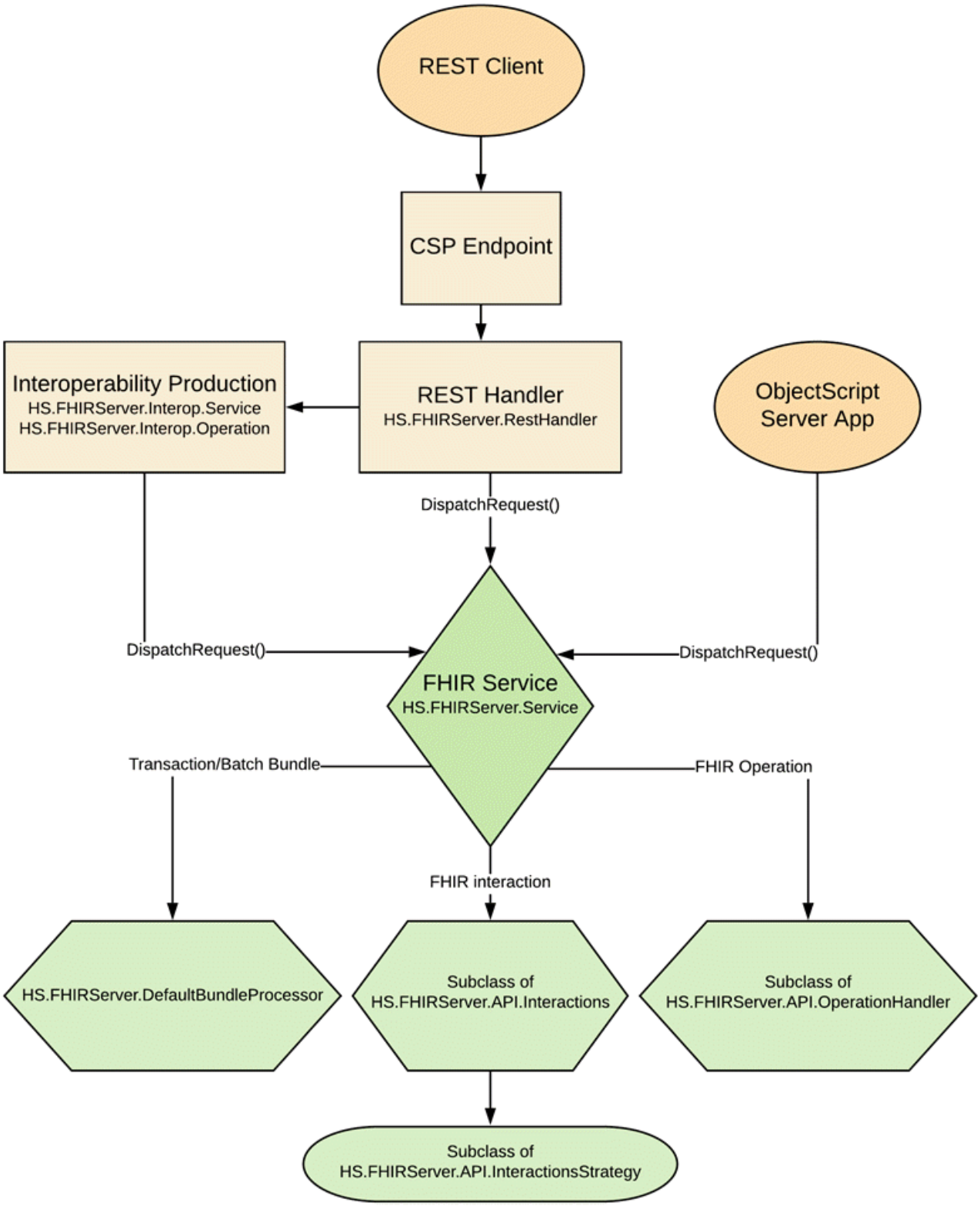
3.1 アーキテクチャ

リソース・リポジトリを使用する FHIR サーバとカスタムのバックエンドを使用する FHIR サーバは、同じアーキテクチャを使用します。FHIR サーバを通じて FHIR 要求をトレースすると、このサーバのアーキテクチャに関する主要機能の概要を適切に理解することができます。最初に、FHIR 要求はサービスに到達する必要があります。サービスは、要求がサーバの FHIR メタデータ標準に適合していることを確認してから、要求の処理に適したコンポーネントに要求をルーティングします。FHIR 要求は、このサービスに 3 つの方法で到達できます。REST ハンドラから到達する方法、相互運用プロダクション経由で到達する方法、または ObjectScript FHIR クライアントから到達する方法です。このサービスは、相互運用プロダクションのビジネス・サービスとは無関係です。

サービスが要求をどのように処理するかは、要求のタイプによって異なります。

- FHIR 相互作用に対応する HTTP メソッドとエンドポイントが要求に含まれる場合、サービスは、そのタイプの FHIR 相互作用を処理する Interactions クラスのメソッドに、その要求を転送します。例えば、`read` 対話作用が含まれる要求は、Interactions クラスの `Read` メソッドに送信されます。この Interactions クラスは、InteractionsStrategy クラスを使用して FHIR 相互作用を実行し、FHIR サーバの全体的な目的に従って相互作用を処理します。
- FHIR オペレーションの場合、サービスは、オペレーションを実行するために設計された特別なクラスに要求を転送します。リソース・リポジトリを使用する FHIR サーバでは、特定の FHIR オペレーションのサポートがすぐに利用可能です。

- タイプが `transaction` または `batch` のバンドルが要求に含まれる場合、サービスは、バンドルをアンパックして個々の HTTP オペレーションを実行する特別なクラスに要求を転送します。



3.1.1 サービスの詳細

サービスは、1 つのエンドポイントに対して自身の 1 つのインスタンスのみをインスタンス化できるシングルトン・クラスです。このインスタンス化は、最初の FHIR 要求が REST ハンドラまたはビジネス・オペレーションによってサービスに転送されたときに実行されます。インスタンス化されると、サービスは、プロセスが終了するまで存在します。FHIR 要求をプログラムによって実行するサーバ・アプリケーションの場合、アプリは、最初の要求を送信する前に、`HS.FHIRServer.Service.EnsureInstance()` を呼び出してサービスを取得する必要があります。

ほとんどの場合、Service クラス (`HS.FHIRServer.Service`) は、サブクラスを作成することなく、エンドポイントの FHIR 標準に準拠して要求をルーティングする準備ができています。FHIR サーバの動作を指定するカスタム・ロジックは、サービスにではなく、Interactions サブクラスおよび InteractionsStrategy サブクラスに記述します。

エンドポイント向けの新しいサービスの作成やサービスの削除など、サービスを管理するメソッドは、`HS.FHIRServer.API.RepoManager` のサブクラスに属しています。

3.1.2 InteractionsStrategy の詳細

InteractionsStrategy クラスは、FHIR サーバの全体的なストラテジを規定します。これは、FHIR サーバ・アプリケーションのバックエンドで、FHIR データの処理環境を作成および実装します。InteractionsStrategy のスーパークラスは、`HS.FHIRServer.API.InteractionsStrategy` です。

多くの場合、InteractionsStrategy は、FHIR サーバが FHIR リソースを保存および取得する方法の "ストレージ・ストラテジ" です。例えば、リソース・リポジトリは、FHIR データの保存と取得に使用されるリソースとインデックス・テーブルを作成する `HS.FHIRServer.API.InteractionsStrategy` のサブクラスによって実装されます。FHIR データを保存しないアプリケーションの場合、このストラテジにより、外部の FHIR サーバ、またはサーバの FHIR データを操作する他のカスタム・ロジックと通信する環境を設定できます。

InteractionsStrategy は、InteractionsStrategy を使用するサービスを管理する `HS.FHIRServer.API.RepoManager` のサブクラスに関連付けられています。

3.1.3 Interactions クラスの詳細

InteractionsStrategy クラスはアプリケーションのバックエンドですが、Interactions クラスを使用して、サービスが受信する FHIR 相互作用を実際に実行します。このプロセス中に、Interactions クラスは、特に FHIR サーバ・ストラテジ全体に共通する構造とロジックのために、InteractionsStrategy クラスのメソッドを呼び出すことがよくあります。このような相互依存関係があるため、統合アプローチでは、Interactions クラスと InteractionsStrategy クラスは同時にサブクラスが作成されます。Interactions のスーパークラスは、`HS.FHIRServer.API.Interactions` です。

FHIR 要求を処理する際にサービスによって呼び出される Interactions クラスのメソッドは、サーバ側 ObjectScript アプリケーションから直接呼び出すこともできます。例えば、サーバ側アプリケーションから、`POST` 要求をサービスに送信する代わりに、Interactions クラスの `Add` メソッドを呼び出すことができます。サービスをバイパスする場合、サーバ・アプリケーションは、サービスのメタデータによって FHIR サーバに設定された制約をバイパスできます。例えば、サービスを経由する要求に対してエンドポイントが読み取り専用であっても、サーバ・アプリケーションで FHIR サーバのストレージに入力することができます。

また、Interactions クラスは、FHIR オペレーションの実行、バンドルの処理、および FHIR データの検証のためにサービスで使用される特殊クラスの追跡も行います。サービスは、アクションが必要になると、Interactions オブジェクトからこれらのクラスの名前を取得します。

3.2 リソース・リポジトリ

リソース・リポジトリは、FHIR サーバの既定のストレージ・ストラテジで、他の開発タスクなしに、完全に機能する FHIR® サーバをインストールすることができます。リソース・リポジトリは、サーバが受信した FHIR データを自動的に、FHIR データの JSON データ構造をカプセル化したダイナミック・オブジェクトとして保存します。リソース・リポジトリを使用する FHIR サーバをインストールするには、インストール時に Interactions Strategy クラスとして `HS.FHIRServer.Storage.Json.InteractionsStrategy` を選択します。

リソース・リポジトリを使用する FHIR サーバで使用可能な FHIR の相互作用とオペレーションのリストは、"サポートされる相互作用とオペレーション" を参照してください。

リソース・リポジトリは、以下のアーキテクチャ・クラスで構成されます。

アーキテクチャ・コンポーネント	リソース・リポジトリ・クラス
Interactions	`HS.FHIRServer.Storage.Json.Interactions`
InteractionsStrategy	`HS.FHIRServer.Storage.Json.InteractionsStrategy`
RepoManager	`HS.FHIRServer.Storage.Json.RepoManager`

リソース・リポジトリのサブクラスを作成して、FHIR サーバをカスタマイズできます。詳細は、"FHIR サーバのカスタマイズ" を参照してください。

4

FHIR サーバのインストールと構成

管理ポータルには、新しい FHIR® サーバをインストールし、構成することができる [**サーバ構成**] ページが用意されています。または、プログラムによってサーバをインストールおよび構成することができます。

FHIR サーバは Foundation ネームスペースにインストールする必要があります。複数の FHIR サーバを同一の Foundation ネームスペースにインストールすることができます。

重要　FHIR サーバをインストールする前に、カスタマイズを今するのか、後でするのかを検討する必要があります。多くの場合、エンドポイントを作成する前に InteractionsStrategy のサブクラスを作成しない限り、リソース・リポジトリを使用する FHIR サーバはカスタマイズできません。例えば、バンドルの処理方法の変更や検索結果の後処理を実行するには、リソース・リポジトリのサブクラスを作成する必要があります。FHIR サーバをインストールする前にこのようなカスタマイズを準備する方法の詳細は、"インストール前のサブクラスの作成" を参照してください。

新しい FHIR サーバを管理ポータルからインストールするには、以下の手順を実行します。

1. 管理ポータルを開き、FHIR サーバをインストールする Foundation ネームスペースに切り替えます。Foundation ネームスペースがない場合は、[Health] に移動し、上部のメニュー・バーから [**インストーラ・ウィザード**] を選択します。[**Foundation 構成**] ボタンを使用すると、新しい Foundation ネームスペースを作成できます。作成後に、ネームスペースを必ず有効化してください。
2. [Health] > [MyNamespace] > [**FHIR 構成**] に移動します。[**FHIR 構成**] メニューが表示されない場合は、Foundation ネームスペースを使用していることを確認します。
3. [**サーバ構成**] カードを選択します。
4. [**エンドポイント**] ペインで、[**エンドポイントの追加**] をクリックし、新しい FHIR エンドポイントを作成します。
5. コア FHIR パッケージを選択します。各パッケージは、エンドポイントでサポートしている FHIR 標準のバージョンに対応しています。したがって、例えば、FHIR R4 をサポートする FHIR エンドポイントを構成するには、`hl7.fhir.r4.core@4.0.1` パッケージを選択します。
6. 選択したコア FHIR パッケージに応じて自動生成されたエンドポイント URL を確認します。エンドポイントの URL は変更できますが、必ずスラッシュ (/) で始めてください。
7. エンドポイントで追加のパッケージをサポートする場合は、[**追加パッケージ**] ドロップダウン・リストから選択します。パッケージの詳細は、"FHIR のプロファイルと適応" を参照してください。
8. エンドポイントの InteractionsStrategy を選択します。既定の相互作用ストラテジは、リソース・リポジトリ (HS.FHIRServer.Storage.Json.InteractionsStrategy) です。これにより、FHIR データはダイナミック・オブジェクトに JSON データとして格納されます。カスタムの InteractionsStrategy を作成した場合は、リストからそれを選択します。
9. 既定では、ネームスペース内の各エンドポイントのデータは 2 つの別個のデータベースに格納されます。別個のデータベースを維持したくない場合は、[**FHIR リソース・ストレージに別個のデータベースを使用する**] フィールドの

チェックを外します。この場合、すべての FHIR データはネームスペースの共通のデータベース・ファイルに格納されます。別個のデータベースを使用する場合は、既定の場所を受け入れることも、独自に場所を指定することもできます。リソース履歴データベースには、以前のバージョンのリソースが含まれています。これらは頻繁にアクセスされないため、このデータベースをより低速で安価なディスクに置くことができます。

10. **[追加]** を選択します。

コマンド行インタフェースを使用して FHIR サーバをインストールする場合は、"コマンド行オプション" を参照してください。

4.1 FHIR サーバの構成

FHIR サーバをインストールしたら、管理ポータルの **[サーバ構成]** ページを使用して、その設定を構成できます。これらの構成設定は、サーバの ConfigData オブジェクトのプロパティを設定することで、プログラムによって変更することもできます。

FHIR サーバを構成するには、以下の手順に従います。

1. **[Health]** > [MyNamespace] > **[FHIR 構成]** に移動します。FHIR サーバのネームスペースにいることを確認します。
2. **[サーバ構成]** カードを選択します。
3. 構成する FHIR サーバのエンドポイントを選択します。
4. ページが展開されたら、下方にスクロールして **[編集]** ボタンを選択します。
5. 以下の説明をガイドとして、設定を構成します。

設定	説明
有効	エンドポイントを有効にするかどうかを指定します。無効になっているエンドポイントは、FHIR クライアントからの要求を拒否します。
検索ページの既定のサイズ	検索に _count パラメータが含まれない場合に使用する検索結果ページのサイズ。
検索ページの最大サイズ	過剰なユーザ指定ページ・サイズを防止するための検索結果ページの最大サイズ。
検索結果の最大数	サーバがクエリにエラーで応答するまでの、1 つの検索で選択できるリソースの最大数。この数値には、実際の検索で選択されるリソースのみが含まれます。_include 検索パラメータで組み入れられるリソースは含まれません。この値は、検索で返されるページのサイズには影響しません。大量のリソースを選択する過剰に広範な検索は、実行するのに大量のシステム・リソースを必要とし、多くの場合、クライアントが実際に必要とするよりも広範になります。
条件付き削除結果の最大数	条件付き削除で削除できるリソースの最大数。条件付き削除検索でこの数を超えるリソースが見つかった場合、条件付き削除全体が「HTTP 412 前提条件が失敗しました」のエラーで拒否されます。

設定	説明
FHIR セッション・タイムアウト	サービスへの要求からセッション・データが古いと見なされるまでの最大秒数。
既定の推奨処理	検索条件に不明なパラメータが含まれる場合の既定の処理を指定します。不明なパラメータを無視し、OperationOutcome リソースによって問題が特定されたバンドルを返すには、`lenient` を指定します。検索要求を拒否してエラーを返すには、`strict` を指定します。Prefer ヘッダが含まれる FHIR 検索要求では、この既定値は上書きされます。
OAuth クライアント名	必要に応じて、OAuth リソース・サーバとして、FHIR サーバが OAuth 2.0 承認サーバへのアクセスに使用するアプリケーション名を指定します。OAuth 2.0 のサポートの詳細は、"OAuth 2.0 承認" を参照してください。
必須のリソース	InterSystems セキュリティ・リソースを指定する場合、FHIR クライアントは、サーバ上で相互作用を実行するため、リソースに対する特権を持っている必要があります。詳細は、"承認要件の追加" を参照してください。
サービス構成名	FHIR サーバに到達する前に相互運用プロダクション経由で FHIR 要求をルーティングするには、要求を受信するビジネス・サービスのパッケージと名前を入力します。ビジネス・サービスにカスタム名がない場合は、このエントリは `HS.FHIRServer.Interop.Service` になります。詳細は、"相互運用プロダクション" を参照してください。
認証なしアクセスを許可	認証と承認のストラテジを無視して、すべての FHIR 要求がサーバに届くようにすることができます。
新しいサービス・インスタンス	すべての FHIR 要求について、新しいサービス・オブジェクトをインスタンス化します。
トレースバックを含む	FHIR サーバは、OperationOutcome リソースでスタック・トレースを送信することによって、FHIR 要求に応答します。
SMART on FHIR 機能	エンドポイントの SMART on FHIR 機能のコンマ区切りリスト。このリストは、エンドポイントの機能を制御するものではありません。クライアントがエンドポイントの URL に `/.well-known/smart-configuration` を追加したときに JSON ドキュメントで返される機能を指定するものです。既知の URI で取得した FHIR 機能での SMART の詳細は、"FHIR Authorization Endpoint and Capabilities Discovery using a Well-Known Uniform Resource Identifiers (URIs)" を参照してください。

コマンド行インタフェースを使用して FHIR サーバを構成する場合は、"コマンド行オプション" を参照してください。

4.2 FHIR エンドポイントの削除

既定では、管理ポータルを使用して FHIR サーバ・エンドポイントを削除すると、そのエンドポイントに関連付けられている FHIR データも削除されます。ただし、エンドポイントを削除しても、そのすべての FHIR データは保持したい場合は、コマンド行インタフェースを使用して、エンドポイントを削除するのではなくデコミッションします。コマンド行インタフェースを使用したエンドポイントのデコミッションの詳細は、"コマンド行オプション" を参照してください。

エンドポイントを削除するには、以下の手順を実行します。

1. [Health] > [MyNamespace] > **[FHIR 構成]** に移動します。FHIR サーバのネームスペースにいることを確認します。
2. **[サーバ構成]** カードを選択します。
3. 削除するエンドポイントを選択します。
4. [ごみ箱] アイコンを選択します。

4.3 プログラムによるインストール

管理ポータルを使用する代わりにプログラムによって FHIR サーバをインストールする必要があるアプリケーションでは、最初にサーバをインストールしたうえで、構成を行う必要があります。

FHIR サーバは Foundation ネームスペースで実行する必要があるため、Foundation ネームスペースを作成することが FHIR サーバをインストールするための前提条件です。Foundation ネームスペースを作成したら、HS.FHIRServer.Installer の以下のメソッドを順番に呼び出す必要があります。

HS.FHIRServer.Installer メソッド	説明
InstallNamespace()	FHIR サーバ用の既存の Foundation ネームスペースを準備します。新しい Foundation ネームスペースは作成しません。引数なしで呼び出すと、インストーラは、アクティブなネームスペースを Foundation ネームスペースであると見なし、これを FHIR サーバ用に準備します。
InstallInstance()	FHIR サービスのインスタンスを現在のネームスペースにインストールします。このメソッドには、以下の引数が必要です。 ・ FHIR エンドポイントの一意の URL。URL は必ずスラッシュ (/) で始めてください。 ・ FHIR サーバの InteractionsStrategy のクラス名。 ・ FHIR パッケージのリスト (US Core などの実装ガイドのパッケージのリストなど)。詳細は、"pPackageList パラメータ" を参照してください。 また、`InstallInstance()` に渡すことができるオプション・パラメータもあります。これらのオプション・パラメータの詳細は、"InstallInstance()" を参照してください。

4.3.1 pPackageList パラメータ

InstallInstance() メソッドの pPackageList パラメータは、システムにロードされた FHIR パッケージのリストを受け入れます。多くの場合、パッケージは特定の実装ガイドに相当しますが、あるバージョンの FHIR のコア・メタデータである場合もあります。パッケージのリストを InstallInstance に渡すことにより、1 つ以上のパッケージをサポートするようエンドポイントを構成できます。パッケージの詳細は、"FHIR のプロファイルと適応" を参照してください。

pPackageList パラメータに渡すことができるパッケージのリストを取得するには、HS.FHIRMeta.Storage.Package.GetAllPackages() メソッドを使用します。例えば、以下のコードは、使用可能なパッケージの識別子を表示します。

```
set packages = ##class(HS.FHIRMeta.Storage.Package).GetAllPackages()
for i=1:1:packages.Count()
  { write packages.GetAt(i).id,! }
```

結果は以下のようになります。

```
hl7.fhir.r4.core@4.0.1
hl7.fhir.us.core@3.3.0
hl7.fhir.uv.vhdir@0.2.0
```

次に、$lb を使用して、これらのパッケージの識別子の一部を引数として pPackageList パラメータに渡すことができます。以下に例を示します。

```
Do ##class(HS.FHIRServer.Installer).InstallInstance(myURL,
                                    strategyClass,
                                    $lb("hl7.fhir.r4.core@4.0.1","hl7.fhir.us.core@3.1.0")
```

FHIR パッケージの作成に使用する API の詳細は、"パッケージ API" を参照してください。

4.3.2 プログラムによるインストールの例

以下の ObjectScript コードの例では、2 つのパッケージをサポートし、既定のストレージ・ストラテジ (リソース・リポジトリ) を使用する FHIR サーバをインストールします。

```
Set appKey = "/myfhirserver/fhir/r4"
Set strategyClass = "HS.FHIRServer.Storage.Json.InteractionsStrategy"
Set metadataPackages = $lb("hl7.fhir.r4.core@4.0.1","hl7.fhir.us.core@3.1.0")

//Install a Foundation namespace and change to it
Do ##class(HS.HC.Util.Installer).InstallFoundation("FHIRNamespace")
Set $namespace = "FHIRNamespace"

// Install elements that are required for a FHIR-enabled namespace
Do ##class(HS.FHIRServer.Installer).InstallNamespace()

// Install an instance of a FHIR Service into the current namespace
Do ##class(HS.FHIRServer.Installer).InstallInstance(appKey, strategyClass, metadataPackages)
```

4.4 プログラムによる構成

FHIR サーバをインストールしたら、HS.FHIRServer.Installer.UpdateInstance() メソッドを使用してプログラムによって構成できます。このメソッドは、サーバを構成する引数を複数受け入れます。その中には、サーバの HS.FHIRServer.API.ConfigData オブジェクトを受け入れる引数もあり、ここにはサーバのほとんどの構成オプションが含まれます。これらの構成オプションのリストは、クラス・リファレンスを参照してください。ConfigData オブジェクトで定義するオプション以外に、サーバのいくつかの設定 ([サービス構成名]、[OAuth クライアント名]、および [有効]) も、UpdateInstance() メソッドの専用のパラメータを使用して指定します。

以下のコードは、UpdateInstance() メソッドを使用して既存の FHIR サーバを構成します。

```
Set appKey = "/fhirendpoint/r4"

//Get and modify FHIR server's configuration object
Set strategy = ##class(HS.FHIRServer.API.InteractionsStrategy).GetStrategyForEndpoint(appKey)
Set configData = strategy.GetServiceConfigData()
Set configData.DefaultPreferHandling = "strict"
Set configData.DebugMode = 1
//stringify configData before updating FHIR Server
Set jsonConfigData = configData.AsJSONString()

// Define additional settings
Set enabled = 1
Set serviceConfigName = "HS.InteropPackage.myBusinessService"
Set oAuthClient = "OAuthClientName"

// Update FHIR Server
Do ##class(HS.FHIRServer.Installer).UpdateInstance(appKey, jsonConfigData, enabled, serviceConfigName,
 oAuthClient)
```

4.5 コマンド行オプション

管理ポータルよりもコマンド行インタフェースを好む開発者は、InterSystems ターミナルのコンソール設定を使用して、ユーザ・インタフェースで使用可能なアクションと同じアクションを数多く実行できます。コンソール設定を実行するには、InterSystems ターミナルを開いて以下を実行します。

```
do ##class(HS.FHIRServer.ConsoleSetup).Setup()
```

後続のセクションでは、コンソール設定で使用可能な各オプションについて説明します。

Create FHIRServer Endpoint

新しい FHIR サーバ・エンドポイントをインストールします。以下のプロンプトが表示されます。

- Choose the Storage Strategy ― Json がリソース・リポジトリです。
- Choose the FHIR version for this endpoint ― エンドポイントでサポートしているコア FHIR 仕様のバージョンを選択します。
- Enter any package numbers ― インポートされたパッケージが候補として表示されます。エンドポイントは複数のパッケージをサポートできます。複数のパッケージを指定するには、数字をコンマで区切って指定します。パッケージは後から追加できますが、待機する場合には、追加の手順を実行する必要がある場合があります。Upload a FHIR Metadata Package オプションを使用して、パッケージをリストに追加します。
- Do you want to create the default repository endpoint ― エンドポイントの既定の URL を受け入れる場合は、Enter キーを押します。エンドポイントに異なる URL を指定する場合は、N と指定し、URL を入力します (URL は必ずスラッシュ (/) で始めてください)。
- Enter the OAuth Client Name for this Endpoint ― OAuth 2.0 を使用してエンドポイントを保護する場合は、FHIR サーバのクライアント名を入力します。詳細は、"OAuth 2.0 承認" を参照してください。
- Do you want to create separate database files for your FHIR data? ― yes を指定すると、エンドポイントの FHIR データは、同じネームスペース内のその他のエンドポイントの FHIR データとは別に格納されます。no を指定すると、複数のエンドポイントがある場合でも、すべての FHIR データがネームスペースのデータベース・ファイルに格納されます。個別のデータベース・ファイルを作成している場

合は、既定の場所を受け入れることも、別の場所を指定することもできます。バージョン・データベースには、リソースの以前のバージョンが含まれています。これらは頻繁にアクセスされないため、バージョン・データベースをより低速で安価なディスクに置くことができます。

Add a profile package to an endpoint

FHIR パッケージを既存のエンドポイントに追加して、パッケージのプロファイル、検索パラメータ、およびその他の Conformance リソースをサポートできるようにします。このオプションを使用する前に、Conformance リソースを含む FHIR パッケージ (NPM のようなパッケージ) をアップロードしておく必要があります。`Upload a FHIR Metadata Package` オプションを使用して、FHIR パッケージをインポートできます。US Core Implementation Guide などの一部の共通パッケージは、既に利用できます。

パッケージに新しい検索パラメータが含まれている場合は、完了時に、`Index new SearchParameters for an Endpoint` オプションを実行する必要があります。

Display a FHIRServer Endpoint Configuration

FHIR サーバの現在の構成オプションを表示します。これらの構成オプションを変更するには、`Configure a FHIRServer Endpoint` オプションを使用します。

Configure a FHIRServer Endpoint

各構成オプションに値を指定することにより、FHIR サーバのエンドポイントを構成できます。各構成項目の詳細は、"FHIR サーバの構成" を参照してください。

Decommission a FHIRServer Endpoint

FHIR サーバ・エンドポイントは削除されますが、エンドポイントが収集した FHIR データは保持されます。FHIR データを含む SQL テーブルが保持されます。エンドポイントおよびすべての FHIR データを削除する場合は、`Delete a FHIRServer Endpoint` オプションを使用します。

Delete a FHIRServer Endpoint

FHIR サーバ・エンドポイントを削除し、かつエンドポイントの FHIR データを削除します。エンドポイントを削除するものの、エンドポイントが収集した FHIR データは保持する場合は、`Decommission a FHIRServer Endpoint` オプションを使用します。

Update the CapabilityStatement Resource

FHIR サーバの機能宣言書を更新します。詳細は、"機能宣言書の変更" を参照してください。

Index new SearchParameters for an Endpoint

公開またはカスタムのパッケージを使用して、新しい検索パラメータを既存のエンドポイントに追加する場合、FHIR クライアントは、パッケージの適用後に、新しいパラメータを使用して、リポジトリに追加されたリソースを取得できます。ただし、新しい検索パラメータを追加する前に存在していたリソースは、エンドポイントのインデックスを再作成するまで、返されません。エンドポイントが大量の FHIR データを収集していた場合、このオプションは既存のすべてのリソースを再処理するため、実行に長い時間がかかる可能性があります。

Upload a FHIR metadata package

Conformance リソースを定義する JSON ファイルの FHIR パッケージのインポートに使用します。このオプションは、パッケージをエンドポイントに適用する前に使用しておく必要があります。カスタム FHIR パッケージのアップロード準備の詳細は、"カスタム・パッケージの作成" を参照してください。

Delete a FHIR metadata package

エンドポイントに適用可能なパッケージのリストからパッケージを削除します。これにより、FHIR パッケージの JSON ファイルがローカル・システムから削除されることはありません。エンドポイントに適用されているパッケージは削除できません。

5
サポートされるFHIR相互作用とオペレーション

FHIR® サーバで提供されるリソース・リポジトリ・ストレージ・ストラテジを使用する場合、サーバは以下の相互作用とオペレーションをサポートします。カスタムのFHIRサーバがリソース・リポジトリを拡張する場合、これらの相互作用とオペレーションも既定でサポートされます。

5.1 相互作用

FHIR相互作用は、FHIRクライアントがリソースで実行できる一連のアクションです。これらの相互作用は、インスタンス、タイプ、またはシステム全体のどれに対して作用するのかに応じて、グループ化されます。インスタンスとは、リソースの特定のインスタンスのことであり、例えば、`Patient/1` は、`id` が `1` の Patient リソースのインスタンスを意味します。タイプは、Patient や Observation など、特定の FHIR リソースを表します。

以下の表は、リソース・リポジトリ、あるいは、リソース・リポジトリを拡張したカスタム FHIR サーバの FHIR 相互作用のサポート概要を示しています。相互作用をクリックすると、その相互作用が HL7 REST API でどのように定義されているかと、その使用方法を参照できます。

相互作用	サポート・レベル
create	条件付き作成を含め、完全にサポートされています。
read	条件付き読み取りはサポートされていません。
vread	条件付き読み取りはサポートされていません。
update	条件付き更新を含め、完全にサポートされています。
patch	JSON パッチ・ドキュメントのみサポートされています。
delete	条件付き削除を含め、完全にサポートされています。
history	インスタンスの相互作用のみサポートされています。タイプやシステムはサポートされていません。例えば、`GET [baseURL]/Patient/1/_history` はサポートされていますが、`GET [baseURL]/Patient/_history` や `GET [baseURL]/_history` はサポートされていません。 `_count` および `_at` パラメータはサポートされていません。 ページングはサポートされていません。
batch	完全にサポートされています。
transaction	バンドル内の循環参照はサポートされていません。
search	一部制限付きでサポートされています。詳細は、"search 相互作用" を参照してください。

5.1.1 search 相互作用

FHIR クライアントは、search 相互作用を使用して、リソース・リポジトリからリソースを取得します。search 相互作用の詳細は、FHIR 仕様を参照してください。このセクションでは、FHIR サーバがリソース・リポジトリを使用または拡張している場合の、search 相互作用に対する既定のサポートの概要を説明します。

5.1.1.1 一般的な制限

リソース・リポジトリを使用または拡張する FHIR サーバには、以下の制限があることに留意してください。

- 複数のリソース・タイプにまたがる検索はサポートされていません。例えば、`GET [base]?_id=1` はサポートされていません。
- コンパートメント内のすべてのリソース・タイプに対する検索は実行できません。例えば、`[base]/Patient/10000001/?_id=008` の検索はできません。したがって、コンパートメントのコンテキスト内の検索では、そのコンパートメントのリソース・タイプを指定する必要があります。例えば、`[base]/Patient/100000001/Observation` を使用して特定の患者のコンパートメント内のすべての Observation を返したり、`[base]/Patient/100000001/Observation?status=final` を使用して、コンパートメント内で Observation サブセットを検索することができます。Patient のコンパートメント全体を取得する場合は、`$everything` オペレーション (`[base]/Patient/100000001/$everything` など) を使用します。

5.1.1.2 検索パラメータ・タイプ

各検索パラメータには、パラメータの動作方法を決定する検索パラメータ・タイプがあります。

パラメータのタイプ	サポート・レベル
number	完全にサポートされています。
date	完全にサポートされています。
string	完全にサポートされています。
token	完全にサポートされています。
reference	完全にサポートされています。
composite	サポート対象外です。
quantity	完全にサポートされています。
uri	完全にサポートされています。

5.1.1.3 パラメータ

以下に、リソース・リポジトリからリソースを取得する場合の、標準の検索パラメータに対するFHIRサーバのサポートの概要を示します。

パラメータ	サポート・レベル
_id	正式な仕様の説明にあるとおり、完全にサポートされています。
_lastUpdated	正式な仕様の説明にあるとおり、完全にサポートされています。
_tag	正式な仕様の説明にあるとおり、完全にサポートされています。
_profile	正式な仕様の説明にあるとおり、完全にサポートされています。
_security	正式な仕様の説明にあるとおり、完全にサポートされています。
_text	サポート対象外です。
_content	サポート対象外です。
_list	サポート対象外です。
_source	完全にサポートされています。
_has	正式な仕様の説明にあるとおり、完全にサポートされています。
_type	サポート対象外です (オペレーション・クエリ・パラメータ _type は Patient および Encounter の $everything ではサポートされています)。
_filter	サポート対象外です。
_query	サポート対象外です。

5.1.1.4 修飾子

修飾子をパラメータの末尾に追加して、検索結果に影響を及ぼすことができます。

修飾子	サポート・レベル
:missing	サポート対象外です。
:exact	アクセント文字を除いた文字列をサポートしています。例えば、?given:exact=Nino は指定の名前 Niño を持つ Patient を返します。
:contains	文字列でサポートされています。
:above	URI でサポートされています。
:below	URI でサポートされています。
:type	参照でサポートされています。
:text	サポート対象外です。

5.1.1.5 接頭語

タイプが number、date、quantity の検索パラメータを使用する場合は、接頭語をパラメータの値に追加して、検索に一致するリソースに影響を及ぼすことができます。例えば、[parameter]=le100 は 100 以下の値を返します。

接頭語	サポート・レベル
eq	完全にサポートされています。
ne	完全にサポートされています。
gt	完全にサポートされています。
lt	完全にサポートされています。
ge	完全にサポートされています。
le	完全にサポートされています。
sa	サポート対象外です。
eb	サポート対象外です。
ap	サポート対象外です。

5.1.1.6 検索結果パラメータ

検索結果パラメータは、検索から返されるリソースの管理に役立ちます。

検索結果パラメータ	サポート・レベル
_sort	正式な仕様の説明にあるとおり、完全にサポートされています。
_count	正式な仕様の説明にあるとおり、完全にサポートされています。
_include	正式な仕様の説明にあるとおり、完全にサポートされています。
_revinclude	正式な仕様の説明にあるとおり、完全にサポートされています。
_summary	_summary=count のみサポートしています。詳細は、正式な仕様を参照してください。
_total	サポート対象外です。
_elements	正式な仕様の説明にあるとおり、完全にサポートされています。
_contained	サポート対象外です。
_containedType	サポート対象外です。

5.2 オペレーション

既定のリソース・リポジトリを使用または拡張する FHIR サーバについては、以下のオペレーションがサポートされています。

オペレーション	サポート・レベル
$everything	Patient および Encounter では完全にサポートされています。 Group および MedicinalProduct ではサポート対象外です。
$validate	検証モード (create、update、delete) はサポートされます。 プロファイルによる検証はサポートされません。 FHIR $validate 要求にリソース・ペイロードが含まれる場合、リソースを Parameters リソース内に埋め込むことができます。
$lastn	完全にサポートされています。

5.2.1 オペレーション・クエリ・パラメータ

特定のオペレーションでは、所定のオペレーション・クエリ・パラメータがサポートされています。

オペレーション	クエリ・パラメータ
`$everything`	• Patient と Encounter で `_since` がサポートされていて、その値として dateTime を使用できます。 • Patient および Encounter では `_type` がサポートされています。詳細は “$everything での _type オペレーション・クエリ・パラメータの再帰的動作” を参照してください。
`$lastn`	`max` がサポートされています。

5.2.1.1 $everything での _type オペレーション・クエリ・パラメータの再帰的動作

`$everything` オペレーションに対して `_type` クエリ・パラメータでリソース・タイプのリストが指定されている場合、コンパートメント検索は、リストされているタイプのリソースのみを返します。コンパートメントで再帰的にリソース参照を取得する場合、`_types` パラメータで指定されていないリソース・タイプへの参照はスキップされます。`$everything` の `_type` クエリ・パラメータが Patient コンパートメントに対してどのように動作するかについて、いくつか例を挙げて説明します。

1. `/Patient/123/$everything?_type=DiagnosticReport,Observation` ― DiagnosticReport リソースと Observation リソースを返しますが、Patient リソースは返しません。
2. `/Patient/123/$everything?_type=Observation` ― 参照する DiagnosticReport リソースが含まれなくても、患者の Observation リソースを返します。Observation は Patient コンパートメント内にもあるためです。
3. `/Patient/123/$everything?_type=Practitioner` ― 何も返しません。Practitioner は Patient コンパートメント内になく、Practitioner を参照できる他のリソース・タイプが指定されていませんでした。
4. `/Patient/123/$everything?_type=Patient,DiagnosticReport,Practitioner` ― Patient リソース、すべての DiagnosticReport リソース、および返される DiagnosticReport で直接参照されている Practitioner リソースのみを返します。

5.3 従来のリソース・リポジトリからの移行

InterSystems IRIS for Health 2019.4 以前を使用して開発された FHIR サーバでは、新しい FHIR サーバ・アーキテクチャを使用する前に、従来のリソース・リポジトリ内のデータを移行する必要があります。FHIR データを移行するには、以下の手順に従います。

1. 管理ポータルで、従来の FHIR サーバのネームスペースに切り替え、STU3 エンドポイントを作成します。
2. InterSystems ターミナルを開いて、従来の FHIR サーバのネームスペースに移動します。
3. 以下を実行します。

```
do ##class(HS.FHIRServer.ConsoleSetup).Migrate()
```

4. STU3 エンドポイントを選択して、移行を確認します。

6

FHIR のプロファイルと適応

FHIR® 標準は、特定の医療環境や実装に適応することを目的としています。これらの適応の中心には FHIR プロファイルがあり、ここで特定のリソースの許容できるフィールドを定義します。これらのプロファイルは、FHIR の基本仕様に定められているリソースの定義を拡張したり、制約したりします。プロファイルやその他の FHIR アーティファクトは、Conformance リソースを通じて定義できます。例えば、プロファイルは StructureDefinition リソースによって、検索パラメータは SearchParameter リソースによって、コードは ValueSet および CodeSystems リソースによって定義できます。

ほとんど場合、完全で堅牢な FHIR の適応は実装ガイドに定義されています。これは、適応固有のプロファイルやその他のアーティファクトを説明するドキュメントを含む Conformance リソースの一貫したコレクションです。通常、これらの実装ガイドは、配布サイトからダウンロード可能な NPM のようなパッケージとして配布されます。インターシステムズ製品では、Conformance リソースの FHIR パッケージに実装ガイド全体が含まれていない場合でも、このパッケージをエンドポイントに追加することにより、FHIR サーバのサポート対象を制御します。

インターシステムズ FHIR エンドポイントでは、複数の FHIR パッケージをサポートできます。例えば、FHIR エンドポイントでは、US Core Implementation Guide のパッケージをサポートすると同時に、カスタム・パッケージの一意の Patient プロファイルや検索パラメータをサポートできます。これにより、FHIR クライアントは、サポートされるすべてのパッケージに適合するリソースを検索したり、使用することができます。

FHIR 仕様に従い、インターシステムズの FHIR サーバで、FHIR クライアントから受信するリソースがサポートされているプロファイルに適合するかどうかを自動的に検証することはありません。FHIR クライアントは、リソースの `meta` 要素を使用して、リソースが 1 つ以上のプロファイルに適合することをアサートしますが、FHIR サーバはそのアサーションが真かどうかは確認しません。FHIR クライアントは、`_profile` 検索パラメータを使用して、プロファイルに適合していると主張するリソースを取得できます。

注釈　現在のところ、インターシステムズ製品では、リソースをプロファイルに対して検証する機能は提供されていません。検証は基本の FHIR 仕様に限られます。

FHIR サーバは FHIR の各種コア仕様をサポートしているため、FHIR サーバで許容できることや実現可能なことを FHIR クライアントで正確に判断できることが重要です。このニーズに対応するため、各 FHIR サーバには、サポート対象の API、FHIR オペレーション、検索パラメータ、およびリソースを特定する機能宣言書が用意されている必要があります。FHIR クライアントは、`GET [EndpointBaseURL]/metadata` を呼び出すことで、この機能宣言書を取得できます。

6.1 FHIR パッケージの操作

インターシステムズ製品では、FHIR パッケージは StructureDefinitions や SearchParameters などの Conformance リソースのコレクションです。このように、パッケージには、医療環境のプロファイルが含まれます。パッケージは、FHIR のあるバージョンに対応した標準の Conformance リソースを格納したり、特定の目的で FHIR のあるバージョンを拡張または制約したりできます。これらのパッケージは、JSON ファイルの NPM パッケージとして配布およびインポートされます。パッ

ケージの内容は大きく異なる可能性があります。国の実装ガイド (US Core など) の配布に使用したり、医療ネットワーク固有の Patient プロファイルに限定することもできます。配布サイトからダウンロード可能な標準の公開済みのパッケージを使用して、エンドポイントを構成する必要がある場合もあります。あるいは、カスタムのプロファイルや検索パラメータを含む独自のパッケージを開発する場合もあります。

プログラムによりパッケージを操作する必要がある場合は、"パッケージ API" を参照してください。

6.1.1 パッケージのインポート

エンドポイントを構成して、実装ガイドやカスタム・パッケージをサポートする前に、管理ポータルを使用して、公開済みパッケージやカスタム・パッケージをインポートしておく必要があります。一部の標準パッケージ (US Core など) は、既定で使用可能なため、エンドポイントに適用する前にインポートする必要はありません。

パッケージをインポートするには、以下の手順に従います。

1. パッケージの JSON ファイルがローカル・マシン上にあることを確認します。公開済みのパッケージをインポートする場合は、そのパッケージを配布サイトからローカル・マシンにダウンロードします。カスタム・パッケージに関する追加要件は、"カスタム・パッケージの作成" を参照してください。
2. 管理ポータルで、**[ホーム]** → [Health] → [MyFHIRNamespace] → **[FHIR 構成]** に移動します。
3. **[パッケージ構成]** カードを選択します。
4. 新しいパッケージの依存関係が既にインポートされていることを確認します。**[パッケージ構成]** ページの左側のナビゲーション・バーを見ると、インポート済みのパッケージを確認できます。
5. **[パッケージをインポート]** を選択します。
6. パッケージの JSON ファイルを含むディレクトリを選択します。個々のファイルは選択しないでください。
7. **[インポート]** を選択します。

FHIR パッケージに含まれていたプロファイルとその他のアーティファクトのパッケージがエンドポイントで使用できるようになりました。

6.1.2 パッケージのアンインストール

パッケージが別のパッケージとの間に依存関係がなく、既存のエンドポイントに適用されていない場合は、そのパッケージを FHIR サーバのネームスペースから削除できます。パッケージをアンインストールしても、パッケージのインポートに使用されたローカルの JSON ファイルは削除されません。パッケージをアンインストールするには、以下の手順に従います。

1. 管理ポータルで、**[ホーム]** → [Health] → [MyFHIRNamespace] → **[FHIR 構成]** に移動します。
2. **[パッケージ構成]** カードを選択します。
3. 左側のナビゲーション・バーからパッケージを選択します。
4. **[パッケージのアンインストール]** を選択します。エンドポイントに適用されているパッケージはアンインストールできません。さらに、別のパッケージの依存関係であるパッケージもアンインストールできません。

6.1.3 カスタム・パッケージの作成

カスタム・パッケージを使用して、カスタムのプロファイルや検索パラメータをサポートするよう FHIR エンドポイントを構成できます。例えば、カスタム検索パラメータを追加するには、ローカル・マシンにある JSON ファイルで SearchParameter リソースを定義します。次に、同じディレクトリに package.json というファイルを作成します。少なくとも、このファイルには、

パッケージの名前、バージョン、依存関係が含まれている必要があります。例えば、package.json ファイルは次のようになります。

```
{
  "name":"myorg.implementation.r4",
  "version":"0.0.1",
  "dependencies": {
    "hl7.fhir.r4.core":"4.0.1"
  }
}
```

Conformance リソースの定義を含む JSON ファイルと package.json ファイルが 1 つのディレクトリに用意できたら、新しいパッケージをインポートする準備は完了です。

6.1.4 エンドポイントへのパッケージの適用

新しい FHIR エンドポイントを作成する場合は、エンドポイントでサポートするパッケージを選択します。エンドポイント作成時には、インポートされたこれらのパッケージのみ使用できます。インターシステムズ製品には、既にインポートされた公開済みパッケージがいくつか付属しています。

新しいパッケージを既存のエンドポイントに適用することもできます。既存のエンドポイントにパッケージを追加するには、以下の手順に従います。

1. 管理ポータルで、**[ホーム]** → [Health] → [[MyFHIRNamespace]] → **[FHIR 構成]** に移動します。
2. **[サーバ構成]** カードを選択します。
3. リストからエンドポイントを選択します。
4. **[編集]** を選択します。
5. **[追加パッケージ]** ドロップダウン・リストを使用して、パッケージを選択します。リストにパッケージが表示されない場合は、パッケージがインポートされていることを確認します。
6. **[更新]** を選択します。

重要　既存のエンドポイントにパッケージを適用し、パッケージに新しい検索パラメータがある場合、エンドポイントのインデックスを再作成するまで、新しいパラメータを使用して既存のリソースを取得することはできません。詳細は、"エンドポイントのインデックスの再作成" を参照してください。

6.1.5 エンドポイントのインデックスの再作成

公開またはカスタムのパッケージを使用して、新しい検索パラメータを既存のエンドポイントに追加する場合、FHIR クライアントは、パッケージの適用後に、新しいパラメータを使用して、リポジトリに追加されたリソースを取得できます。ただし、新しい検索パラメータを追加する前に存在していたリソースは、エンドポイントのインデックスを再作成するまで、返されません。エンドポイントのインデックスを再作成するまで、新しい検索パラメータを使用する FHIR クライアントは、検索結果が不完全である可能性があることを示す OperationOutcome を受信します。

新しい検索パラメータを含むパッケージを適用したら、エンドポイントのインデックスを再作成するオプションが [サーバ構成] ページ ([Health] → **[FHIR 構成]** → **[サーバ構成]**) のエンドポイントの URL の横に表示されます。リポジトリに既存のリソースが多数ある場合は、エンドポイントのインデックスの再作成に長時間かかる可能性があります。

6.1.6 パッケージ API

実装で、ユーザ・インタフェースを使用せずに直接パッケージを操作する必要がある場合は、以下の API メソッドを利用できます。

パッケージのインポート

インターシステムズ FHIR サーバは、パッケージを使用して、サポートする FHIR プロファイルやその他のアセットを判断します。インターシステムズ製品には、FHIR の基本バージョンに対応する事前にロードされたパッケージと一般的な実装ガイドが付属していますが、StructureDefinition や ValueSet などの Conformance リソースを定義する JSON ファイルを含むディレクトリを指定することにより、新しいパッケージをインポートすることもできます。FHIR パッケージの詳細は、"FHIR パッケージの操作" を参照してください。

エンドポイントに追加できるよう新しいパッケージをインポートする API は、HS.FHIRMeta.Load.NpmLoader.importPackages() です。例えば、以下のコードでカスタム・パッケージがインポートされます。

```
do
##class(HS.FHIRMeta.Load.NpmLoader).importPackages($lb("C:\fhir-packages\node_modules\myorg.fhir.myPackage\"))
```

利用可能なパッケージのリスト

ネームスペースにインポートされたパッケージのリストを取得するには、HS.FHIRMeta.Storage.Package.GetAllPackages() メソッドを使用します。例えば、以下のコードは、使用可能なパッケージの識別子を表示します。

```
set packages = ##class(HS.FHIRMeta.Storage.Package).GetAllPackages()
for i=1:1:packages.Count()
  { write packages.GetAt(i).id,! }
```

エンドポイント作成時のパッケージの指定

`InstallInstance()` メソッドの `pPackageList` パラメータを使用すると、新しいエンドポイントに適用するパッケージを指定できます。詳細は、"プログラムによるインストールと構成" を参照してください。

既存のエンドポイントへのパッケージの追加

既存のエンドポイントにパッケージを追加する必要がある場合は、HS.FHIRServer.Installer.AddPackagesToInstance() を利用できます。

パッケージのアンインストール

パッケージが別のパッケージとの間に依存関係がなく、既存のエンドポイントに適用されていない場合は、HS.FHIRMeta.Load.NpmLoader.UninstallPackage() を使用して、そのパッケージを FHIR サーバのネームスペースから削除できます。パッケージをアンインストールしても、パッケージのインポートに使用されたローカルの JSON ファイルは削除されません。利用可能なパッケージのリストを表示することにより、アンインストールするパッケージの ID を特定できます。例えば、パッケージをアンインストールする呼び出しは次のようになります。

```
do ##class(HS.FHIRMeta.Load.NpmLoader).UninstallPackage("myorg.r4@1.0.0")
```

6.2 カスタム検索パラメータ

カスタム検索パラメータをエンドポイントに追加するには、SearchParameter リソースを含むカスタム・パッケージを作成し、それをエンドポイントに適用します。プロセスを完了するには、以下の操作を行います。

- テキスト・エディタまたはサードパーティのツールを使用して、SearchParameter JSON ファイルを作成します。
- JSON ファイルと package.json ファイルを 1 つのファイル・ディレクトリに入れると、カスタム・パッケージとしてインポートできます。詳細は、"カスタム・パッケージの作成" を参照してください。
- パッケージをインポートします。
- エンドポイントにパッケージを適用します。

- 既存のエンドポイントにパッケージを適用する場合は、エンドポイントのインデックスの再作成が必要な場合があります。

6.3 拡張機能

FHIR サーバは、基本の FHIR 仕様で定義されている拡張機能の構文に従った適切な形式になっている限り、拡張機能が含まれるリソースを受け付けます。FHIR 仕様に従い、FHIR サーバでは、これらの拡張子が有効なのか、リソースの `meta` フィールドに指定されているプロファイルに適合しているのかを自動的に検証することはありません。

拡張機能へのカスタム検索パラメータの追加の詳細は、"カスタム検索パラメータ" を参照してください。

7

FHIR 相互運用アダプタ

すべてのソリューションに、要求を内部リポジトリにルーティングする FHIR® サーバが必要なわけではありません。例えば、実装で、インターシステムズ製品にペイロードを格納することなく、FHIR 要求を受信して外部 FHIR サーバに転送する必要がある場合があります。FHIR サーバの内部リポジトリを活用せずに、FHIR 要求を処理する必要がある場合は、FHIR 相互運用アダプタを使用して要求を受信して相互運用プロダクションに取り込みます。リポジトリと共に FHIR サーバをインストールするライセンスがない Health Connect 実装の場合、受信 FHIR 要求は、FHIR 相互運用アダプタのインストールにより処理されます。

FHIR 相互運用アダプタをインストールすると、特殊なビジネス・ホストを使用して、プロダクションで FHIR 要求を処理する新しい相互運用 REST エンドポイントが作成されます。この相互運用 REST エンドポイントは、FHIR サーバ・エンドポイントと一緒に管理ポータルに表示されることはありません。

7.1 アダプタのインストール

FHIR 相互運用アダプタをインストールするには、次の手順に従います。

1. 相互運用プロダクションを備えたネームスペースを作成します。
2. InterSystems ターミナルを開いて、作成したネームスペースに変更します。例えば、以下のように入力します。

   ```
   set $namespace = "myFHIRNamespace"
   ```

3. 以下のコマンドを実行し、相互運用 REST エンドポイントの URL を指定します。

   ```
   set status = ##class(HS.FHIRServer.Installer).InteropAdapterConfig("/MyEndpoint/r4")
   ```

   アダプタのエンドポイントの URL は スラッシュ (/) で始まる必要があります。

4. コマンドが正常に実行されたことを確認するには、以下を入力します。

   ```
   write status
   ```

   応答は `1` になります。

5. これがネームスペース用に作成された最初の FHIR 相互運用アダプタである場合は、**[相互運用性]** → **[リスト]** → **[プロダクション]** に移動し、プロダクションを開き、以下のいずれかを実行します。
   - **[更新]** ボタンが表示された場合は、これを選択します。
   - **[更新]** ボタンが表示されず、プロダクションをテストする準備ができている場合は、**[開始]** を選択して、プロダクションを開始します。

7.2 アダプタ・コンポーネント

FHIR 相互運用アダプタをインストールすると、以下が作成されます。

- 指定の URL を持つ Web アプリケーション。
- `InteropService` という、相互運用プロダクションの新しいビジネス・サービス。複数のアダプタをインストールする場合、それらはすべて同じ `InteropService` ビジネス・サービスを使用します。アダプタで異なるビジネス・サービスを使用する場合は、"カスタム・ビジネス・サービスの使用法" を参照してください。
- `InteropOperation` という、相互運用プロダクションの新しいビジネス・オペレーション。これは、ユース・ケースに応じて拡張または置き換えることができるプレースホルダのビジネス・オペレーションです。`InteropOperation` を変更してカスタム機能を実装するまで、FHIR 要求が新しい相互運用 REST エンドポイントによって受信されると、`501 Unimplemented` エラーが返されます。

  注釈　プロダクションを使用している場合は、HS.FHIRServer.Interop.Response の ContentType プロパティを明示的に設定して HTTP 応答の Content-Type ヘッダを作成する必要があります。HS.FHIRServer.API.Data.Response で ResponseFormatCode を設定するだけでは不十分です。

FHIR 相互運用アダプタと併用できるその他のプロダクション・コンポーネントの詳細は、"相互運用プロダクション" を参照してください。

7.3 カスタム・ビジネス・サービスの使用法

既定では、複数の FHIR 相互運用アダプタをインストールする場合、それらはすべて同じビジネス・サービス `InteropService` を共有します。アダプタ・エンドポイントで受信される要求がそれぞれ異なるビジネス・サービスにルーティングされるようにする場合は、REST ハンドラのサブクラスを作成し、これをアダプタの CSP アプリケーションのディスパッチ・クラスとして指定します。カスタム・ビジネス・サービスを使用するこのプロセスは、以下の手順で構成されます。

1. IDE 使用して、`HS.FHIRServer.HC.FHIRInteropAdapter` のサブクラスを作成します。
2. サブクラスの `ServiceConfigName` パラメータを使用して、FHIR 要求を受信するカスタム・ビジネス・サービスの名前を指定します。
3. 管理ポータルで、**[システム管理]** → **[セキュリティ]** → **[アプリケーション]** → **[ウェブ・アプリケーション]** に移動します。
4. FHIR 相互運用アダプタの URL を選択します。
5. **[一般]** タブの **[有効]** フィールドを使用して、**[ディスパッチ]** テキスト・ボックスにサブクラスの名前を指定します。
6. **[保存]** を選択します。

7.4 セキュリティ

相互運用 REST エンドポイントのセキュリティは、FHIR 相互運用アダプタ用に作成された Web アプリケーションのセキュリティ設定に応じて異なります。例えば、FHIR 要求を行うユーザがインターシステムズのリソースに対する特権を持つことを求める Web アプリケーションを構成できます。Web アプリケーションのセキュリティ設定は、**[システム管理]** → **[セ**

キュリティ] → [アプリケーション] → [ウェブ・アプリケーション] に移動すると行うことができます。Web アプリケーションは相互運用 REST エンドポイントの URL で識別されます。セキュリティ設定の詳細は、"Web アプリケーションの編集" を参照してください。

FHIR 相互運用アダプタでは、OAuth 2.0 の広範なサポートは提供されません。このアダプタへの要求に OAuth 2.0 トークンが含まれる場合、そのトークンは基本的なテストで検証されます。このテストにより、トークンが Authorization ヘッダ内にあるか、空白以外であるか、および安全な接続上にあるかを判断します。FHIR サーバと異なり、スコープや患者コンテキスト値など、トークンの内容は検証されません。アダプタの基本的なテストに合格すると、トークンは要求メッセージの HS.FHIRServer.API.Data.Request の `AdditionalInfo` プロパティに追加されます。

8

FHIR の相互運用プロダクション

InterSystems の医療製品には、FHIR® 要求を送受信する相互運用プロダクションの作成に使用できるビジネス・ホストが組み込まれています。SDA を FHIR に、FHIR を SDA に変換するビジネス・プロセスも利用可能です。

相互運用プロダクションを使用できる FHIR 実装に対する理解を深めるには、"ユース・ケース" を参照してください。

注釈　インターシステムズの FHIR サーバには、相互運用プロダクションは必要ありません。既定では、エンドポイントの REST ハンドラで受信された FHIR 要求は、FHIR サーバのサービスに直接送信されます。相互運用プロダクションを使用しない FHIR サーバの方が、はるかに高速になる可能性があります。

8.1 FHIR 要求の受け入れ

FHIR の実装では、次の 2 つの方法で FHIR 要求を受け入れ、相互運用プロダクションに取り込むことができます。

- FHIR サーバを使用する (インターシステムズ製品の内部アーキテクチャとストレージを活用する実装) 場合は、FHIR 要求は `HS.FHIRServer.Interop.Service` を介してリポジトリに送信できます。詳細は、"FHIR サーバ要求の受け入れ" を参照してください。
- FHIR サーバを使用しない実装の場合は、FHIR 相互運用アダプタを使用して、FHIR 要求を受け入れ、相互運用プロダクションに取り込むことができます。詳細は、"FHIR 相互運用アダプタ" を参照してください。

8.1.1 受信要求のセキュリティ

FHIR 要求は、FHIR 相互運用アダプタ、またはプロダクションを使用する FHIR サーバを使って相互運用プロダクションに到達できます。セキュリティは、要求の受信に使用している機能に応じて異なる方法で処理されます。プロダクションで FHIR 相互運用を使用している場合は、"アダプタのセキュリティ" を参照してください。プロダクションで FHIR サーバを使用している場合は、"サーバ・セキュリティ" を参照してください。

8.1.2 FHIR サーバ要求の受け入れ

組み込みのビジネス・サービス HS.FHIRServer.Interop.Service は、FHIR サーバ・エンドポイントに送信された FHIR 要求を受信するよう設計されています。構成されると、エンドポイントの REST ハンドラは、要求を FHIR サーバのサービスではなく、`HS.FHIRServer.Interop.Service` にルーティングします。

FHIR サーバ要求を相互運用プロダクション経由でルーティングするようにエンドポイントを設定するのは、以下の 2 段階のプロセスです。

1. 相互運用プロダクションを作成し、HS.FHIRServer.Interop.Service ビジネス・サービスを追加します。
2. エンドポイントの [**サービス構成名**] フィールドを構成し、相互運用プロダクションに追加されたビジネス・サービスの名前を指定します。

エンドポイントの構成内でのビジネス・サービスの名前が相互運用プロダクション内での名前に一致している限り、これらの手順は、設定が完了したら任意の順序で実行できます。

注釈　この 2 段階プロセスでは、FHIR サーバを使用していることを前提としています。実装で FHIR サーバの内部アーキテクチャやリポジトリを利用しない場合は、FHIR 相互運用アダプタを使用して、FHIR 要求を受け入れます。

8.1.2.1 相互運用プロダクションの作成

FHIR サーバ・エンドポイントの Foundation ネームスペースが作成されると、インストール・プロセスにより、FHIR プロダクションとして使用する相互運用プロダクションも作成されます。プロダクションを変更し、要求をプロダクション経由でルーティングするためにエンドポイントで使用する、必須のビジネス・サービスを追加する必要があります。

FHIR 要求を REST ハンドラから受信する相互運用プロダクションには、HS.FHIRServer.Interop.Service ビジネス・サービスが含まれる必要があります。このビジネス・サービスにはカスタム名を付けることができますが、エンドポイントの [**サービス構成名**] オプションで指定されている名前と一致する名前にしてください。

8.1.2.2 エンドポイントの構成

FHIR サーバのエンドポイントをインストールしたら、相互運用プロダクションを使用するようにエンドポイントを構成できます。これは、プロダクションの作成前を含む任意の時点で行うことができます。エンドポイントの構成中にビジネス・サービスの名前を指定しても、プロダクションにそのビジネス・サービスが自動的に作成されることはありません。

FHIR 要求がプロダクション経由でルーティングされるように既存のエンドポイントを構成するには、以下の手順に従います。

1. [**Health**] → [MyNamespace] → [**FHIR 構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. [**サーバ構成**] カードを選択します。
3. エンドポイントを選択します。
4. [**編集**] を選択します。
5. [**相互運用性**] セクションの [**サービス構成名**] フィールドで、FHIR 要求のルーティングで経由するプロダクションのビジネス・サービスの名前を指定します。例えば、ビジネス・サービスにカスタム名を付けていない場合は、HS.FHIRServer.Interop.Service を指定します。
6. [**更新**] を選択します。

8.2 FHIR 要求の送信

相互運用プロダクションでは、ビジネス・オペレーションが FHIR 要求を確実に FHIR エンドポイントへ送信します。この要求は、InterSystems エンドポイントにアクセスする外部 FHIR クライアントや、HL7 メッセージを FHIR 要求に変換するビジネス・プロセスなど、さまざまなソースから生じます。その要求元に関係なく、要求の送信には、次の 2 つのビジネス・オペレーションを利用できます。

ビジネス・オペレーション・クラス	説明
HS.FHIRServer.Interop.Operation	FHIR 要求をローカル・ネームスペース内のインターシステムズ FHIR サーバの内部サービスに送信します。カスタム・アーキテクチャが実装されているという稀な場合を除き、Health Connect ユーザは、リソース・リポジトリの使用ライセンスがないと、このビジネス・オペレーションを使用することはできません。 このビジネス・オペレーションでは、エンドポイントの URL (要求メッセージの SessionApplication プロパティに含まれる) に基づいてインターシステムズの適切な FHIR サーバを特定します。REST ハンドラ経由で FHIR サーバのエンドポイントに送信された要求からのメッセージの場合、エンドポイントの URL は既にメッセージの一部に組み込まれています。メッセージが SDA を FHIR に変換するビジネス・プロセス (HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process) から送信された場合は、サーバはビジネス・プロセスの FHIREndpoint 設定によって特定されます。
HS.FHIRServer.Interop.HTTPOperation	FHIR 要求を、HTTP 経由で内部または外部の FHIR エンドポイントに送信します。 組み込みのビジネス・ホストを使用して、要求をこのビジネス・オペレーションに送信する場合は、そのビジネス・ホストの TargetConfigName 設定を使用します。 FHIR エンドポイントの既定の HTTP アドレスは、ビジネス・オペレーションの ServiceName 設定を使用して指定されます。これにより、サービス・レジストリのエントリが参照されます。この既定の設定は、要求に ServiceName という AdditionalInfo 項目が含まれる場合にオーバーライドされます。これにより、別のエンドポイントを指すサービス・レジストリ・エントリが指定されます。

組み込みのビジネス・ホスト (HS.FHIRServer.Interop.Service など) が要求メッセージ (HS.FHIRServer.Interop.Request) を HS.FHIRServer.Interop.HTTPOperation ビジネス・オペレーションに送信する場合、要求はカスタム・コードなしで HTTP 経由で送信されます。ただし、FHIR ペイロードが、FHIR 要求にペイロードを含める必要があるカスタム・ビジネス・ホスト内に作成される場合は、相互運用 FHIR クライアントをインスタンス化してメッセージを送信する必要があります。同様に、カスタム・ビジネス・ホストがエンドポイントから FHIR データを取得する必要がある場合は、プロダクションで FHIR クライアントを使用する必要があります。

8.3 相互運用 FHIR クライアント

インターシステムズのテクノロジは、カスタム・ビジネス・ホストから FHIR 要求を作成し、HTTP 経由で FHIR エンドポイントに送信するプロセスを簡素化する FHIR クライアント・オブジェクトを提供しています。HTTP 経由で要求を送信するために FHIR クライアントで使用されるビジネス・オペレーション HS.FHIRServer.Interop.HTTPOperation を相互運用プロダクションに追加する必要があります。プロダクションを構成したら、HS.FHIRServer.RestClient.Interop をイ

ンスタンス化し、FHIR の相互作用とオペレーションに対応するメソッドを呼び出すことにより、カスタム・ビジネス・ホストで FHIR クライアントを使用できるようになります。

HTTP 経由で FHIR 要求を送信するすべてのプロダクションに、相互運用 FHIR クライアントのインスタンス化が必要なわけではありません。例えば、`HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process` を使用して SDA が FHIR に変換される場合は、このビジネス・プロセスから `HS.FHIRServer.Interop.HTTPOperation` に転送される FHIR は、FHIR クライアントを使用せずに HTTP 経由で送信されます。ただし、FHIR ペイロードがプロダクション内のカスタム・ビジネス・ホストによって作成される場合、HTTP 経由で FHIR エンドポイントに送信する際に推奨される方法は、FHIR クライアントのインスタンス化です。

カスタム・ビジネス・ホストのコンテキスト内で FHIR クライアントをインスタンス化する場合は、CreateInstance( ) メソッドに対する呼び出しに以下のパラメータを含める必要があります。

- `pServiceName` ― FHIR エンドポイントを指すサービス・レジストリ内のエントリの名前。この値は、`HS.FHIRServer.Interop.HTTPOperation` ビジネス・オペレーションの ServiceName 設定をオーバーライドします。
- `pTargetConfigName` ― `HS.FHIRServer.Interop.HTTPOperation` ビジネス・オペレーションの名前。
- `pHostObj` ― FHIR クライアントをインスタンス化するビジネス・ホストのオブジェクト・インスタンス。`$this` を使用して、FHIR クライアントをインスタンス化する現在のビジネス・ホスト・オブジェクトを指定できます。

例えば、必要な引数のみを使用してビジネス・ホスト内で FHIR クライアントをインスタンス化するには、以下のように入力します。

```
Set fhirClient = ##class(HS.FHIRServer.RestClient.Interop).CreateInstance("MyFHIR.HTTP.Service", , ,
, , ,
"HS.FHIRServer.Interop.HTTPOperation",
                                                                $this)
```

また、CreateInstance( ) メソッドは、FHIR `prefer` ヘッダの値を指定し、OAuth トークンを要求と共に送信するオプションの引数も受け入れます。

FHIR クライアントをインスタンス化すると、それを使用して要求を送信したり、オペレーションを実行できます。FHIR クライアントのメソッドを使用してこれらのアクションを実行する方法の詳細は、"相互作用とオペレーション" を参照してください。

注釈　相互運用 FHIR クライアントのクラス (HS.FHIRServer.RestClient.Interop) を、相互運用プロダクションを通じて FHIR 要求を送信する必要があるスタンドアロンの ObjectScript アプリケーションで使用することもできます。この場合、`HS.HC.Util.BusinessService` ビジネス・サービスは、`HS.FHIRServer.Interop.HTTPOperation` と共にプロダクションに追加する必要があります。クライアントのインスタンス化は似ていますが、スタンドアロン・アプリケーションの場合、`CreateInstance` の呼び出しに `pHostObj` パラメータの引数を含めることはできません。

8.4 変換

組み込みのビジネス・プロセスをプロダクションに追加して、SDA-FHIR 変換を起動できます。例えば、プロダクションでは、HL7 メッセージを利用し、ビジネス・プロセスを使用して HL7 を SDA に変換し、さらに組み込みの SDA-FHIR ビジネス・プロセスを使用して SDA を FHIR に変換できます。これらの変換を実行するプロダクションは、Foundation ネームスペースに含まれている必要があります。

組み込みのビジネス・プロセスを使用した SDA-FHIR 変換の詳細は、"変換ビジネス・プロセス" を参照してください。

8.5 ユース・ケース

以下のユース・ケースでは、組み込みの相互運用コンポーネントを使用してFHIRリソースを操作する方法の例を示しています。

- プロキシ・サーバ
- HL7 から FHIR への変換
- プロダクションベースのインターシステムズ FHIR サーバ

プロキシ・サーバ

インターシステムズの医療製品は、外部 FHIR クライアントからの FHIR 要求を受け入れ、それらを外部 FHIR エンドポイントに転送し、さらに FHIR エンドポイントからの応答を外部クライアントに戻すプロキシ・サーバとして使用できます。このシナリオでは、インターシステムズ製品が FHIR を受け入れ、生成するサーバではないこと、また、要求や応答は必要に応じてプロダクション内で操作できることを、FHIR クライアントが認識していない可能性があります。

単純なプロキシ・サーバは、以下を実行することにより実装できます。

- FHIR 相互運用アダプタのインストール。
- `HS.FHIRServer.Interop.HTTPOperation` のプロダクションへの追加、および ServiceName 設定の編集による外部 FHIR エンドポイントの指定。
- `InteropService` の TargetConfigName を編集することによる、`HS.FHIRServer.Interop.HTTPOperation` の指定。

もちろん、プロキシ・サーバのユース・ケースは多様です。例えば、複数の `HS.FHIRServer.Interop.HTTPOperation` ビジネス・オペレーションを追加し、ビジネス・プロセスを使用して、プロキシ・サーバのターゲットにする外部 FHIR エンドポイントを決定することもできます。

FHIR データを外部 FHIR エンドポイントに送信すると共に、内部 InterSystems リポジトリに格納したい場合は、FHIR 相互運用アダプタではなく、FHIR サーバを使用して開始する必要があります。この場合、`HS.FHIRServer.Interop.HTTPOperation` を使用して外部エンドポイントに要求を送信できますが、`HS.FHIRServer.Interop.Operation` を使用してデータを内部に格納することもできます。

HL7 から FHIR への変換

インターシステムズの医療製品は、受信 HL7 メッセージから臨床データを抽出し、そのデータを FHIR リソースに変換するプロセスを簡素化します。FHIR に変換すると、臨床データは外部 FHIR エンドポイントに転送したり、FHIR クライアントが照会できる内部 FHIR リポジトリに格納することができます。HL7 メッセージを FHIR リソースに変換する基本の相互運用プロダクションには、以下が含まれます。

- HL7 メッセージを受け入れてプロダクションに取り込む、`EnsLib.HL7.Service.HTTPService` などの組み込みのビジネス・サービスの追加。
- ビジネス・ホストを使用した HL7 から SDA (InterSystems の中間データ形式) への変換。ビジネス・プロセスに追加された以下のコードで、HL7 を SDA に変換できます。

  ```
  do ##class(HS.Gateway.HL7.HL7ToSDA3).GetSDA(request,.con)
  ```

  この変換方法の詳細は、"インターシステムズの医療製品のデータ変換" を参照してください。
- `HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process` ビジネス・プロセスのプロダクションへの追加。このビジネス・プロセスは SDA を FHIR に変換します。

- HL7 から SDA への変換方法を含むビジネス・ホストの TargetConfigName 設定の変更による、HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process の名前の指定。

HL7 データを FHIR に変換したら、それを外部 FHIR エンドポイントに送信したり、FHIR サーバの場合は、内部リソース・リポジトリに格納したりできます。特定の機能を実行するビジネス・オペレーションを追加することにより、FHIR データの転送先を制御します。これらのビジネス・オペレーションの詳細は、"FHIR 要求の送信" を参照してください。要求を FHIR サーバの内部ストレージに転送するビジネス・オペレーションを使用する場合は、HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process の FHIREndpoint 設定を使用して、インターシステムズ FHIR サーバのエンドポイントを指定します。

HL7 メッセージの FHIR サーバとの統合の実践例は、"FHIR R4 Integration QuickStart" を参照してください。

プロダクションベースのインターシステムズのサーバ

既定では、インターシステムズの FHIR サーバへの要求は相互運用プロダクションを経由しませんが、場合によっては、プロダクションを使用したいことがあります。例えば、開発中にはプロダクションを使用して、メッセージ・トレースやプロダクションのその他のメリットを活用し、その後の実動時にはサーバのサービスに要求を直接送信するように少々変更を加えるような場合です。また、別のユース・ケースでは、インターシステムズの FHIR サーバに到着する前に、ビジネス・プロセスを使用して FHIR 要求を操作したい場合があります。

最も簡潔な形式では、プロダクションベースの FHIR サーバは、"FHIR サーバ要求の受け入れ" で説明されているようにプロダクションを構成し、その後、"FHIR 要求の送信" で説明されているように HS.FHIRServer.Interop.Operation を追加して構成されます。両方のビジネス・ホストがプロダクションに追加されたら、HS.FHIRServer.Interop.Operation の TargetConfigName 設定を変更し、HS.FHIRServer.Interop.Operation ビジネス・オペレーションの名前を指定します。

開発中にプロダクションを使用することが目的である場合は、要求をサービスに直接送信する FHIR サーバに切り替え、サーバが実稼働するときには、[**サービス構成名**] フィールドの値を削除して、サーバのエンドポイントを再構成します。

9

SDA-FHIR 変換

インターシステムズでは、データ変換言語 (DTL) を使用して SDA オブジェクトを FHIR® リソースに (またはその逆に) 変換することができます。SDA は、ある標準から別の標準へと簡単に移行できる中間の臨床形式です。例えば、HL7v2 を FHIR に直接変換するのではなく、HL7v2 を SDA に変換してから、SDA を FHIR に変換できます。SDA の詳細は、"SDA : インターシステムズの臨床データ形式" を参照してください。

双方向の SDA-FHIR 変換は、以下を含むさまざまなユース・ケースで役立つ機能を提供できます。

- SDA 対応システムからコンテンツを取得して、FHIR システムに提供する
- SDA 対応システムからコンテンツを取得して、FHIR リポジトリに保存する
- 複数の SDA 対応システムからコンテンツを取得して、FHIR システムで使用または保存するために正規化する
- FHIR システムからコンテンツを取得して、SDA 対応システムに提供する

SDA オブジェクトを FHIR リソースに、またはその逆に変換する DTL 変換を起動するためのオプションは 2 つあります。つまり、組み込みのビジネス・プロセスを相互運用プロダクションに追加して DTL 変換を起動するか、カスタム・ビジネス・プロセスなどから変換 API を直接呼び出すかのいずれかです。

9.1 変換ビジネス・プロセス

組み込みのビジネス・プロセスを使用して、SDA から FHIR への変換プロダクションまたは FHIR から SDA への変換プロダクション内で SDA-FHIR 変換を起動できます。例えば、プロダクションでは、HL7 メッセージを利用し、ビジネス・プロセスを使用して HL7 を SDA に変換し、さらに組み込みの SDA-FHIR ビジネス・プロセスを使用して SDA を FHIR に変換できます。

組み込みのビジネス・プロセスで使用される基礎となる変換コードの詳細は、"変換 API" を参照してください。これらの API はカスタム・ビジネス・プロセスから直接呼び出すことができます。

9.1.1 SDA から FHIR への変換プロダクション

組み込みのビジネス・プロセスである HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process は、プロダクションに追加して SDA オブジェクトやコンテナを FHIR バンドルに変換できます。このプロダクションは、Foundation ネームスペースに存在する必要があります。

プロダクションに追加したら、ビジネス・プロセスでは以下が可能になります。

- SDA コンテナを入力として受け付けて、含まれる各オブジェクトをループする
- SDA コンテナを FHIR `Bundle` リソース形式の FHIR コンテンツに変換する

- TargetConfigName 設定で指定されたビジネス・ホストに FHIR コンテンツを転送する
- ビジネス・ホストから応答を受信する
- 応答 (受信した内容に基づく) を、最初にそれを呼び出したビジネス・ホストに返す

SDA から FHIR への変換プロダクションのビジネス・プロセスは HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR クラスのメソッドを呼び出して変換を実行します。このクラスによる変換の処理方法の詳細は、"変換の詳細" を参照してください。

9.1.1.1 ビジネス・プロセスの追加

最初に、管理ポータルの [プロダクション構成] ウィンドウでプロダクションを開き、`HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process` ビジネス・プロセスを追加します。追加したら、変換に影響を与えるビジネス・プロセスの設定を変更できます。ビジネス・プロセスの相互運用プロダクションへの追加方法の概要は、"機能紹介 : 相互運用プロダクションを使用したシステムの接続" を参照してください。

9.1.1.2 ビジネス・プロセスの設定

SDA から FHIR への変換に影響を与える `HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process` の設定には、以下があります。

- TargetConfigName ― HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process が出力を送信するビジネス・ホストを指定します。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[基本設定]** セクションにあります。
- TransmissionMode ― このビジネス・プロセスが追加処理のために FHIR バンドルを送信する方法を指定します。
  - transaction ― このビジネス・プロセスは、リソースのバンドルを単一の相互作用で送信し、処理はバンドル全体を対象として成功または失敗します。1 つのリソースの処理に失敗した場合、その他のリソース (およびバンドル全体) の処理が停止します。これが既定値です。
  - individual ― このビジネス・プロセスは、バンドルからの各リソースを専用の相互作用として個別に送信します。

  この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。
- FullTransactionResponse ― 選択した場合、このプロセスが送信する FHIR 要求メッセージは、"PREFER" ヘッダの値が "return=representation" に設定されて作成されます。FHIR 仕様に基づき、このヘッダは、FHIR サーバに対して、作成または更新された各リソースを、保存されているますべて (すなわち、サーバにより適用されたすべての変更を含む) 返す必要があることを示します。サーバが実際にこれを行うかどうかはサーバによります。一般に、この情報を要求すると FHIR サーバからの応答時間が増大する傾向があるため、この設定は、デバッグ中、または FHIR クライアントに作成/更新したリソースを受信するという特定のニーズがある場合を除き、チェックを付けないままにする必要があります。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。
- FHIRFormat ― コンテンツが XML 形式と JSON 形式のどちらであるかを指定します。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。
- FormatFHIROutput ―コンテンツを読みやすい形式にするかどうかを指定します。この設定を選択すると、パフォーマンスが影響を受けます。そのため、開発時およびテスト時にのみ有効にしてください。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。
- CallbackClass ― 非推奨。
- ValidResourceRequired ― 非推奨。
- OutputToQuickStream ― 選択した場合、このビジネス・プロセスにより送信された FHIR ペイロードは HS.SDA3.QuickStream オブジェクトに配置され、QuickStream オブジェクトの ID は要求メッセージの `QuickStreamId` プロパティに配置されます。選択しないままにすると、変換からの FHIR 出力は、要求メッセージの `Payload` プロ

パティに配置されます。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。

- TransformClass ― 変換を実行するクラスの名前を指定します。`HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR` のサブクラスを作成して変換動作をカスタマイズする場合は、そのサブクラスの名前を指定する必要があります。
- FHIRMetadataSet ― パッケージに基づいて送信 FHIR のバージョンを指定します。使用可能なすべてのパッケージがドロップダウン・リストに表示されます。
- FHIREndpoint ― FHIR サーバのエンドポイントを指定します。この設定は、ビジネス・プロセスが送信 FHIR を `HS.FHIRServer.Interop.Operation` ビジネス・オペレーションを経由して FHIR サーバのサービスに送信する場合に必要です。

9.1.1.3 患者 ID の割り当て

`HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process` に送信される SDA メッセージの `AdditionalInfo` プロパティを使用して、SDA-FHIR 変換で作成された Patient リソースに ID を割り当てることができます。SDA メッセージに `PatientResourceId` という `AdditionalInfo` 項目が含まれる場合、変換で `PatientResourceId` の値が取得され、生成された Patient リソースの ID フィールドに割り当てられます。

注釈　変換ビジネス・プロセスで使用される基礎となるクラス `HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR` には、オーバーライドして Patient リソースを含むリソースに ID を割り当てることができるメソッドが含まれています。詳細は、"変換 API クラスのカスタマイズ" を参照してください。

9.1.1.4 メッセージ

HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process への要求メッセージは、Ens.Container または HS.Message.XMLMessage です。

HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process からの応答メッセージはありません。代わりに、成功または失敗のステータスを返します。

9.1.2 FHIR から SDA への変換プロダクション

組み込みのビジネス・プロセスである HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process は、プロダクションに追加して FHIR リソースやバンドルを SDA オブジェクトやコンテナに変換できます。このプロダクションは、Foundation ネームスペースに存在する必要があります。

プロダクションに追加したら、ビジネス・プロセスでは以下が可能になります。

- FHIR のリソースまたはバンドルを入力として受け付ける
- FHIR コンテンツを SDA コンテナに変換する
- TargetConfigName 設定で指定されたビジネス・ホストにそのコンテナを転送する
- ビジネス・ホストから応答を受信する
- FHIR の応答 (受信した内容に基づく) を、最初にそれを呼び出したビジネス・ホストに返す

SDA から FHIR への変換プロダクションのビジネス・プロセスは HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 クラスのメソッドを呼び出して変換を実行します。このクラスによる変換の処理方法の詳細は、"変換の詳細" を参照してください。

9.1.2.1 ビジネス・プロセスの追加

最初に、管理ポータルの [プロダクション構成] ウィンドウでプロダクションを開き、`HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process` ビジネス・プロセスを追加します。追加したら、変換に影響を与えるビジネス・プロセスの設定を変更できます。ビジネス・プロセスの相互運用プロダクションへの追加方法の概要は、"機能紹介 : 相互運用プロダクションを使用したシステムの接続" を参照してください。

9.1.2.2 ビジネス・プロセスの設定

FHIR から SDA への変換に影響を与える `HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process` の設定には、以下があります。

- TargetConfigName – DTL 変換により FHIR から変換された後、SDA3 ストリームを含む XMLMessage が送信されるビジネス・ホストを指定します。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[基本の設定]** セクションにあります。
- CallbackClass – 非推奨。
- OutputToQuickStream – 既定で、HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process の出力は、DTL 変換により生成される SDA3 ストリームを含む HS.Message.XMLMessage オブジェクトです。この設定にチェックが付いている場合、SDA3 ストリームは個別の HS.SDA3.QuickStream オブジェクトに配置され、QuickStream オブジェクトの QuickStreamID は XMLMessage の AdditionalInfoItem プロパティに配置されます。この設定が選択されていない場合、SDA3 ストリームは XMLMessage の ContentStream プロパティに配置されます。この設定は、[プロダクション構成] ウィンドウの **[設定]** タブの **[追加設定]** セクションにあります。
- TransformClass – 変換を実行するクラスの名前を指定します。`HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3` のサブクラスを作成して変換動作をカスタマイズする場合は、そのサブクラスの名前を指定する必要があります。
- FHIRMetadataSet – パッケージに基づいて受信 FHIR のバージョンを指定します。使用可能なすべてのパッケージがドロップダウン・リストに表示されます。

9.2 変換 API

アプリケーションは、SDA から FHIR への変換 API および FHIR から SDA への変換 API の両方にアクセスできます。

9.2.1 SDA から FHIR への変換 API

SDA から FHIR に変換するためにコードで使用する API は、HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR にあります。アプリケーションは、SDA が `%Stream.Object` と SDA オブジェクトのどちらに存在するのかに応じて、`TransformStream` または `TransformObject` メソッドを呼び出すことができます。

これらのメソッドはどちらも、その `bundle` プロパティに FHIR 出力を含む変換オブジェクト (`HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR`) を返します。また、そのタイプは `%DynamicObject` です。このバンドルには、変換によって生成されたすべてのリソース (すべての参照は解決済み) が含まれます。

コードに以下のコードを追加することで、ダイナミック・オブジェクトから JSON または XML まで、この bundle プロパティをシリアル化できます。これは、SDA3ToFHIRObject が、いずれかの変換メソッドから返される変換オブジェクトであることを前提としています。

```
Set stream = ##class(%Stream.TmpCharacter).%New()
Set metadataSetKey = "R4"
If format="JSON" {
  Do SDA3ToFHIRObject.bundle.%ToJSON(stream)
} ElseIf format="XML" {
  Set schema = ##class(HS.FHIRServer.Schema).LoadSchema(metadataSetKey)
  Do ##class(HS.FHIRServer.Util.JSONToXML).JSONToXML(SDA3ToFHIRObject.bundle, stream, schema)
}
Do stream.Rewind()
```

9.2.1.1 TransformStream メソッドの使用法

HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR の TransformStream メソッドは、%Stream として SDA を取得し、それを FHIR バンドルに変換します。そのシグニチャは、以下のとおりです。

```
TransformStream(stream As %Stream.Object, SDAClassname As %String,
    fhirVersion As %String, patientId As %String = "",
    encounterId As %String = "")
```

パラメータ：

- stream ― SDA オブジェクトまたはコンテナの %Stream 表現。
- SDAClassname ― ストリームに含まれるオブジェクトのクラス名 (HS.SDA3.Container など)。
- fhirVersion ― 変換によって生成される FHIR のバージョン。STU3 や R4 など。
- patientId ― このオプションのパラメータを指定した場合、生成された Patient リソースの Id フィールドに指定された値が入ります。
- encounterId ― このオプションのパラメータを指定した場合、生成された Encounter リソースの Id フィールドに指定された値が入ります。stream パラメータが SDA コンテナである場合、このパラメータは無視されます。これは、コンテナには、複数の Encounter を含めることができるので、どの FHIR Encounter に指定のリソース ID を付与するのか判断できないためです。

9.2.1.2 TransformObject メソッドの使用法

HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR の TransformObject メソッドは、コンテナやオブジェクト・クラスとして SDA を取得し、それを FHIR バンドルに変換します。そのシグニチャは、以下のとおりです。

```
TransformObject(source, fhirVersion As %String, patientId As %String = "", encounterId As %String =
"")
```

パラメータ：

- source ― FHIR に変換される SDA コンテナまたは SDA オブジェクト・クラス。
- fhirVersion ― 変換によって生成される FHIR のバージョン。STU3 や R4 など。
- patientId ― このオプションのパラメータを指定した場合、生成された Patient リソースの Id フィールドに指定された値が入ります。
- encounterId ― このオプションのパラメータを指定した場合、生成された Encounter リソースの Id フィールドに指定された値が入ります。stream パラメータが SDA コンテナである場合、このパラメータは無視されます。これは、コンテナには、複数の Encounter を含めることができるので、どの FHIR Encounter に指定の ID を付与するのか判断できないためです。

9.2.1.3 変換の詳細

以下に SDA から FHIR への変換の既定の動作を説明します。オーバーライドして変換動作をカスタマイズできるメソッドの概要は、"オーバーライド可能な SDA から FHIR への変換メソッド" を参照してください。

- 受信ストリームまたはオブジェクトは個々のストリームレットに分割され、それらが STU3 リソースに変換されます。
- 既定では、UUID が生成され、Bundle リソースの `fullUrl` フィールドに割り当てられます。この場合、リソース自体には `Id` はありません。リソース ID を提供する場合は、`GetId` メソッドをオーバーライドします。この場合、`fullUrl` の値は *`baseURL/resourceType/id`* で、リソース参照は *`resourceType/id`* になります。
- これらのメソッドは、既定では受信 URL を一切変更しません。この動作を `GetBaseURL` メソッドでオーバーライドできます。例えば、特定のリポジトリにポストする場合、そのリポジトリの URL 接頭語を指定できます。
- ID を割り当てるために使用されたメカニズムに関係なく、リソースには他のリソースへの参照が含まれます。
- Patient および Encounter のストリームレットを使用して、利用可能なすべてのリソースに Patient および Encounter の参照が追加されます。Encounter の参照は、SDA ストリームレットの `EncounterNumber` フィールドが使用されている場合にのみ、正常に作成できます。これらが空の場合、参照は生成されません。
- Organization、Practitioner、Medication などの共有リソースの場合、各リソースの先頭 32 キロバイトのハッシュがハッシュ・テーブルに追加されます。それ以降の各共有リソースは、直接一致するものをハッシュ・テーブルで検索することで、重複がないかどうかが確認されます。一致が見つかった場合、リソースは重複としてマークされます。この動作は、`IsDuplicate` メソッドをオーバーライドすることにより変更できます。
- 各リソースは、バンドルに追加される前に検証されます。リソースが検証に失敗すると、エラーがスローされ、処理が停止します。つまり、バンドルは返されません。この既定の動作は、`HandleInvalidResource` メソッドをオーバーライドすることにより変更できます。
- 1 つ以上の SDA プロパティがターゲット・スキーマの FHIR リソース・フィールドにマップされない場合は、変換により、SDA データが FHIR 拡張にマップされます。詳細は、"FHIR 拡張" を参照してください。
- 特定の SDA オブジェクトやプロパティがターゲットの FHIR リソースやフィールドにマップされる方法の詳細は、"SDA-FHIR マッピングの理解" を参照してください。

9.2.2 FHIR から SDA への変換 API

FHIR から SDA に変換するためにコードで使用する API は、HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 にあります。このクラスには、ユース・ケースに応じて、FHIR から SDA への変換に使用できる複数の API が含まれています。

ほとんどの場合、アプリケーションで単一の FHIR リソースまたはバンドルの変換が必要な際には、FHIR が `%Stream.Object` とダイナミック・オブジェクトのどちらに存在するのかに応じて、`TransformStream` または `TransformObject` クラス・メソッドを呼び出す必要があります。ただし、連続して複数の FHIR バンドルやリソースを変換する場合は、変換クラスを一旦インスタンス化してから、`Transform` メソッドを複数回呼び出すと、より効率的です。

これらの変換メソッドはすべて、`container` プロパティに SDA 出力を含む変換オブジェクト (`HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3`) を返します。また、そのタイプは `HS.SDA3.Container` です。変換オブジェクトの `object` プロパティには、変換によって最後に生成された SDA コンテナまたはオブジェクトが含まれます。最後の入力がバンドルの場合は、`object` プロパティは SDA コンテナになります。最後の入力が個々のリソースである場合は、`object` は SDA オブジェクトになります。

9.2.2.1 TransformStream メソッドの使用法

HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 の TransformStream メソッドは、%Stream として表現される FHIR リソースやバンドルを取得し、それを SDA コンテナに変換します。FHIR バンドルがメソッドに渡される場合にのみ、リソース参照は処理されます。そのシグニチャは、以下のとおりです。

```
TransformStream(stream As %Stream.Object, fhirVersion As %String, fhirFormat As %String)
```

パラメータ：

- stream － FHIR リソースまたはバンドルの %Stream 表現。
- fhirVersion － 変換される FHIR リソースまたはバンドルのバージョン。"STU3" や "R4" など。
- fhirFormat － FHIR リソースまたはバンドルの形式を指定します。許容値は "JSON" と "XML" です。

9.2.2.2 TransformObject メソッドの使用法

HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 の TransformObject メソッドは、ダイナミック・オブジェクトとして FHIR リソースやバンドルを取得し、それを SDA コンテナに変換します。バンドルがメソッドに渡される場合にのみ、リソース参照は処理されます。そのシグニチャは、以下のとおりです。

```
TransformObject(source As %DynamicObject, fhirVersion As %String)
```

パラメータ：

- source － ダイナミック・オブジェクトとして表現される FHIR リソースまたはバンドル。
- fhirVersion － 変換される FHIR リソースまたはバンドルのバージョン。"STU3" や "R4" など。

9.2.2.3 Transform メソッドの使用法

HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 の Transform メソッドは、ダイナミック・オブジェクトとして FHIR バンドルを取得し、それを SDA コンテナに変換します。バンドルがメソッドに渡される場合にのみ、リソース参照は処理されます。

Transform は、FHIR を SDA に変換するクラス・メソッドによって呼び出されるメソッドです。連続して複数の FHIR リソースを変換する場合に直接呼び出すことができるので、HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 オブジェクトを毎回インスタンス化する必要はありません。例えば、以下のコードでは、同一の変換オブジェクトを使用して、Patient リソース、Encounter リソース、および Observation リソースを変換します。

```
set r4schema = ##class(HS.FHIRServer.Schema).LoadSchema("R4")
set transformer = ##class(HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3).%New(r4Schema)
do transformer.Transform(patient)
do transformer.Transform(encounter)
do transformer.Transform(Observation)
```

Transform メソッドのシグニチャは次のとおりです。

```
Transform(source As %DynamicObject)
```

パラメータ：

- source － ダイナミック・オブジェクトとして表現される FHIR リソースまたはバンドル。

9.2.2.4 変換の詳細

以下に、FHIR から SDA への変換の既定の動作の概要を示します。オーバーライドして変換動作をカスタマイズできるメソッドの概要は、"オーバーライド可能な FHIR から SDA への変換メソッド" を参照してください。

- 受信 FHIR バンドルは個々のリソースに分割され、それらが SDA3 ストリームレットに変換されます。
- 受信 FHIR バンドル内で別のリソースによって参照されているリソースがバンドルに存在しない場合、バンドルの変換は続行されます。この動作を変更するには、`HandleMissingResource` メソッドをオーバーライドします。
- 参照をオブジェクトに変換しようとしても、以下の場合には、SDA ストリームレットにオブジェクトは作成されません。
  - サブ変換が存在するが、マッピングがあるどの要素についても、参照されているリソースに値がない。
  - 参照されているリソース・タイプから SDA3 オブジェクトのデータ型へのサブ変換がない。
- Encounter ストリームレットの `EncounterNumber` フィールドは 1 から入力され、Encounter が処理されるたびに増やされます。Encounter リソースを参照する以降のリソースは、SDA3 への変換時に、リソース ID に基づいてルックアップを実行し、使用すべき Encounter 番号を特定します。Encounter 番号の割り当ては、`GetIdentifier` メソッドでオーバーライドできます。変換されるリソースのコンテンツにアクセスすると、返すべき EncounterNumber の決定に役立ちます。インスタンス・プロパティ `%currentReference` には、ダイナミック・オブジェクトとしてリソースを取得するために、インスタンス・メソッド `GetResourceFromReference` に渡すことができる FHIR 参照オブジェクトが含まれています。
- Encounter 番号と同じように、HealthConcern リソースと Goal リソースの `ExternalID` の値も、既定では 1 から入力されます。この動作は、`GetIdentifier` メソッドでオーバーライドできます。
- SDA `Container:SendingFacility` プロパティの値は次のように設定されます。Patient の `managingOrganization` フィールドに Organization への参照が含まれており、その Organization が Bundle に存在する場合は、それが使用されます。そうでない場合は、患者識別子で MRN と割り当て機関が検索され、その割り当て機関が使用されます。いずれの項目も見つからない場合は、文字列 `FHIR` が使用されます。この動作は、`GetSendingFacility` メソッドでオーバーライドできます。
- SDA3 拡張は使用されません。SDA3 にフィールドが存在しない場合、その内容は削除されます。
- Patient リソースなしでバンドルを受信した場合、エラーがスローされます。それ以外には、コンテナに対して検証は実行されません。単に、受信したままの状態で返されます。
- 包含関係に関する情報を確認するには、Bundle リソースの管理ポータルで FHIR アノテーション ([Health] → [FHIR アノテーション]) を参照します。
- 特定の FHIR リソースまたはフィールドが SDA オブジェクトやプロパティにマップされる方法の詳細は、"SDA-FHIR マッピングの理解" を参照してください。

9.3 SDA-FHIR マッピングの理解

SDA-FHIR 変換を実行するために変換 API と組み込みのビジネス・プロセスのどちらを使用するにしても、**FHIR アノテーション**・ツールを使用すると、SDA または FHIR データが目的の形式にどのように変換されたのかについて正確に把握することができます。このツールでは、特定の FHIR リソースにマップされた SDA オブジェクト (またはその逆) の概要を確認できると同時に、マッピングをドリルダウンして、SDA オブジェクトのプロパティがどのようにして FHIR リソースのフィールドに取り込まれたか (またはその逆) について正しく把握できます。**FHIR アノテーション**・ツールを使用する場合、SDA プロパティはフィールドとして参照されます。例えば、マッピングはプロパティとフィールドではなく、フィールドとフィールドとして参照されます。また、ルックアップ・テーブルを使用して SDA と FHIR 間でコードをマップする方法について調べ、変換に関わるデータ型について学習し、変換に使用された ObjectScript メソッドを検出することもできます。

マッピングの背後にあるロジックを把握するには、"マッピング規則" を参照してください。

9.3.1 FHIR アノテーション・ツールへのアクセス

FHIR アノテーション・ツールにアクセスするには、以下の手順に従います。

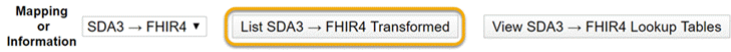
1. 管理ポータルに **%Ens_EDISchemaAnnotations** ロールを持つユーザとしてログインします。
2. [Health] → [MyFHIRnamespace] に移動します。
3. **[スキーマ・ドキュメント]** メニュー・オプションを展開して、**[FHIR アノテーション]** をクリックします。

マッピングの調査を開始するには、**[マッピングまたは情報]** ドロップダウン・リストを使用して、変換元と変換先を選択します。例えば、SDA3 から FHIR R4 へのマッピングが目的の場合は、**[SDA3 –> FHIR4]** を選択します。

FHIR アノテーション・ツールを使用して SDA-FHIR マッピングを調べるときには、**[ヘルプ]** ボタンと [FAQ] ボタンを選択すると、ツールのユーザ・インタフェースの使用および解釈方法についてのガイダンスが表示されます。また、ユーザ・インタフェースの多くの要素について、カーソルを合わせると情報が表示されます。

9.3.2 マッピングの概要

特定のマッピング (フィールド間マッピングなど) の詳細にまでドリルダウンする前に、SDA オブジェクトと FHIR リソース間のすべてのマッピングの概要を入手しておくと役に立つ場合があります。オブジェクトとリソース相互のマッピング方法のリストを表示するには、**[変換された <transform> をリスト]** を選択します。



9.3.3 マッピングの詳細

特定の SDA オブジェクトまたは FHIR リソースがどのように目的の形式にマップされているのか詳しく知りたい場合は、ドロップダウン・リストから該当のオブジェクトまたはリソースを選択できます。例えば、Appointment リソースの SDA3 へのマッピングを表示するには、**[名前別の FHIR4]** ドロップダウン・リストから [Appointment] を選択します。

各マッピングは、SDA フィールドと FHIR フィールド間のマッピング、ソース・フィールドのカーディナリティ、ソース・フィールドのデータ型、およびその他の有用な情報をすべて表示するテーブルに表示されます。テーブルの要素の詳細を確認するには、以下のようにします。

- テーブルの各要素の上にカーソルを移動して、追加情報を取得します。
- **[アクション]** 列のアイコンなどのリンクをクリックして、要素の詳細を開きます。例えば、データ型をクリックして、そのデータ型のマッピング方法を調べることができます。
- [マッピング定義] アイコン ( ) をクリックして、マッピングの技術的詳細まで掘り下げます。[マッピング定義] が開いたら、FHIR データ型をクリックして、正式な FHIR 仕様で表示することができます。また、マッピングで使用されるカーディナリティ、既定値、サブ変換やクラス・メソッドなどの技術的詳細も表示できます。場合によっては、マッピングの説明を補足するその他のメモもあります。

以下は、マッピング・テーブルの **[アクション]** 列にあるアイコンの凡例です。

アイコン	意味
	マッピング定義の詳細を表示します。
	サブ変換またはクラス・メソッドを使用します。
	このフィールドの設定条件を使用して、マッピングの使用を制御します。
	FHIR 拡張をターゲットとして使用します。
	ソースに見つからないまたは無効なデータが含まれている場合は、既定値がターゲットに割り当てられます。
	マッピングが使用されている場合は、変換により、新しいターゲット・オブジェクトが作成されるのではなく、既存のターゲット・オブジェクトにデータが追加されます。
	FHIR リソース・フィールドのサブフィールドのマッピングを表示します。

9.3.4 ルックアップ・テーブルのマッピング

FHIR アノテーション・ツールでは、SDA と FHIR 間でコードをマップするルックアップ・テーブルを表示できます。例えば、`HS.SDA3.Alert` オブジェクトの `Status` プロパティにあるコード A は、FHIR `event-status` 値セットの `in-progress` コードにマップされていることがわかります。ソースからターゲットへのコードのマップに使用されるルックアップ・テーブルを調べるには、[<transform> **ルックアップ・テーブルの表示**] を選択します。

ルックアップ・テーブルをカスタマイズするには、"ルックアップ・テーブルのカスタマイズ" を参照してください。

9.3.5 マッピング規則

このセクションでは、SDA-FHIR マッピングの背後にあるロジックについて説明します。

9.3.5.1 フィールド間マッピング

ほとんどのマッピングはフィールド間のマッピングです。マッピングによってソース・フィールドでデータ値を見つけ、その値をターゲット・フィールドに割り当てます。例えば、SDA プロパティの値は FHIR フィールドに割り当てられます。

9.3.5.2 条件付きマッピング

一部のフィールド間マッピングは条件付きです。特定の条件が満たされた場合にのみ、値がターゲット・フィールドに割り当てられます。この情報を提示する場合、**FHIR アノテーション**・ツールに [**このフィールドの設定条件**] というラベルが表示されます。DTL の <if> 要素により、これをコード内で制御します。

9.3.5.3 リテラル値

定義済みマッピングの中には、リテラル値からターゲット・フィールドへのマッピングがあります。これらのマッピングの目的の 1 つは、ターゲットで必要なデータを提供できるフィールドがソース・オブジェクト定義に含まれない場合に、必要なターゲット・フィールドの値を提供することです。

多くの場合、このタイプのマッピングは条件付きで定義され、必要時にのみ使用されます。

9.3.5.4 除外されるフィールド

臨床的な意義がないメタデータを含む SDA フィールドは、FHIR フィールドにマップされません。例えば、`UpdatedOn` プロパティは FHIR に変換されません。

また、クラス・リファレンス内で使用されていませんとマークされている SDA フィールドは、FHIR に変換されません。例えば、`LabResultItem` の `ExternalId` プロパティは、Observation リソースのフィールドにマップされません。

9.3.5.5 単一値からリストへのマッピング

ソース・フィールドが単一値であるのにターゲット・フィールドがリストの場合、ソース項目は、変換により、ターゲット・リストの最初のエントリにマップされます。変換後、リストには 1 つのエントリのみが含まれます。この機能は、変換用コードの生成中に自動的に処理されます。単一値からリストへの場合、マッピング定義に特別な注意は必要ありません。

9.3.5.6 リストから単一値へのマッピング (値)

ソース・フィールドが値のリストで、ターゲット・フィールドが単一値に制限されている場合、変換により、値のリストは、リストの各値をセミコロンとスペースで区切った 1 つの値に連結されます。

9.3.5.7 リストから単一値へのマッピング (オブジェクト)

SDA から FHIR の場合 : 受信 SDA がオブジェクトのリストで、ターゲット FHIR のオブジェクトが 1 つのみである場合、マッピング・テーブルには、ソース・リスト・フィールドに対するマッピング・エントリが 2 つ含まれます。

- 一方のマッピングは、ソース・リスト・フィールドをターゲットの単一値フィールドにマップします。このマッピングから生成される変換は、単に最初のリスト・エントリをターゲット・フィールドに配置します。
- もう一方のマッピングでは、オブジェクトの完全リストを格納するターゲット FHIR 拡張にソース・リスト・フィールドをマップします。FHIR 拡張の URL は完全なソース・フィールド名です。これはリソース名を含みますが、ハイフンで区切ったすべて小文字のテキストを使用します。

FHIR から SDA の場合 : 受信 FHIR にオブジェクトのリストがあり、SDA のオブジェクトが 1 つである場合、変換では最初のオブジェクトが使用され、他はすべて削除されます。

9.3.5.8 SDA CodeTableDetail の FHIR コードへのマッピング

変換により、SDA **`CodeTableDetail`** (またはそのサブクラスの 1 つ) は、以下のように **`Coding`** や **`CodeableConcept`** などの FHIR コード化オブジェクトにマップされます。

1. Code 値は code フィールドに割り当てられます。
2. Description は display フィールドに割り当てられます。
3. OriginalText フィールドが存在する場合は、text フィールドにマップされます。

9.3.5.9 ルックアップ・テーブルを使用したコード値の FHIR へのマッピング

マッピングは、ルックアップ・テーブルを参照して、このマッピングにおいてソース・スキーマ (SDA または FHIR) のコード値をターゲット・スキーマ (FHIR または FHIR DSTU2) のコード値にマップするエントリを見つけます。

マッピングでルックアップ・テーブルが見つからない場合、またはルックアップ・テーブルに一致エントリが見つからず、空以外の既定値が定義されている場合、既定値がコード・フィールドに適用されます。それ以外の場合、ターゲットはこのマッピングから値を受け取りません。

マッピングが SDA から FHIR で、ソース・フィールドに空以外の値が含まれる場合は、規約により、このソース・フィールドに対してマッピング・エントリが 2 つ存在します。両方のエントリは、同じ [このフィールドの設定条件] に従って実行されます。

- 一方のエントリは、ルックアップを実行して、ターゲット・フィールドに割り当てる値を取得します。
- もう一方は、元のソース・フィールドの値を文字列値の FHIR 拡張に格納します。

どちらの場合も、Description または OriginalText が Code 値と共に存在する場合、可能であれば FHIR にマップされます。

9.3.5.10 FHIR コードの SDA CodeTableDetail へのマッピング

FHIR プリミティブ・コードまたはコード化オブジェクト (**Coding**、**CodeableConcept** など) がコード値を FHIR から SDA に変換するためにルックアップを使用しない場合、以下のように SDA **CodeTableDetail** (またはそのサブクラスの 1 つ) に変換されます。

- CodeableConcept.text は HS.SDA3.CodeTableTranslated.OriginalText に変換されます。
- CodeableConcept.coding.display (または Coding.display) は HS.SDA3.CodeTableDetail.Description に変換されます。
- CodeableConcept.coding.code (または Coding.code、あるいは単に code) は HS.SDA3.CodeTableDetail.Code に変換されます。
- CodeableConcept.coding.system (または Coding.system) の GetCodeforURI は HS.SDA3.CodeTableDetail.SDACodingStandard に変換されます。
- CodeableConcept.coding.version (または Coding.version) は HS.SDA3.CodeTableDetail.CodeSystemVersionId に変換されます。

9.3.5.11 ルックアップ・テーブルを使用した FHIR コード値の SDA へのマッピング

FHIR から SDA 用のコード・ルックアップ・テーブルをマッピングに使用したい場合、マッピング・テーブルにソース・フィールドのマッピング・エントリが 2 つ含まれています。

- 2 つのエントリの一方は、ルックアップ・テーブルを参照して、FHIR のコード値を SDA の Code にマップするエントリを見つけます。
- このペアのもう一方のマッピング・エントリは、ルックアップ・テーブルのエントリが利用できないか一致を提供しない場合に処理を引き継ぎます。このエントリは、前述のようにソースの FHIR コード値 (変更なし) を SDA の **CodeTableDetail** オブジェクトにマップします。つまり、FHIR の code が **Coding** オブジェクトまたは **CodeableConcept** オブジェクトの内部にある場合は、FHIR の code、display、system、version、および text の値はすべて SDA **CodeTableDetail** フィールドに適切にマップされます。

9.3.5.12 SDACodingStandard のマッピング

変換で、SDA オブジェクトの `SDACodingStandard` プロパティが検出されると、`SDACodingStandard` 値が OID レジストリに存在するかが確認され、以下のいずれかが実行されます。

- `SDACodingStandard` 値が URL を含む OID レジストリのエントリである場合は、変換により、FHIR Coding リソースの `system` フィールドがその URL に設定されます。
- `SDACodingStandard` 値が URL を定義していない OID レジストリのエントリである場合は、変換により、FHIR Coding リソースの `system` フィールドがその OID に設定されます。
- `SDACodingStandard` 値が OID レジストリのエントリでない場合は、変換により、FHIR 拡張に値が保存されます。

9.3.5.13 文字列値の数値へのマッピング

ターゲットが FHIR で、文字列値が数値にマップされている場合、文字列に単位や命令などの非数値テキストが含まれることがあります。これを処理するため、ソース・リスト・フィールドに以下の 2 つのマッピング・エントリがあります。

- 2 つのエントリの一方は常に、ソース文字列値を、1 つの文字列値フィールドで構成される FHIR 拡張に割り当てます。
- もう一方のマッピング・エントリは、ソース文字列値をテストして数値かどうかを確認します。数値であれば、この数値をターゲットの数値フィールドにマップします。

9.3.5.14 FHIR コード・オブジェクトのマルチパート・リテラル値

Coding オブジェクトまたは **CodeableConcept** オブジェクトである一部の FHIR ターゲット・フィールドでは、リテラル値からの一連のマッピングにより、必要に応じてフィールドに割り当てられるマルチパート値が形成されます。このようなオブジェクトが格納できるフィールドの完全なセットは、`code`、`system`、`display`、`text`、`version`、および `userSelected` です。

これに該当する場合、このコードが、マルチパート・リテラル値を受け取る **Coding** オブジェクトまたは **CodeableConcept** オブジェクト内に存在することが、code フィールドの DTL アノテーション要素で説明されます。このコードに関連するリテラル値マッピングのセットはすべて、[このフィールドの設定条件] で同じ値を持つことが [FHIR アノテーション] に示されます。

9.3.5.15 FHIR 拡張へのマッピング

変換ターゲットが FHIR である場合、1 つ以上の SDA プロパティがターゲットの FHIR スキーマ内に対応するフィールドを持っていないことがあります。その場合、変換により、SDA データは FHIR 拡張にマップされます。この拡張の URL 接頭語は `http://intersystems.com/fhir/extn/sda3/lib` です。完全 URL は完全な SDA プロパティ名です。これはリソース名を含みますが、ハイフンで区切ったすべて小文字のテキストを使用します。

例えば、SDA プロパティ HS.SDA3.Administration:AdministeredAmount の FHIR 拡張は次のようになります。

- 拡張名 : `administration-administered-amount`
- FHIR 拡張の完全な URL : `http://www.intersystems.com/fhir/extn/sda3/lib/administration-administered-amount`

9.3.5.16 SDA CustomPairs のマッピング

変換では、タイプ HS.SDA3.SuperClass の SDA クラスの古い CustomPairs プロパティがサポートされます。

CustomPairs はタイプ HS.SDA3.NVPairs のオブジェクトのコレクションで、そのそれぞれに 2 つのプロパティ Name と Value があります。変換コードが顧客の SDA データ内でこのプロパティを検出し、ターゲットが FHIR である場合、Parameters リソースを格納する FHIR 拡張にコレクションはマップされます。この Parameters リソースは、ペアになったフィールド name と valueString のコレクションです。

以下の例では、カスタマイズされた SDA Encounter オブジェクトに、3 つのメンバを持つ SDA CustomPairs コレクションがあり、それぞれに `PlanOfCareInstructionsText` という名前が付いています。

```
{
   "resourceType": "Encounter",
   "contained":
      [
        {
          "resourceType": "Parameters",
          "id": "63",
          "parameter":
            [
              {
                "name": "PlanOfCareInstructionsText",
                "valueString": "Doctor recommends at least 30 minutes of exercise per day"
```

```
            },
            {
              "name": "PlanOfCareInstructionsText",
              "valueString": "Use sports heart rate monitor to aid in monitoring effort level"
            },
            {
              "name": "PlanOfCareInstructionsText",
              "valueString": "Read \"South Beach Diet\""
             }
          ]
      }
    ],
  "extension":
    [
      {
        "url": "http://intersystems.com/fhir/extn/sda3/lib/encounter-custom-pairs",
        "valueReference":
          {
            "reference": "#63"
          }
      }
    ],
  "id": "914"
}
```

9.4 変換のカスタマイズ

SDA-FHIR 間の各変換では、データ変換言語 (DTL) クラスを使用して、SDA オブジェクトを FHIR リソース (またはその逆) にマップします。DTL エディタを使用して、これらの DTL をカスタマイズできます。

より高度なカスタム変換動作を実装したい場合は、適切な変換 API クラスのサブクラスを作成し、そのメソッドをオーバーライドします。詳細は、"変換 API クラスのカスタマイズ" を参照してください。従来の FHIR の実装から新しいカスタマイズ・アーキテクチャへのアップグレードの詳細は、"従来の変換のアップグレード" を参照してください。

インターシステムズ製品はまた、変換で使用されるルックアップ・テーブルをカスタマイズするメカニズムも提供します。

各ネームスペースで異なるカスタマイズができるよう、インスタンス全体に対してではなく、特定のネームスペース内で変換をカスタマイズします。複数のネームスペースが同じようにカスタマイズした変換を持つようにするには、各ネームスペースでカスタマイズ・プロセスを繰り返す必要があります。

9.4.1 カスタム DTL の実装

SDA から FHIR に (またはその逆に) 変換するために使用される DTL をカスタマイズするストラテジには、標準 DTL のコピーの作成とその変更が含まれます。カスタム DTL のパッケージを手動で指定すると、変換により、標準の DTL ではなく、カスタム DTL が自動的に選択されます。

9.4.1.1 カスタム DTL のパッケージの指定

DTL をカスタマイズする前に、カスタマイズされるすべての DTL クラスのための単一のパッケージを指定する必要があります。クラス・パッケージには `HS.Local.FHIR.DTL` という名前を付けることをお勧めします。すべてのカスタム DTL に使用されるパッケージを決定したら、InterSystems ターミナルを使用して、このパッケージを指定する必要があります。カスタム DTL パッケージを指定するには、以下の手順に従います。

1. InterSystems ターミナルを開きます。
2. SDA-FHIR 変換を含むネームスペースに移動します。例を以下に示します。

   ```
   set $namespace = "Myfhirnamespace"
   ```

3. カスタム DTL パッケージが既に存在しているかどうかを確認するには、以下のように入力します。

   ```
   Write ##class(HS.FHIR.DTL.Util.API.ExecDefinition).GetCustomDTLPackage()
   ```

4. カスタム DTL パッケージがまだ存在しない場合は、以下のコマンドを入力します。HS.Local.FHIR.DTL はカスタム DTL パッケージの名前に置き換えます。

```
set status = ##class(HS.FHIR.DTL.Util.API.ExecDefinition).SetCustomDTLPackage("HS.Local.FHIR.DTL")
```

5. パッケージが正常に定義されたことを確認するには、以下のように入力します。

```
write status
```

応答は 1 になります。

9.4.1.2 カスタム DTL の作成

既存の標準の DTL のコピーを保存してから編集し、カスタム DTL を作成します。変換する際に、標準の DTL ではなく、カスタムの DTL を使用することがわかるように、カスタム DTL のパッケージと名前は名前付け標準に準拠する必要があります。カスタム DTL を作成するには、以下の手順に従います。

1. 管理ポータルを開き、FHIR ネームスペースに移動します。
2. [**相互運用性**] → [**リスト**] → [**データ変換**] の順に選択します。
3. カスタマイズする変換の名前を見つけます。例えば、SDA から FHIR STU3 への変換は、先頭に HS.FHIR.DTL.SDA3.vSTU3 が付きます。一方、FHIR STU3 から SDA への変換は、先頭に HS.FHIR.DTL.vSTU3.SDA3 が付きます。
4. カスタマイズする変換の名前をダブルクリックして、DTL エディタで開きます。
5. InterSystems ターミナルを開きます。
6. カスタマイズされた DTL クラスに必要な名前を取得するには、ターミナルで以下のように入力します。

```
Write ##class(HS.FHIR.DTL.Util.API.ExecDefinition).PreviewDTLCustomClass("standard_class_name")
```

standard_class_name は、カスタマイズしている変換のフルネーム (パッケージを含む) です。これは、DTL エディタで開いている変換の名前です。[**変換**] タブで名前を参照できますが、拡張子 .dtl は含まれていません。

7. ターミナルでの応答を必ずメモしてください。カスタマイズされた DTL クラスにこの名前を付ける必要があります。
8. DTL エディタで、[**名前を付けて保存**] をクリックします。
9. [**パッケージ**] フィールドで、ターミナルに表示されたカスタマイズされた DTL クラスの名前にあるパッケージを入力します。例えば、ターミナルに表示されたカスタマイズされたクラス名が HS.Local.FHIR.DTL.SDA3.vSTU3.Address.Address の場合、HS.Local.FHIR.DTL.SDA3.vSTU3.Address と入力します (実際のクラス名を除く)。
10. [**名前**] フィールドに、カスタマイズされたクラスの名前を入力します。例えば、ターミナルに表示されたカスタマイズされたクラス名が HS.Local.FHIR.DTL.SDA3.vSTU3.Address.Address の場合、Address と入力します。
11. 説明を入力し、[OK] をクリックします。

9.4.1.3 カスタム・クラスのミラー・メンバへのコピー

ご使用の環境でミラーリングを使用しており、カスタマイズのパッケージがミラーリングされていないデータベースに存在する場合、カスタマイズされた DTL クラスを、各ミラー・メンバのカスタム・パッケージにコピーする必要があります。例えば、カスタマイズされたクラスのパッケージを HS.Local.FHIR.DTL と定義した場合、HS.Local はミラーリングされていない HSCUSTOM ネームスペースに存在するため、このカスタマイズされた DTL クラスを、各ミラー・メンバの HS.Local.FHIR.DTL にコピーする必要があります。カスタム・パッケージがミラーリングされるデータベースに存在する場合、これ以上のアクションは必要ありません。

9.4.2 変換 API クラスのカスタマイズ

変換 API クラスには、カスタムの変換動作を実装するためにオーバーライドできる複数のメソッドが含まれています。メソッドをオーバーライドするには、`HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR` または `HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3` のサブクラスを作成し、カスタム・メソッドを記述します。例えば、条件に基づいて DTL を選択する場合は、`GetDTL` メソッドをオーバーライドできます。以下に、オーバーライド可能な変換メソッドの概要を簡単に説明します。

9.4.2.1 オーバーライド可能な SDA から FHIR への変換メソッド

HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR クラスの以下のメソッドは、カスタムの変換動作を実装するためにオーバーライドできます。

`GetDTL`

所定の SDA オブジェクトの変換に使用する DTL クラスを指定します。カスタム DTL を使用するためにこのメソッドをオーバーライドする必要はありません。カスタム DTL パッケージを指定すると、`GetDTL` メソッドにより、標準の DTL を使用する前に、カスタムの DTL が検索されます。ただし、条件に基づいて複数の候補から DTL を 1 つ選択する場合は、このメソッドをオーバーライドできます。

`IsDuplicate`

バンドル内の別のリソースによって参照される生成済みのリソースが既に存在するかどうかを変換で確認する方法を変更するには、このメソッドをオーバーライドします。例えば、Organization、Practitioner、または Medication などの共有リソースを重複として識別するための要件を緩和したい場合があります。既定では、共有リソースの先頭の 32 KB はハッシュ・テーブルにハッシュとして追加されます。それ以降の共有リソースへの各参照については、JSON の直接一致がないかハッシュ・テーブルを検索することにより、参照されているリソースが重複かどうかが変換によって判断されます。

`IsDuplicate` メソッドにより、参照されているリソースが既に存在すると判断された場合、そのリソースはバンドル出力には含まれません。

`ResourceLookup`

既定では、`ResourceLookup` メソッドが呼び出されると、変換によって作成されたバンドルのみを対象に、指定のリソースがないか検索されます。ただし、リポジトリやバンドル出力に指定のリソースがないかアプリケーションで検索したい場合などには、このメソッドをオーバーライドできます。

`GetReference`

別のストリームレットへの参照を含む SDA を変換する場合、このメソッドは、参照対象の SDA オブジェクト用に作成された FHIR リソースへの参照を返します。例えば、`EncounterNumber` がこのメソッドに渡されると、指定の `EncounterNumber` によって参照された SDA Encounter に対応する FHIR Encounter リソースへの参照が返されます。このメソッドをオーバーライドして、指定の FHIR リソースへのカスタム参照を生成します。

`GetId`

既定では、変換によってバンドルが生成される場合、個々のリソースに `id` は割り当てられません。`GetId` メソッドをオーバーライドして、バンドル内のリソースに `id` を割り当てます。この場合、バンドルの `fullUrl` フィールドの値は *`baseURL/resourceType/id`* で、バンドル内のリソース参照は *`resourceType/id`* になります。

GetBaseURL

GetBaseURL メソッドをオーバーライドして、各リソースの URL 接頭語を変更します。例えば、FHIR リソースを特定のリポジトリにポストする場合は、リポジトリを識別する URL 接頭語を指定できます。

HandleInvalidResource

変換では、各リソースをバンドル出力に追加する前に、それぞれのリソースを検証します。HandleInvaidResource メソッドをオーバーライドして、検証に失敗したリソースをどのように処理するかをカスタマイズします。既定では、エラーがスローされ、処理が停止します。つまり、バンドルは返されません。

9.4.2.2 オーバーライド可能な FHIR から SDA への変換メソッド

HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 クラスの以下のメソッドは、カスタムの変換動作を実装するためにオーバーライドできます。

GetDTL

所定の FHIR リソースの変換に使用する DTL クラスを指定します。カスタム DTL を使用するためにこのメソッドをオーバーライドする必要はありません。カスタム DTL パッケージを指定すると、GetDTL メソッドにより、標準の DTL を使用する前に、カスタムの DTL が検索されます。ただし、条件に基づいて複数の候補から DTL を 1 つ選択する場合は、このメソッドをオーバーライドできます。

GetResourceFromReference

このメソッドでは、バンドル内の別のリソースによって参照されているリソースを、変換で検索する場所を制御します。例えば、このメソッドをオーバーライドして、同じバンドル内ではなく、リポジトリ内で参照されているリソースを検索することができます。

GetSendingFacility

このメソッドをオーバーライドして、SDA SendingFacility プロパティの値を設定する方法をカスタマイズします。

既定では、SendingFacility プロパティは次のように設定されます。Patient の managingOrganization フィールドに Organization への参照が含まれており、その Organization が Bundle に存在する場合は、それが使用されます。そうでない場合は、患者識別子で MRN と割り当て機関が検索され、その割り当て機関が使用されます。いずれの項目も見つからない場合は、文字列 FHIR が使用されます。

GetIdentifier

このメソッドをオーバーライドして、特定の識別子が SDA プロパティに割り当てられる方法をカスタマイズします。

例えば、このメソッドをカスタマイズして、値を Encounter ストリームレットの EncounterNumber フィールドに割り当てることができます。この場合、変換されるリソースのコンテンツにアクセスすると、返すべき EncounterNumber を決定するのに役に立ちます。インスタンス・プロパティ %currentReference には、ダイナミック・オブジェクトとしてリソースを取得するために、インスタンス・メソッド GetResourceFromReference に渡すことができる FHIR 参照オブジェクトが含まれています。既定では、EncounterNumber プロパティの値は、1 から順に割り当てられます。

このメソッドのオーバーライドは、SDA HealthConcern または Goal の ExternalID 値の割り当てにも役立ちます。既定では、ExternalID プロパティの値は、1 から順に割り当てられます。

HandleMissingResource

既定では、受信 FHIR バンドル内で別のリソースによって参照されているリソースがバンドルに存在しない場合、バンドルの変換は続行されます。不明のリソースがバンドルにある場合の処理方法を変更するには、HandleMissingResource メソッドをオーバーライドします。

9.4.3 ルックアップ・テーブルのカスタマイズ

FHIR アノテーション・ツールを使用すると、変換でソース・データ形式のコードをターゲット形式のコードにマップするのに使用されるルックアップ・テーブルを調べることができます。これらのルックアップ・テーブルは、InterSystems ターミナル・ユーティリティを使用するか、ルックアップ・テーブルを含む JSON ファイルを手動で変更することにより、カスタマイズできます。

9.4.3.1 ターミナル・ユーティリティを使用したルックアップ・テーブルのカスタマイズ

インターシステムズでは、ルックアップ・テーブルのカスタマイズ・プロセスを導くユーティリティを用意しています。カスタマイズ・ユーティリティを実行するには、以下の手順に従います。

1. InterSystems ターミナルを開きます。
2. FHIR ネームスペースを変更するには、以下のように入力します。

   ```
   set $namespace = "Myfhirnamespace"
   ```

   Myfhirnamepace は、作成した FHIR ネームスペースです。
3. ユーティリティを開始するには、以下のように入力します。

   ```
   do ##class(HS.FHIR.DTL.Util.API.LookupTable).EditLookupTable()
   ```

4. カスタマイズしているルックアップ・テーブルのマッピング・ソースを入力します。例えば、値を SDA3 から STU3 へマッピングするルックアップ・テーブルをカスタマイズしている場合は、SDA3 と入力します。
5. カスタマイズしているルックアップ・テーブルのマッピング・ターゲットを入力します。例えば、値を SDA3 から STU3 へマップするルックアップ・テーブルをカスタマイズしている場合は、STU3 と入力します。
6. カスタマイズするルックアップ・テーブルのマッピング・ソースの値セットに対応する数を入力します。
7. マッピング・ソースの値セットを含むルックアップ・テーブルが 1 つだけ存在する場合、マッピング・ターゲットの値セットは自動的に選択され、次の手順に進むことができます。それ以外の場合、カスタマイズするマッピング・ターゲットの値セットに対応する数を入力します。
8. 編集するコードからコードへのマッピングを選択します。ルックアップ・テーブルに新しいコードからコードへのマッピングを追加するには、+ を入力します。
9. コードからコードへのマッピングのターゲットの値を編集している場合は、マッピングの新しいターゲット値を入力します。

   コードからコードへのマッピングのソースの値を編集する場合は、- を入力して、コードからコードへのマッピング全体を削除し、ユーティリティを再実行して、正しいソースおよびターゲットの値で新しいマッピングを追加する必要があります。

9.4.3.2 Lookup.json の編集によるルックアップ・テーブルのカスタマイズ

ターミナル・ユーティリティを使用するのではなく、変換で使用されるルックアップ・テーブルのすべてが含まれる JSON ファイルで、キー/値ペアを追加、削除、または編集することにより、ルックアップ・テーブルをカスタマイズできます。それを開始する前に、提供された Lookup.json ファイルのカスタム・コピーを作成して、カスタム・ディレクトリのネームスペース固有のディレクトリに配置する必要があります。

カスタム Lookup.json ファイルの作成

ルックアップ・テーブルにアクセスする際に変換で使用するカスタム JSON ファイルを永続的な場所に作成するには以下の手順に従います。

1. 使用している FHIR ネームスペース用のカスタム・ディレクトリを永続的な場所に作成します。
   - キットのインストールでは以下のように指定します。

     ```
     <install-dir>\dev\fhir\lookup\custom\<MYFHIRNAMESPACE>
     ```

   - コンテナでは以下のように指定します。

     ```
     <ISC_DATA_DIRECTORY>\dev\fhir\lookup\custom\<MYFHIRNAMESPACE>
     ```

   <MYFHIRNAMESPACE> は使用する FHIR ネームスペースの名前です。すべて大文字で記述します。例えば、FHIR プロダクションを含むネームスペースが `Myfhirnamespace` と呼ばれている場合、MYFHIRNAMESPACE というディレクトリを作成します。
2. <install-dir>¥dev¥fhir¥lookup ディレクトリから、作成した新しいカスタム・ネームスペース・ディレクトリに Lookup.json をコピーします。

   これで、新しい Lookup.json のコピーで、ルックアップ・テーブルの編集を開始できます。

カスタムの Lookup.json ファイルの編集

ルックアップ・テーブルのカスタマイズを開始するには、以下の 4 つの情報を収集する必要があります。

- マッピング・ソース
- マッピング・ターゲット
- マッピング・ソースの値セット
- マッピング・ターゲットの値セット

これらの値は、管理ポータルの [FHIR アノテーション] にあります。これらの値を確認するには、以下の手順に従います。

1. 管理ポータルを開き、FHIR ネームスペースに移動します。
2. ホーム・ページから、[Health]→[**スキーマ・ドキュメント**]→[**FHIR アノテーション**] を選択します。
3. 最初のドロップダウン・リストで、カスタマイズしているルックアップ・テーブルを含む変換のタイプを選択します。例えば、ルックアップ・テーブルでの SDA3 と FHIR STU3 のコードの相互マッピング方法が目的の場合は、[FHIR3->SDA3] を選択します。

   マッピング・ソースとマッピング・ターゲットをメモします。変換ペアの最初のインタフェース形式がマッピング・ソースです。2 番目のインタフェース形式がマッピング・ターゲットです。例えば、[FHIR3—>SDA3] を選択する場合、vSTU3 がマッピング・ソースで、SDA3 がマッピング・ターゲットです。
4. [<transformation> **ルックアップ・テーブルの表示**] ボタンをクリックします。ボタンのフルネームは選択した変換ペアによります。
5. [**ルックアップ・テーブルの表示**] ダイアログで、ドロップダウン・リストを使用してマッピング・ソースの値セットとマッピング・ターゲットの値セットをメモします。マッピング・ソースの値セットは左側のドロップダウン・リスト内の名前です。マッピング・ターゲットの値セットは右側のドロップダウン・リスト内の名前です。

これで、マッピング・ソース、マッピング・ターゲット、マッピング・ソースの値セット、およびマッピング・ターゲットの値セットを取得したので、カスタムの Lookup.json ファイルで適切なキー/値ペアを追加、削除、または編集することにより、ルックアップ・テーブルを編集できます。

Lookup.json の最上位のキー/値ペアは、マッピング・ソースからマッピング・ターゲットへのリレーションシップに対応します。例えば、SDA3 から FHIR STU3 への変換で使用されるルックアップ・テーブルは以下のようになります。

```
"SDA3" : {
    "vSTU3" : {
```

次のレベルのキー/値ペアは、マッピング・ソースの値セットからマッピング・ターゲットの値セットへのリレーションシップに対応します。対応するキー/値ペアを探すことで、正しいルックアップ・テーブルを検索します。以下に例を示します。

```
"HS.SDA3.Alert:Status" :
    {"event-status" : {
```

ルックアップ・テーブルを見つけたら、コードからコードへのマッピングに対応するキー/値ペアを追加、削除、または編集できます。

```
"A":"in-progress",
"C":"unknown",
"I":"aborted",
"INT":"completed"
```

カスタムの Lookup.json ファイルのロード

Lookup.json をカスタマイズしたら、SDA-FHIR 変換で使用する前に、ターミナルを使用してロードしておく必要があります。JSON ファイルをロードするには、以下の手順に従います。

1. ターミナルを開きます。
2. FHIR ネームスペースに移動します。例を以下に示します。

   ```
   set $namespace = "Myfhirnamespace"
   ```

3. 次のコマンドを実行します。

   ```
   set status = ##class(HS.FHIR.DTL.Util.API.LookupTable).ImportLookupJSONToGlobal()
   ```

10

FHIR クライアント

インターシステムズ製品には、標準の FHIR® クライアント・クラスが付属しており、スタンドアロンの ObjectScript アプリケーションや相互運用プロダクションはこれを使用して FHIR 要求を HTTP 経由で FHIR REST エンドポイントに送信したり、ローカルのインターシステムズ FHIR サーバに送信できます。要求を行うためにアプリケーションで使用されるメソッドは、アプリケーションで使用されている FHIR クライアント・クラスに関係なく、同じです。いずれの場合も、ユース・ケースに対応するクライアント・クラスをインスタンス化した後、アプリケーションで FHIR 相互作用またはオペレーションに対応するメソッドを呼び出します。

以下の 3 つのクライアント・クラスから選択できます。

HS.FHIRServer.RestClient.HTTP

FHIR 要求を HTTP 経由で FHIR エンドポイントに送信します。クラスをインスタンス化する際には、FHIR サーバのエンドポイントの URL はサービス・レジストリのエントリによって特定されます。

HS.FHIRServer.RestClient.FHIRService

FHIR 要求を同じネームスペース内のインターシステムズ FHIR サーバのサービスに送信します。クラスをインスタンス化する際には、インターシステムズ FHIR サーバはサーバのエンドポイント (`/fhirapp/fhir/r4` など) によって識別されます。

HS.FHIRServer.RestClient.Interop

相互運用プロダクションを使用して、FHIR 要求を HTTP 経由で FHIR エンドポイントに送信します。それには、以下の 2 つの方法があります。

- カスタム・ビジネス・ホスト内で生成された FHIR ペイロードを送信するか、ビジネス・ホストから FHIR データを取得する。
- HTTP 経由で送信する前に、スタンドアロン ObjectScript アプリケーションからの FHIR 要求を、相互運用プロダクションを通じてルーティングする。

この相互運用 FHIR クライアントの詳細は、"相互運用 FHIR クライアント" を参照してください。

これらのクラスはすべて単一のベース・クラス HS.FHIRServer.RestClient.Base から継承されます。このクラスには、FHIR クライアントが FHIR 相互作用やオペレーションを実行するために使用するメソッドのロジックが含まれています。各タイプの FHIR クライアントは、`CreateInstance` メソッドを使用してインスタンス化されます。

10.1 相互作用とオペレーション

FHIR 仕様の RESTful アーキテクチャ内で、FHIR クライアントは相互作用を通じてサーバ上のリソースを操作します。インターシステムズのテクノロジによって開発された FHIR クライアントには、これらの相互作用に対応するメソッドが用意されており、ObjectScript コードで単一のメソッド呼び出しとの相互作用を行うことを可能にしています。

FHIR クライアントは、各相互作用に 1 つ以上のメソッドを提供しますが、ある 1 つのメソッドを、FHIR サーバで実行されているオペレーションに関係なく提供します。このメソッドを起動してオペレーションを実行する方法の詳細は、クラス・リファレンスで Operation( ) を参照してください。

10.1.1 相互作用メソッドの呼び出し

FHIR クライアントが `update` などの相互作用を使用してサーバに書き込みを行う場合は、`SetRequestFormat` メソッドを使用して、サーバに書き込まれるペイロードの形式を指定する必要があります。使用可能な形式は、`JSON`、`XML`、`Form`、`XPatch`、および `Jpatch` です。同様に、FHIR クライアントは `SetResponseFormat` を使用して、FHIR サーバから返されるリソースの適切な形式を指定できます。 使用可能な形式は、`JSON` と `XML` です。

要求および応答の形式が個々の相互作用に合わせて変更されない限り、アプリケーションは要求や応答を一度設定すると、すべての相互作用メソッドに適用できます。例えば、要求を HTTP 経由で FHIR サーバに送信するスタンドアロンの FHIR クライアントは、クライアントのインスタンス化後すぐに、要求と応答の形式を設定できます。

```
Set clientObj = ##class(HS.FHIRServer.RestClient.HTTP).CreateInstance("MyFHIR.HTTP.Service")
Do clientObj.SetRequestFormat("JSON")
Do clientObj.SetResponseFormat("JSON")
```

FHIR クライアント・クラスがインスタンス化され、要求と応答の形式が設定されると、アプリケーションはサーバ上で実行したい FHIR 相互作用に対応するメソッドを呼び出すことができます。シグニチャなど、FHIR クライアントで利用可能な FHIR 相互作用メソッドを確認するには、クラス・リファレンスで HS.FHIRServer.RestClient.Base を参照してください。条件付きのアクションを許可する FHIR 相互作用には、2 つの異なるメソッドがあります。例えば、`update` 相互作用が条件付きであるかどうかに応じて、アプリケーションでは `Update` または `ConditionalUpdate` を呼び出すことができます。

引数として渡されるペイロードのデータ型は、インスタンス化された FHIR クライアントのタイプによって決まります。

- HTTP 経由で FHIR サーバにアクセスするクライアントの場合、ペイロードの引数は文字列またはストリームになります。
- ローカル・ネームスペースでインターシステムズの FHIR サーバにアクセスするクライアントの場合、ペイロードの引数は、文字列、ストリーム、またはダイナミック・オブジェクトにできます。

以下に、FHIR クライアントをインスタンス化し、外部 FHIR サーバ上で `read` 相互作用を実行する例を示します。

```
Set clientObj = ##class(HS.FHIRServer.RestClient.HTTP).CreateInstance("MyFHIR.HTTP.Service")
Do clientObj.SetResponseFormat("JSON")
Set clientResponseObj = clientObj.Read("GET", "Patient", "123")
```

10.1.2 カスタム・ヘッダの追加

API キーを渡す場合などで FHIR 要求にカスタム・ヘッダ情報が必要な場合は、SetOtherRequestHeaders() メソッドを使用します。このメソッドは、参照による多次元配列を入力として取ります。この配列では、各カスタム・ヘッダを添え字付き要素とします。配列にカスタム・ヘッダを取り込むには、次のように header name と header value を指定します。

ObjectScript

```
Set otherHeaders("X-API-Key") = "123"
Do clientObj.SetOtherRequestHeaders(.otherHeaders)
```

REST **clientObj** インスタンスで“他のヘッダ”コレクションをクリアするには ClearOtherRequestHeaders() API メソッドを使用します。このメソッドは引数を取りません。

ObjectScript

```
Do clientObj.ClearOtherRequestHeaders()
```

10.2 要求と応答のカスタマイズ

内部的には、各相互作用メソッドが 3 つのオーバーライド可能なメソッドを呼び出します。これらのメソッドをカスタマイズすることにより、要求の送信方法を変更したり、要求で受信される応答を操作できます。この 3 つのメソッド `MakeRequest`、`InvokeRequest`、および `MakeClientResponseFromResponse` は、ベース・クラスではなく、FHIR クライアントのタイプ別に実装されます。詳細は、FHIR クライアント・クラス (HS.FHIRServer.RestClient.HTTP、HS.FHIRServer.RestClient.FHIRService、または HS.FHIRServer.RestClient.Interop) のコメントを参照してください。

10.3 FHIR クライアント・クラスなしの要求

ObjectScript アプリケーションから内部 FHIR サーバへの要求を行う場合、FHIR クライアント・クラスの使用が推奨されますが、これらの標準クライアント・メソッドなしで、サーバ上で CRUD オペレーションを実行するカスタム・クラスを記述できます。例えば、サービスを経由しない (したがって、許可されている相互作用に関する制約をバイパスする)、FHIR サーバと相互作用するカスタム・クラスを記述できます。また、`DispatchRequest` メソッドを使用して、サービスに対して直接呼び出しを行うこともできます。このような特殊なケースの詳細は、"ObjectScript アプリケーション" を参照してください。

11

FHIR 要求と FHIR 応答

この章では、FHIR® サーバおよび FHIR 相互運用アダプタで使用される要求と応答について説明します。

インターシステムズの FHIR クライアントで使用される要求と応答の詳細は、"FHIR クライアント" を参照してください。

11.1 プロダクション以外の要求/応答

相互運用プロダクションを使用しない FHIR サーバの場合は、以下のようになります。

- FHIR 要求を渡すためにサーバ・アーキテクチャで使用されるメッセージ・クラスは、HS.FHIRServer.API.Data.Request です。
- サーバからの応答を要求元の FHIR クライアントに渡すために、サーバ・アーキテクチャで使用されるメッセージ・クラスは、HS.FHIRServer.API.Data.Response です。

11.1.1 FHIR ペイロードへのアクセス

既定では、REST ハンドラで FHIR 要求を受信すると、FHIR ペイロードが Request オブジェクト (`HS.FHIRServer.API.Data.Request`) の `Json` プロパティに保存されます。これにより、JSON 構造がダイナミック・オブジェクトに自動的に取り込まれます。XML が含まれる FHIR 要求は、JSON に変換された後に、`Json` プロパティのダイナミック・オブジェクトとして表されます。FHIR サーバからの応答 (`HS.FHIRServer.API.Data.Response`) にも FHIR データの `Json` プロパティは含まれます。

FHIR データを操作するには、まず、要求または応答の `Json` プロパティにアクセスします。FHIR ペイロードを入手したら、それをダイナミック・オブジェクトとして操作できます。例については、"FHIR データ" を参照してください。

11.2 相互運用の要求/応答

相互運用プロダクションを活用する FHIR サーバ、FHIR 相互運用アダプタ、または FHIR クライアントの場合は、以下のようになります。

- プロダクションを通じて FHIR 要求を渡すために使用されるメッセージ・クラスは、HS.FHIRServer.Interop.Request です。
- プロダクションを通じて応答を渡すために使用されるメッセージ・クラスは、HS.FHIRServer.Interop.Response です。

注釈　HS.FHIRServer.Interop.Response オブジェクトを構築する場合は、HTTP 応答の Content-Type ヘッダを作成するために ContentType プロパティを明示的に設定する必要があります。
HS.FHIRServer.API.Data.Response で ResponseFormatCode を設定するだけでは不十分です。

これらのクラスには、FHIR ペイロードを指す `QuickStreamId` プロパティが含まれます。

11.2.1 FHIR ペイロードへのアクセス

FHIR の実装で相互運用プロダクションが使用されている場合は、プロダクションが使用されていない実装とは異なり、メッセージ・オブジェクトの FHIR ペイロードにアクセスします。プロダクションベースの実装では、要求および応答メッセージ (HS.FHIRServer.Interop.Request および HS.FHIRServer.Interop.Response) には、FHIR ペイロードを含む QuickStream オブジェクトへのアクセスに使用される `QuickStreamId` が含まれています。相互運用要求メッセージには、`HS.FHIRServer.API.Data.Request` タイプの `Request` プロパティも含まれますが、この `Request` プロパティを FHIR ペイロードへのアクセスに使用することはできません。これは、その `Json` プロパティが一時的であるためです (相互運用応答についても、これは当てはまります)。その結果、FHIR ペイロードにアクセスする必要のあるプロダクションのビジネス・ホストは、`QuickStreamID` を使用して、ペイロードを取得する必要があります。

ペイロードが JSON 形式である場合、ビジネス・ホストは、ペイロードを変更するため、そのペイロードにアクセスしてダイナミック・オブジェクトに変換できます。例えば、BPL ビジネス・プロセスでは、以下のコードを使用して、JSON 形式の要求メッセージの FHIR ペイロードにアクセスし、変更することができます。

```
//Identify payload as a Patient resource and convert to dynamic object
if ((request.Request.RequestMethod="POST") & (request.Request.RequestPath="Patient")){
  set stream = ##class(HS.SDA3.QuickStream).%OpenId(request.QuickStreamId)
  set myPatient = ##class(%DynamicObject).%FromJSON(stream)

  // Modify Patient resource
  do myPatient.%Set("active", 0, "boolean")

  //Update payload with modified Patient resource
  do myPatient.%ToJSON(stream)
  do stream.%Save()
}
```

FHIR データをダイナミック・オブジェクトとして操作するその他の例は、"FHIR データ" を参照してください。

11.3 ObjectScript アプリケーション

ObjectScript アプリケーションがリソース・リポジトリからリソースを取得する必要がある場合は、プロダクション以外の要求オブジェクト (`HS.FHIRServer.API.Data.Request`) を構築してから、それをエンドポイントのサービスにディスパッチできます。アプリケーションでデータを取得する場合は、データはプロダクション以外の応答オブジェクト (`HS.FHIRServer.API.Data.Response`) として返されます。詳細は、"DispatchRequest の直接呼び出し" を参照してください。

12

FHIR データ

FHIR® サーバ・アーキテクチャでは、FHIR データはダイナミック・オブジェクトとして表されるため、データを操作するには、ダイナミック・オブジェクトの操作方法に関する知識と、FHIR リソースを JSON で表す方法に関する知識とを組み合わせて使用します。FHIR リソースの JSON 表現の詳細は、FHIR 仕様を参照してください。

FHIR ペイロードが JSON 形式である場合、相互運用要求や応答などでは、%FromJSON メソッドを使用してペイロードをダイナミック・オブジェクトに変換し、操作できるようにします。

12.1 FHIR データとダイナミック・オブジェクト

FHIR データはインターシステムズ製品内でダイナミック・オブジェクトとして表されることがよくあるため、ダイナミック・オブジェクトの操作方法に関する知識は必須です。以下のコードは、FHIR データを含むダイナミック・オブジェクトを操作する方法を示しています。ご覧のように、FHIR リソースの JSON 表現内のフィールドの構造を把握できる程度に、FHIR 仕様に精通している必要があります。ダイナミック・オブジェクトの処理方法の詳細は、"JSON の使用" を参照してください。

これらのコード・サンプルでは、FHIR Patient リソースを含むダイナミック・オブジェクトである変数 `patient` が存在することを前提としています。

値の検索

以下のコードでは、2 つの異なるアプローチを使用して、Patient リソースの識別子を検索し、特定のシステムを見つけます。このコードを記述するためには、FHIR 仕様に十分精通し、Patient リソースの JSON 構造に `system` の名前/値ペアを持つ `identifier` が含まれることを把握しておく必要があります。

```
// Put JSON representation of Patient resource into a dynamic object
set patient = ##class(%DynamicObject).%FromJSON("c:\localdata\myPatient.json")

//Searching for a identifier with a specific system
set mySystem = "urn:oid:1.2.36.146.595.217.0.1"

//Approach 1: Use an Iterator
if $isobject(patient.identifier) {
  set identifierIterator = patient.identifier.%GetIterator()
  while identifierIterator.%GetNext(, .identifier) {
    if identifier.system = mySystem {
      write "Found identifier: " _ identifier.value,!
    }
  }
}

//Approach 2: Use a 'for' loop
if $isobject(patient.identifier) {
  for i=0:1:patient.identifier.%Size()-1 {
    set identifier = patient.identifier.%Get(i)
    if identifier.system = mySystem {
```

```
        write "Found identifier: " _ identifier.value,!
      }
    }
  }
```

値の抽出

以下のコードの断片は Patient リソースから姓を抽出します。

```
if $isobject(patient.name) && (patient.name.%Size() > 0) {
  set myFamilyname = patient.name.%Get(0).family
}
```

値の変更

以下のコードの断片は Patient リソースの `active` フィールド (ブーリアン値) を `0` に設定します。

```
do patient.%Set("active", 0, "boolean")
```

新規 JSON オブジェクトの追加

新規 JSON オブジェクトを既存のダイナミック・オブジェクトに追加する場合は、ObjectScript 構文または JSON 構文のどちらを使用するのかを選択できます。例えば、以下のコードは、同じ結果をもたらす 2 つの異なるアプローチを使用して、新しい `identifier` を患者に追加します。

```
set mySystem = "urn:oid:1.2.36.146.595.217.0.1"
set myValue = "ABCDE"

// Approach 1: Use JSON syntax
if '$isobject(patient.identifier) {
  set patient.identifier = ##class(%DynamicArray).%New()
 }

do patient.identifier.%Push({
  "type": {
    "coding": [
      {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
        "code": "MR"
      }
    ]
  },
  "system": (mySystem),
  "value": (myValue)
})

//Approach 2: Use ObjectScript syntax
set identifier = ##class(%DynamicObject).%New()

set typeCode = ##class(%DynamicObject).%New()
set typeCode.system = "http://terminology.hl7.org/CodeSystem/v2-0203"
set typeCode.code = "MR"

set identifier.type = ##class(%DynamicObject).%New()
set identifier.type.coding = ##class(%DynamicArray).%New()
do identifier.type.coding.%Push(typeCode)
set identifier.system = mySystem
set identifier.value = myValue

if '$isobject(patient.identifier) {
  set patient.identifier = ##class(%DynamicArray).%New()
 }
 do patient.identifier.%Push(identifier)
```

12.2 データ・ロード・ユーティリティ

データ・ロード・ユーティリティは、ローカル・システム・ディレクトリ内に格納されているリソースとバンドルを、HTTP を経由するか、または HTTP を経由せずに FHIR サーバに直接送信します。データ・ロード・ユーティリティに提供するローカル FHIR データは、個々のリソース、バンドル、またはその両方にすることができ、JSON、XML、またはその両方で表

すことができます。このユーティリティの一般的な使用法は、大量の合成データをオープン・ソースの患者ジェネレータから FHIR サーバに提供することです。

可能な限り高速にデータを FHIR サーバに送ることが目的である場合は、HTTP を使用せずにデータをサーバに直接送信する方が効率的です。この場合、`FHIRServer` 引数を、サーバのエンドポイントと共にデータ・ロード・ユーティリティに渡します。例えば、サーバのエンドポイントが `/fhirapp/fhir/r4` で、FHIR バンドルが含まれるディレクトリが `c:\localdata` であるとします。この場合、データ・ロード・ユーティリティを実行するには、以下のように入力します。

```
Set status =
##class(HS.FHIRServer.Tools.DataLoader).SubmitResourceFiles("c:\localdata","FHIRServer","/fhirapp/fhir/r4")
```

ファイルの処理が終了すると、"`Completed Successfully`" と表示されます。表示されない場合は、"`Do $SYSTEM.Status.DisplayError(status)`" と入力するとエラーを表示できます。

または、`HTTP` を、サービス・レジストリの HTTP サービスの名前と共に渡すことで、すべてのバルク・データを HTTP 上で送信することもできます。HTTP サービスの作成の詳細は、"サービス・レジストリの管理" を参照してください。例えば、以下を実行できます。

```
Set status =
##class(HS.FHIRServer.Tools.DataLoader).SubmitResourceFiles("c:\localdata","HTTP","MyUniqueServiceName")
```

データ・ロード・ユーティリティは、オプションの引数を 3 つ取ります。進行状況を表示するかどうかを制御するオプション、統計をログに記録するオプション、またはディレクトリ内の処理対象ファイルの数を制御するオプションです。これらの引数の詳細は、HS.FHIRServer.Tools.DataLoader.SubmitResourceFiles() を参照してください。

13

FHIRPath

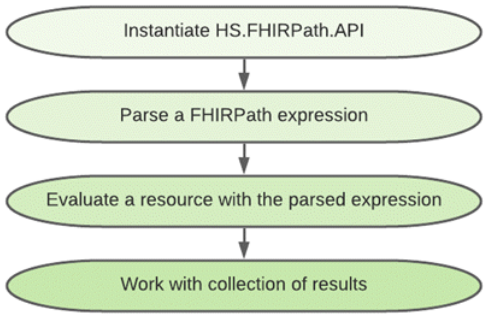
FHIRPath は XPath に似た言語です。これを使用すると、パス、関数、演算を含む簡単な構文を使用して、FHIR® リソースをナビゲートし、データを評価したり、そのフィールドからデータを抽出することができます。例えば、Patient の指定の名前に値が含まれているかどうか評価できます (`Patient.name.given.empty()`)。または、Patient リソースの `telecom` フィールドの値を抽出することもできますが、`official` が、その `use` フィールドの値である場合に限ります (`Patient.telecom.where(use = 'official')`)。

FHIRPath では、式はコレクション・ベースです。各関数は 1 つの入力コレクションに対して機能し、各二項演算子は 2 つの入力コレクションに対して作用し、式から返される値は出力コレクションにまとめられます。一部の関数や演算は、入力コレクションのサイズに制約を課します。

式の構築方法など、FHIRPath の詳細は、HL7 FHIRPath の仕様を参照してください。インターシステムズでは、仕様で定義されている関数と演算のサブセットをサポートしています。

13.1 ワークフロー

インターシステムズのテクノロジにより、FHIRPath を使用してリソースのデータを評価して抽出するプロセスは簡単に実行できます。



以下のセクションでは、ワークフローの各手順を詳細に説明します。

13.1.1 HS.FHIRPath.API のインスタンス化

FHIRPath を使用してリソースのデータを評価して抽出するプロセスは、HS.FHIRPath.API.getInstance() の呼び出しから始まります。このメソッドを呼び出す場合は、FHIR のバージョンに対応する FHIR パッケージを指定する必要がありま

す。例えば、評価しているリソースが FHIR R4 に準拠している場合、対応するパッケージの ID は、現在、hl7.fhir.r4.core@4.0.1 になります。この場合、HS.FHIRPath.API のインスタンス化は、次のようになります。

ObjectScript

```
set fhirPathAPI = ##class(HS.FHIRPath.API).getInstance($lb("hl7.fhir.r4.core@4.0.1"))
```

管理ポータルまたは ObjectScript を使用して、現在ロードされているパッケージの ID を取得できます。

- 管理ポータル ー [**ホーム**] → [Health] → [MyFHIRNamespace] → [**FHIR 構成**] に移動し、[**パッケージ構成**] カードを選択します。@ 記号とバージョン番号をパッケージの名前に追加すると、パッケージの ID になります。例えば、以下のパッケージの ID は hl7.fhir.r4.core@4.0.1 です。

- ObjectScript ー パッケージの ID をプログラムによってリストするには、"利用可能なパッケージのリスト" を参照してください。

HS.FHIRPath.API オブジェクトには、FHIRPath 式を解析し、リソースを評価するのに使用されるメソッドが含まれています。このオブジェクトはまた、FHIRPathAPI プロパティの HS.FHIRMeta.API オブジェクトにプロパティとしても含まれます。

13.1.2 FHIRPath 式の解析

HS.FHIRPath.API オブジェクトをインスタンス化したら、FHIRPath 式を解析する準備は完了です。式を解析するメソッド HS.FHIRPath.API.parse() は、リソースを評価するメソッドで使用されるツリー構造を返します。例えば、前のセクションで示したように、インスタンス化された fhirPathAPI という名前のオブジェクトがあるとした場合、以下のようになります。

ObjectScript

```
set tree = fhirPathAPI.parse("name.given.empty()")
```

13.1.3 リソースの評価

FHIRPath 式を解析したら、そのツリー構造を使用して、リソースのデータを評価または抽出できます。次の 2 つの評価メソッドを使用できます。

- HS.FHIRPath.API.evaluate() ー evaluate() メソッドは、多次元配列で評価結果を返します。
- HS.FHIRPath.API.evaluateToJson() ー evaluateToJson() メソッドは、ダイナミック配列でコレクションを返します。

どちらの場合も、評価されるリソースはダイナミック・オブジェクトとしてメソッドに渡されます。parse() メソッドによって返されたツリーもまた、引数として渡されます。例を以下に示します。

ObjectScript

```
set tree = fhirPathAPI.parse("name.given.empty()")
// myResource is a dynamic object
do fhirPathAPI.evaluate(myResource, tree, .OUTPUT)
set DynArray = fhirPathAPI.evaluateToJson(myResource, tree)
```

追加のメソッド HS.FHIRPath.API.evaluateArray() を使用して、evalaute() メソッドから返される多次元配列を解析できます。

13.1.4 結果の操作

evaluateToJson() によって生成されたダイナミック配列で結果を操作することにはメリットがあります。evaluate() によって生成された多次元配列には、他では入手できない追加情報が含まれます。以下に、多次元配列のデータに関する指針を示します。evaluate() に対する応答は OUTPUT という名前の変数で返されたものとします。

ノード	説明
OUTPUT	配列内の値を含むノード数。
OUTPUT(*n*)	配列の n 番目の要素の値。
OUTPUT(*n*,"t")	FHIR データ型の識別を含む、配列の n 番目の要素のデータ型。

返された多次元配列は、evaluateArray() メソッドを使用してさらに解析することができます。

対照的に、evaluateToJson() を使用してダイナミック配列を生成する場合、配列内の値を見ると、データ型が文字列、ブーリアン、数字、またはオブジェクトのどれであるかは判断できますが、FHIR データ型は判断できません。

13.1.5 ワークフローの例 : evaluate() メソッド

この例には、評価されるリソース、リソースの評価に必要な ObjectScript、および評価によって生成される多次元配列の確認が含まれます。

サンプル・リソース

```
set myResource = {
  "resourceType":"Patient",
  "telecom": [
    {
      "system": "phone",
      "value": "(03) 5555 6473",
      "use": "official"
    },
    {
      "system": "phone",
      "value": "(03) 5555 6473",
      "use": "home"
    },
    {
      "system": "email",
      "value": "myName@email.com",
      "use": "official"
    }
  ]
}
```

リソースからのデータの抽出

ObjectScript

```
set fhirVersion = $lb("hl7.fhir.r4.core@4.0.1")
set fhirPathAPI = ##class(HS.FHIRPath.API).getInstance(fhirVersion)
set tree = fhirPathAPI.parse("telecom.where(use = 'official')")
do fhirPathAPI.evaluate(myResource, tree, .OUTPUT)
```

多次元配列の表示

InterSystems ターミナルで zw OUTPUT コマンドを使用して、evaluate() から返された多次元配列を表示する場合、結果は以下のようになります。

```
OUTPUT=2
OUTPUT(1)={"system":"phone","value":"(03) 5555 6473","use":"official"}
OUTPUT(1,"t")="ContactPoint"
OUTPUT(2)={"system":"email","value":"myName@email.com","use":"official"}
OUTPUT(2,"t")="ContactPoint"
```

値は、ContactPoint FHIR データ型として識別されることに注意してください。

13.1.6 ワークフローの例 : evaluateArray() メソッド

この例では、上記の evaluate() 例の評価で生成された多次元配列を入力として使用し、結果として生じる配列の評価に必要な ObjectScript を示し、評価によって生成される多次元配列を表示します。

出力配列からのデータの抽出

ObjectScript

```
Merge INPUT = OUTPUT
Kill OUTPUT

Set tree = fhirPathAPI.parse("ContactPoint.value")

do fhirPathAPI.evaluateArray(.INPUT, tree, .OUTPUT)
```

多次元配列の表示

InterSystems ターミナルで zw OUTPUT コマンドを使用して、evaluateArray() から返された多次元配列を表示する場合、結果は以下のようになります。

```
OUTPUT=2
OUTPUT(1)="(03) 5555 6473"
OUTPUT(1,"t")="string"
OUTPUT(2)="myName@email.com"
OUTPUT(2,"t")="string"
```

値は、ObjectScript データ型 (文字列、ブーリアン、数字、またはオブジェクト) によって識別されることに注意してください。

13.1.7 ワークフローの例 : evaluateToJson() メソッド

この例には、評価されるリソース、リソースの評価に必要な ObjectScript、および評価によって生成されるダイナミック配列の確認が含まれます。

サンプル・リソース

```
set myResource = {
  "resourceType":"Patient",
  "name": [
    {
      "family": "Cooper",
      "given": [
        "James",
        "Fenimore"
      ]
    }]
}
```

リソースの評価

ObjectScript

```
set fhirVersion = $lb("hl7.fhir.r4.core@4.0.1")
set fhirPathAPI = ##class(HS.FHIRPath.API).getInstance(fhirVersion)
set tree = fhirPathAPI.parse("name.given.empty()")
set dynArray = fhirPathAPI.evaluateToJson(myResource, tree)
```

ダイナミック配列の表示

InterSystems ターミナルで `zw dynArray` コマンドを使用してダイナミック配列を表示する場合、結果は以下のようになります。

```
dynArray=[false]
```

13.2 関数

FHIRPath 仕様では、式に使用できる幅広い関数を定義しています。インターシステムズでは、これらの関数のサブセットをサポートします。

テーブル 13-1: サポートされている FHIRPath 関数

関数	例
[ ]（インデックス）	`Practitioner.name[1]`
aggregate	`item.factor.aggregate($total+$this,0)`
as	`Condition.abatement.as(string)`
children	`children().ofType(Reference)`
descendants	`descendants().ofType(Reference)`
empty	`name.empty()`
endsWith	`'abcdefg'.endsWith('efg')`
exists	`Patient.telecom.exists(system = 'phone')`
extension	`extension('http://intersystems.com/fhir/extn/sda3/lib/code-table-detail-care-provider-description').value as string`
hasExtension	指定した URL による拡張が入力コレクションのいずれかにある場合は `true` が返されます（FHIRPath v2.0.0 仕様には、この関数がありません）。
iif	`iif(1=1,2,3)`
indexOf	`'abcdefg'.indexOf('cd')`
is	`Condition.abatement.is(dateTime)`
not	`gender.not()`
ofType	`Bundle.entry.resource.ofType(Patient)`
resolve	`Organization.partOf.resolve()`
startsWith	`'abcdefg'.startsWith('abc')`

関数	例
substring	'abcdefg'.substring(1, 2)
union	Practitioner.name.family.union(Practitioner.id)
where	Patient.telecom.where(use = 'official')

13.3 演算

FHIRPath 仕様では、式に使用できる幅広い演算を定義しています。インターシステムズでは、これらの演算のサブセットをサポートします。

テーブル 13-2: サポートされている FHIRPath 演算

演算	例
+ (加算)	8 + 3 5 seconds + 3 seconds 'string1' + ' and ' + 'string2'
& (文字列連結)	'string1' & ' and ' & 'string2'
= (等しい)	Practitioner.name[0].family = 'Cooper' Practitioner.meta.versionId = 10
!= (等しくない)	Practitioner.name[1].family != 'Smith'
| (和集合)	Practitioner.name.family | Practitioner.id
and	true and false
as	実装の注意点を参照してください。
implies	Patient.name.given.exists() implies Patient.name.family.exists()
is	Practitioner.name[0] is HumanName
or	true or false
xor	true xor false

as に関する実装の注意点

FHIRPath 仕様によると、as 演算の左のオペランドは、単一項目のコレクションである必要があります。ただし、インターシステムズの FHIRPath の実装では、このコレクションは複数の値を持つことができます。例えば、ある Patient を参照する複数の拡張を持つ Observation があるとします。インターシステムズの FHIRPath の実装では、extension.value as Reference という式はまだ有効です。

13.4 パフォーマンスの向上

インターシステムズでは、解析した FHIRPath 式を格納できるメモリ内キャッシュを提供することにより、一連の式が頻繁に繰り返される場合のパフォーマンスを向上させます。キャッシュを有効にすると、キャッシュがクリアされるまで、parse() メソッドによって生成されたツリー構造が格納されます。

メモリ内キャッシュを有効にするには、以下を呼び出します。

```
do fhirPathAPI.enableCache(1)
```

キャッシュを無効にするには、以下を呼び出します。

```
do fhirPathAPI.enableCache(0)
```

14

FHIR サーバのセキュリティ

インターシステムズのセキュリティ・ストラテジと OAuth 2.0 を使用して、FHIR® サーバへの要求を行うことができるクライアントや、実行可能な相互作用を制御できます。同じ要求に両方の認証形式が指定されている場合、要求を正常に実行するには、その両方が有効である必要があります。

開発およびデバッグ時に、一時的にすべてのセキュリティ制限を無効にできます。

14.1 基本認証

既定では、FHIR サーバは基本認証を実施します。この認証では、インターシステムズ製品に対する資格情報を持つすべてのユーザは、REST 呼び出しのヘッダにその資格情報を含めることにより、FHIR サーバにアクセスすることができます。このセキュリティ戦略では、インターシステムズ製品内でのユーザの承認が問題になることはありません。認証済みのユーザは、FHIR サーバで CRUD 相互作用を実行できます。

14.1.1 承認要件の追加

基本認証に承認要件を追加することで、特定のセキュリティ・リソース (FHIR リソースとは関係ありません) を操作することが承認された InterSystems ユーザに、サーバ・アクセスを制限することができます。インターシステムズのセキュリティ用語では、リソースに対する特権を持つロールに属するユーザのみが、サーバで相互作用を実行することを承認されます。必要なリソースに対する書き込み特権を持つユーザは、FHIR サーバで作成、削除、更新、および条件付き更新の各相互作用を実行できます。リソースに対する読み取り特権を持つユーザは、書き込みアクセスが必要な相互作用を除くすべての相互作用を実行できます。FHIR トランザクションは再帰的であるため、トランザクション要求に書き込み相互作用と読み取り相互作用の両方が含まれる場合、ユーザは書き込み特権を保持する必要があります。

以下に、リソースを作成する方法、ロールのリソースに特権を割り当てる方法、およびロールにユーザを割り当てる方法の基本的な概要を示します。インターシステムズの承認の詳細は "承認ガイド" を、セキュリティの概要は "インターシステムズのセキュリティについて" を参照してください。

1. ユーザがサーバで相互作用を実行することを承認するかどうかを制御するリソースを作成するには、管理ポータルを開き、**[システム管理]** > **[セキュリティ]** > **[リソース]** に移動します。**[パブリック許可]** を [読み込み] に設定すると、認証済みのすべてのユーザが読み取り相互作用を実行できるようになります。詳細は、"リソースの作成または編集" を参照してください。
2. リソースに対する特権を持つロールを作成するには、**[システム管理]** > **[セキュリティ]** > **[ロール]** に移動します。一般的には、読み取りアクセスが必要なユーザ用のロールと、書き込みアクセスが必要なユーザ用の別のロールの 2 つのロールがあります。詳細は、"ロールの作成" を参照してください。
3. ロールに特権を付与するには、以下の手順に従います。

a. ロールの [**一般**] タブの [**権限**] セクションで、[**追加**] をクリックします。

b. サーバの承認を制御するリソースを選択して、[OK] をクリックします。

c. 新しい特権の横にある [**編集**] をクリックします。

d. リソースに対してロールに付与する許可を選択します。

詳細は、"ロールに対する新しい特権の付与" を参照してください。

4. これで、セキュリティ・リソースに対する許可を持つロールが作成されたので、[**メンバ**] タブを選択して、それらの許可を付与するユーザを追加します。詳細は、"現在のロールに対するユーザまたはロールの割り当て" を参照してください。

14.1.1.1 サーバの構成

ユーザが FHIR 相互作用を実行できるかどうかを制御するセキュリティ・リソースを作成または選択したら、以下の手順を使用して、このリソースを要求するようにサーバを構成します。

1. 管理ポータルで、[Health] → [**FHIR 構成**] → [**サーバ構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. FHIR サーバのエンドポイントを選択します。
3. [**編集**] を選択します。
4. [**必須のリソース**] フィールドで、FHIR サーバへのアクセスを制御するセキュリティ・リソースの名前を入力します。
5. [**更新**] を選択します。

14.2 OAuth 2.0 認証

FHIR サーバを OAuth 2.0 リソース・サーバとして設定することにより、OAuth 2.0 承認サーバから取得した有効なアクセス・トークンがない限り、クライアントの FHIR 要求を拒否できるようになります。FHIR 要求のアクセス・トークンは 2 回チェックされます。REST ハンドラによって 1 回、FHIR サーバの内部サービスに到達したときにもう 1 回です。アクセス・トークンは要求が REST ハンドラに達したときに強制されるため、要求は、相互運用プロダクションに到達したときには (FHIR サーバがプロダクションを使用するように構成されている場合)、既に検証済みです。REST ハンドラとサービスは同じクラスを使用してトークンを検証します。これは、サーバの Interactions クラスの `OAuth2TokenHandlerClass` パラメータで指定されたクラスです。既定の FHIR サーバでは、このトークン処理クラスは HS.FHIRServer.Util.OAuth2Token です。

FHIR サーバをリソース・サーバとして識別する最初の手順は、[**システム管理**] → [**セキュリティ**] → [OAuth 2.0] → [**クライアント**] を使用して、クライアント構成を作成することです。OAuth 2.0 承認サーバ用にサーバ・デスクリプションを作成したら、FHIR サーバに新しいクライアント構成を作成し、リソース・サーバ・タイプと指定します。インターシステムズ製品にリソース・サーバを設定する方法の詳細は、"OAuth 2.0 リソース・サーバとしての InterSystems IRIS Web アプリケーションの使用法" を参照してください。

FHIR サーバのクライアント構成を定義したら、以下の手順に従います。

1. 管理ポータルで、[Health] → [**FHIR 構成**] → [**サーバ構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. FHIR サーバのエンドポイントを選択します。
3. [**編集**] を選択します。
4. [OAuth **クライアント名**] フィールドに、管理ポータルで定義したリソース・サーバの**アプリケーション名**を入力します。

5. [更新] を選択します。

14.2.1 アクセス・トークンのスコープ

このセクションでは、FHIR サーバが、要求と共に渡される OAuth 2.0 アクセス・トークンのスコープを適用する方法について説明します。FHIR サーバでスコープを別に解釈する必要がある場合は、FHIR サーバをカスタマイズし、`OAuth2TokenHandlerClass` パラメータをオーバーライドしてカスタム・トークン処理クラスを指定する必要があります。

基本の処理

要求に付随するアクセス・トークンには、少なくとも 1 つの患者臨床スコープまたはユーザ臨床スコープが含まれている必要があります。含まれていないと、要求は HTTP 403 エラーで拒否されます。アクセス・トークンに患者臨床スコープとユーザ臨床スコープの両方が含まれている場合は、FHIR サーバでは、患者臨床スコープを適用し、ユーザ臨床スコープを無視します。

患者臨床スコープ/患者コンテキスト値

アクセス・トークンに患者臨床スコープが含まれている場合は、Patient リソースの ID である患者コンテキスト値 (“起動コンテキスト”とも呼ばれる) も含まれている必要があります。この患者コンテキスト値により、指定の Patient リソースや関連リソースへのアクセス権が付与されます。ほとんどの場合、患者臨床スコープは、関連するリソースへの明示的なアクセス権を付与する必要があります。例えば、患者コンテキスト値が `1234` で、患者臨床スコープが `patient/Observation.*` の場合、FHIR サーバは ID `1234` の患者を参照する Observation に対するアクセス権を付与できます。この場合には、`patient/Observation.*` (または、Observations へのアクセス権を付与する別のスコープ) が必要になります。この要件の例外として、FHIR クライアントは、そのリソースに固有の患者臨床スコープを取得せずに、複数の Patient 間で共有されるリソースにアクセスできます。例えば、スコープが `patient/Patient.read` である場合、クライアントは、スコープ `patient/Organization.read` を取得せずに、患者によって参照される Organization にアクセスできます。

アクセス・トークンに患者コンテキスト値が含まれている場合は、患者臨床スコープも含まれている必要があります。含まれていないと、要求は HTTP 403 エラーで拒否されます。

アクセス・トークンから患者コンテキスト値を取得するには、FHIR サーバで、2 つの場所を以下の順序で調べます。

- `launch/patient/` スコープの 3 番目の空白以外の部分。
- アクセス・トークンの `patient` プロパティ。

検索

FHIR サーバは、有効なアクセス・トークンを伴う検索要求を、以下の方法で処理します。

- `_include` パラメータと `_revinclude` パラメータを使用できます。
- FHIR サーバが患者コンテキスト値を適用する場合 :
  - 連鎖および逆連鎖した (`_has`) パラメータは使用できません。
  - 親のスコープで検索リソース・タイプを許可しておく必要があります。
  - 検索リソース・タイプが Patient ではない場合、患者コンテキストにある Patient リソースを参照検索パラメータ値で指定する必要があります。
  - `_include` パラメータと `_revinclude` パラメータが存在する場合は、スコープで許可されているリソースに、これらのパラメータで引き出し処理のみを指定する必要があります。
  - Patient 検索では、あらゆる `_id` が患者コンテキスト値と一致している必要があります。

- Patient コンパートメント検索では、コンパートメントのリソース ID は患者コンテキスト値と一致する必要があります。
- 他のすべてのケースでは、検索を実行し、得られた結果セットにあるすべてのリソースがスコープとコンテキストで許可されていることを確認します。

`Create` 相互作用

新しい Patient リソースの作成要求には、書き込み権限を付与するユーザ臨床スコープ (`user/Patient.write` または `user/Patient.*`) が含まれている必要があります。患者臨床スコープを含む Patient リソースに `create` 相互作用を実行することはできません。患者臨床スコープには、患者コンテキスト値が含まれている必要があり、`create` 相互作用には、リソース ID を含めることはできません。

`$everything`

Patient または Encounter の `$everything` オペレーションに対する要求には、その要求で返される可能性があるリソースすべてに対する読み取りアクセス権を持つアクセス・トークンが含まれている必要があります。リソースがスコープの範囲ではないコンパートメントで見つかった場合は、要求全体が HTTP 403 Forbidden エラーで拒否されます。

この要件は、現実的には以下のように適用されます。

- `_type` オペレーション・クエリ・パラメータが指定されている場合、スコープに、要求されるリソース・タイプすべてに対する読み取りアクセスが含まれる必要があります。
- タイプが指定されておらず、アクセス・トークンで患者臨床スコープが使用されている場合、コンパートメントで見つかったリソースを返すには、`patient/*.read` または `patient/*.*` のスコープが必要です。
- タイプが指定されておらず、アクセス・トークンでユーザ臨床スコープが使用されている場合、コンパートメントで見つかったリソースを返すには、`user/*.read` または `user/*.*` のスコープが必要です。

14.3 認証を使用しない場合

ライブ FHIR サーバでは認証は必須ですが、開発やテストの際に FHIR サーバに資格情報を入力するよう強制されると、煩わしいことがあります。認証と承認のストラテジを一時的に無視して、すべての FHIR 要求がサーバに届くようにすることができます。認証なしアクセスを許可するには、以下の手順に従います。

1. 管理ポータルで、[Health] → [**FHIR 構成**] → [**サーバ構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. FHIR サーバのエンドポイントを選択します。
3. [**編集**] を選択します。
4. [**デバッグ**] セクションの [**認証なしアクセスを許可**] チェック・ボックスにチェックを付けます。
5. [**更新**] を選択します。

15

FHIR サーバのデバッグ

インターシステムズでは、開発時の FHIR® サーバのデバッグに役立つデバッグ・モードとログを提供しています。

15.1 FHIR サーバのデバッグ

FHIR サーバをデバッグ・モードにすると、開発時に問題を解決するのに役立つほか、FHIR 要求を認証する必要が一時的になくなります。デバッグ・オプションを設定するには、以下の手順に従います。

1. 管理ポータルで、[Health] → [**FHIR 構成**] → [**サーバ構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. FHIR サーバのエンドポイントを選択します。
3. [**編集**] を選択します。
4. [**デバッグ**] セクションで、有効にするデバッグ・オプションのチェック・ボックスにチェックを付けます。
   - [**認証なしアクセスを許可**] ー 認証と承認のストラテジを無視して、すべての FHIR 要求がサーバに届くようにすることができます。
   - [**新しいサービス・インスタンス**] ー すべての FHIR 要求について、新しいサービス・オブジェクトをインスタンス化します。Interactions や InteractionsStrategy サブクラスなどのカスタムのアーキテクチャ・クラスに変更を加える場合は、このオプションを設定します。
   - [**トレースバックを含む**] ー FHIR サーバは、OperationOutcome リソースでスタック・トレースを送信することによって、FHIR 要求に応答します。
5. [**更新**] を選択します。

15.2 ログ

FHIR サーバには、次の 2 種類のログが用意されています。

- 内部 FHIR サーバのログ ー どのクラス・メソッドが呼び出されているかなど、FHIR サーバ・アーキテクチャがどのように FHIR 要求を処理しているかに関する情報を提供します。
- HTTP 要求のログ ー REST クライアントから FHIR サーバへの HTTP 要求に関する情報を提供します。

15.2.1 内部 FHIR サーバのログ

FHIR サーバは、サーバが受信する FHIR 要求をアーキテクチャが処理する様子についての基本的なログ情報を提供します。この情報には、呼び出されるクラス・メソッド、SQL 関連メッセージ、および _include 検索がどのように処理されているかなどが含まれます。このタイプのログを有効にするには、以下の操作を実行します。

1. InterSystems ターミナルを開きます。
2. FHIR サーバのネームスペースに移動します。例えば、以下のように入力します。

```
set $namespace = "FHIRNamespace"
```

3. 保存するログ情報のタイプを指定するグローバル ^FSLogChannel を作成します。グローバルを作成するための構文は以下のとおりです。

```
set ^FSLogChannel(channelType) = 1
```

   channelType は以下のいずれかになります。

   - Msg ー ステータス・メッセージをログに記録します。
   - SQL ー SQL 関連の情報をログに記録します。
   - _include ー _include パラメータおよび _revinclude パラメータを使用する検索に関連する情報をログに記録します。
   - all ー 3 つのタイプの情報すべてをログに記録します。

   例えば、すべてのタイプの情報のログを有効にするには、以下のように入力します。

```
set ^FSLogChannel("all") = 1
```

注釈　新しいタイプのログ情報 (例えば、Msg から SQL) に切り替えるには、既存の ^FSLogChannel グローバルを削除してから、もう一度新しい channelType で設定します。

15.2.1.1 ログの表示

FHIR サーバ・アーキテクチャのログを有効にすると、ログ・エントリが ^FSLOG グローバルに保存されます。管理ポータルを使用してログを表示するには、[**システムエクスプローラ**] → [**グローバル**] に移動して、FSLOG グローバル (FSLogChannel ではない) を表示します。FHIR サーバのネームスペースにいることを確認します。

グローバルの各ノードは以下のような構造になっています。

CurrentMethod^CurrentClass|LogType|LogMessage

例えば、^FSLOG グローバルのノードのログ・エントリは、以下のようになる場合があります。

```
"runQuery^HS.FHIRServer.Storage.Json.Interactions|SQL|Parameters: (2)"
```

15.2.1.2 ログの無効化

FHIR サーバ・アーキテクチャのログを無効にするには、単に ^FSLogChannel グローバルを削除するか、このグローバルを 0 に設定します。例えば、ターミナルで以下のように入力できます。

```
kill ^FSLogChannel
```

15.2.2 HTTP 要求のログ

HTTP 要求のログが有効な場合、FHIR クライアントから要求を受信する REST ハンドラは、各 HTTP 要求に関する情報を ISCLog グローバルに書き込みます。このタイプのログを有効にするには、以下の操作を実行します。

1. InterSystems ターミナルを開きます。
2. 任意のネームスペースから、以下のコマンドを入力して、グローバル ^%ISCLog を構成し、HTTP 要求のログ記録を開始します。

```
set ^%ISCLOG=5
set ^%ISCLOG("Category","HSFHIR")=5
set ^%ISCLOG("Category","HSFHIRServer")=5
```

15.2.2.1 ログの表示

HTTP 要求のログが有効になったら、ログ・エントリが ^ISCLOG グローバルに保存されます。このグローバルは %SYS ネームスペースにあります。

管理ポータルを使用してログを表示するには、[**システムエクスプローラ**] → [**グローバル**] に移動して、ISCLOG グローバル (%ISCLOG ではない) を表示します。%SYS ネームスペースにいることを確認します。

15.2.2.2 ログの無効化

HTTP 要求のログを無効にするには、ターミナルを開いて以下のコマンドを入力します。

```
set ^%ISCLOG=1
```

15.2.3 FHIR テスト・ユーティリティ

管理ポータルに表示される FHIR テスト・ユーティリティ ([Health] → [**FHIR テスト・ユーティリティ**]) は、現在の FHIR アーキテクチャでは機能しません。このユーティリティは、今でも、従来の FHIR テクノロジで機能します。

16

FHIR サーバのメンテナンス

運用中の FHIR® サーバを保守する間、エンドポイントに対する FHIR 要求の処理を停止し、保守が完了したらエンドポイントを再度有効にしなければならないことがあります。

エンドポイントを停止して再開するには、以下の手順に従います。

1. 管理ポータルで、[Health] → [**FHIR 構成**] → [**サーバ構成**] に移動します。FHIR サーバのネームスペースにいることを確認します。
2. FHIR サーバのエンドポイントを選択します。
3. [**編集**] を選択します。
4. FHIR サーバのエンドポイントを要求に利用できるようにするには、[**構成**] セクションの [**有効**] チェック・ボックスにチェックを付けます。エンドポイントを停止して要求を拒否するには、チェック・ボックスのチェックを外します。

17

FHIR サーバのカスタマイズ

FHIR® サーバを使用する場合、FHIR サーバの動作をカスタマイズするためのストラテジが 2 つあります。従来の FHIR テクノロジのように、相互運用プロダクションでロジックを使用して、サーバの動作を変更できます。ただし、FHIR サーバのアーキテクチャをカスタマイズして、カスタムの機能を実装するオプションもあります。このオプションは、相互運用プロダクションを使用しない FHIR サーバは、これを使用する FHIR サーバよりも著しく高速にできるため重要です。

サーバ・アーキテクチャをカスタマイズする場合は、サーバの環境に固有な部分のみをカスタマイズする、リソース・リポジトリの拡張が最も一般的です。稀なケースとして、FHIR サーバに完全なカスタム・バックエンドを記述する必要がある場合があります。FHIR サーバのアーキテクチャにより、このような柔軟性が与えられています。リソース・リポジトリを拡張するのか、カスタム・バックエンドを記述するのかに関係なく、FHIR サーバのカスタマイズ・プロセスはインストール前のサブクラスの作成から始まります。

FHIR サーバの一部の動作は、アーキテクチャのカスタマイズを必要としない構成オプションを通じて制御されます。これらのオプションの詳細は、"FHIR サーバの構成" を参照してください。

FHIR サーバをカスタマイズするときに、サーバの機能宣言書を更新できます。詳細は、"機能宣言書の変更" を参照してください。

17.1 インストール前のサブクラスの作成

FHIR サーバをカスタマイズするには、まず、IDE を使用してアーキテクチャのサブクラスを作成し、いくつかのパラメータを定義します。InteractionsStrategy はインストール時に指定されるため、この手順は、サーバのエンドポイントがインストール・プロセスで作成される前に実行する必要があります。

通常、FHIR サーバはリソース・リポジトリのアーキテクチャを拡張しています。このような場合は、IDE とサブクラスを開きます。

- `HS.FHIRServer.Storage.Json.Interactions`
- `HS.FHIRServer.Storage.Json.InteractionsStrategy`
- `HS.FHIRServer.Storage.Json.RepoManager`

リソース・リポジトリを使用する代わりに FHIR サーバの完全なカスタム・バックエンドを記述している場合は、次のアーキテクチャのスーパークラス `HS.FHIRServer.API.Interactions`、`HS.FHIRServer.API.InteractionsStrategy`、および `HS.FHIRServer.API.RepoManager` のサブクラスを作成します。

17.1.1 サブクラスのパラメータ

IDE を使用して Interactions、InteractionsStrategy および RepoManager サブクラスを作成したら、InteractionsStrategy と RepoManager の以下のパラメータを変更する必要があります。

スーパークラス	サブクラスのパラメータ
`HS.FHIRServer.API.InteractionsStrategy`	・ `StrategyKey` ― InteractionsStrategy の一意の識別子を指定します。 ・ `InteractionsClass` ― Interactions サブクラスの名前を指定します。
`HS.FHIRServer.API.RepoManager`	・ `StrategyClass` ― InteractionsStrategy サブクラスの名前を指定します。 ・ `StrategyKey` ― InteractionsStrategy の一意の識別子を指定します。InteractionsStrategy サブクラスの `StrategyKey` パラメータと一致する必要があります。

サブクラスをコンパイルしたら、FHIR サーバをインストールする準備が整います。あとは、インストール中に、InteractionsStrategy サブクラスの名前を指定するだけです。

17.2 カスタム・コードの有効化

開発時にカスタムの Interactions または InteractionsStrategy コードに変更を加える場合は、次の FHIR 要求が行われるときに、[**新しいサーバ・インスタンス**] デバッグ・オプションを使用して新しいコードを有効化します。詳細は、"FHIR サーバのデバッグ" を参照してください。

17.3 リソース・リポジトリのカスタマイズ

リソース・リポジトリの FHIR サーバ・アーキテクチャのサブクラスを作成したら、サーバをカスタマイズする準備は完了です。通常、カスタマイズには、`HS.FHIRServer.Storage.Json.Interactions` のサブクラスのメソッドとパラメータのオーバーライドが含まれます。以下に、リソース・リポジトリを使用する FHIR サーバに実行できる最も一般的なカスタマイズについて説明します。

テーブル 17-1: カスタマイズのクイック・スタート

目標	HS.FHIRServer.Storage.Json.Interactions のサブクラスのアクション
特定の FHIR 相互作用のカスタマイズ	相互作用に対応するメソッドをオーバーライドします。
すべての要求の処理	OnBeforeRequest をオーバーライドして、ユーザに透過的なロジックを実装します。このオーバーライドされたメソッドには、スーパー・クラスの呼び出しが含まれる必要があります。例えば、Do ##super(pFHIRService, pFHIRRequest, pTimeout) のようになります。 FHIR クライアントに要求の処理が異なることを認識させる場合は、カスタムの FHIR オペレーションを作成します。
すべての要求の後処理	OnAfterRequest をオーバーライドして、ユーザに透過的なロジックを実装します。このオーバーライドされたメソッドには、スーパー・クラスの呼び出しが含まれる必要があります。例えば、Do ##super(pFHIRService, pFHIRRequest, .pFHIRResponse) のようになります。 FHIR クライアントに要求の処理が異なることを認識させる場合は、カスタムの FHIR オペレーションを作成します。
Read 相互作用の結果の後処理	PostProcessRead をオーバーライドします (例)。
Search 相互作用の結果の後処理	PostProcessSearch をオーバーライドします (例)。
カスタムの FHIR オペレーションの追加	OperationHandlerClass パラメータをオーバーライドして、HS.FHIRServer.Storage.BuiltInOperations のサブクラスの名前を指定します。"カスタムの FHIR オペレーション" を参照してください。
バンドルの処理方法のカスタマイズ	BatchHandlerClass パラメータをオーバーライドして、カスタム・クラスの名前を指定します。既定のハンドラ・クラスは HS.FHIRServer.DefaultBundleProcessor です。
OAuth トークンの処理方法のカスタマイズ	OAuth2TokenHandlerClass パラメータをオーバーライドして、カスタム・クラスの名前を指定します。既定のハンドラ・クラスは HS.FHIRServer.Util.OAuth2Token です。

以下のコード・サンプルでは、リソース・リポジトリを使用する FHIR サーバに実行できるカスタマイズをいくつか示します。

17.3.1 結果の後処理

Read 相互作用や Search 相互作用の結果を操作したいことはよくあります。例えば、Read 相互作用によって返された Patient のデータを変更したり、検索結果から特定のリソースを除外したい場合などです。以下の例では、結果は同意ルールに基づいて変更されます。このサンプル・コードでは、同意処理に対応するために別個のクラスが記述されていることを前提としています。要求から抽出されるロールは InterSystems セキュリティ・ロールです。

Class Definition

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.Json.Interactions
{
Property RequestingUser As %String [ Private, Transient ];
Property RequestingUserRoles As %String [ Private, Transient ];

Method OnBeforeRequest(pFHIRService As HS.FHIRServer.API.Service,
                       pFHIRRequest As HS.FHIRServer.API.Data.Request,
                       pTimeout As %Integer)
{
  //Extract the user and roles for this request
  //so consent can be evaluated.
  set ..RequestingUser = pFHIRRequest.Username
  set ..RequestingUserRoles = pFHIRRequest.Roles
}
Method OnAfterRequest(pFHIRService As HS.FHIRServer.API.Service,
                      pFHIRRequest As HS.FHIRServer.API.Data.Request,
                      pFHIRResponse As HS.FHIRServer.API.Data.Response)
{
  //Clear the user and roles between requests.
  set ..RequestingUser = ""
  set ..RequestingUserRoles = ""
}
Method PostProcessRead(pResourceObject As %DynamicObject) As %Boolean
{
  //Evaluate consent based on the resource and user/roles.
  //Returning 0 indicates this resource shouldn't be displayed - a 404 Not Found
  //will be returned to the user.
  if '##class(MyCustom.Consent).Consented(pResourceObject,
                                          ..RequestingUser,
                                          ..RequestingUserRoles) {
    return 0
  }

  //Modify (anonymize) the resource being returned to the client if they don't have
  //permission to see the full record.
  if (pResourceObject.resourceType = "Patient") &&
  ##class(MyCustom.Consent).Anonymize(..RequestingUser, ..RequestingUserRoles) {
    do pResourceObject.%Remove("name")
  }
  return 1
}
Method PostProcessSearch(pRS As HS.FHIRServer.Util.SearchResult,
                         pResourceType As %String) As %Status
{
  //Iterate through each resource in the search set and evaluate
  //consent based on the resource and user/roles.
  //Each row marked as deleted and saved will be excluded from the Bundle.
  do pRS.%SetIterator(0)
  while(pRS.%Next()) {
   set resourceObject = ..Read(pRS.ResourceType, pRS.ResourceId, pRS.VersionId)
   if '##class(MyCustom.Consent).Consented(resourceObject, ..RequestingUser,
                                           ..RequestingUserRoles)
   {
     do pRS.MarkAsDeleted()
     do pRS.%SaveRow()
   }
  }
  do pRS.%SetIterator(0)
  quit $$$OK
}

}
```

Tipヒント FHIR サーバをカスタマイズする場合は、リソースが共有リソースかどうかを判断すると便利です。共有リソースには、Patient 情報は含まれません。FHIR の条件では、これらのリソース・タイプは Patient コンパートメントには存在しません。`IsSharedResourceType` メソッドを使用すると、リソースが共有されているかどうかを判断できます。例えば、カスタムの Interactions クラスには、以下の条件文を含めることができます。

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.Json.Interactions
Method OnBeforeRequest(pFHIRService As HS.FHIRServer.API.Service,
                       pFHIRRequest As HS.FHIRServer.API.Data.Request,
                       pFHIRResponse As HS.FHIRServer.API.Data.Response)
{
  If pFHIRService.Schema.IsSharedResourceType(pFHIRRequest.Type) {
    //Do x,y,z
}
```

17.3.2 カスタム ID のリソースへの割り当て

Create 相互作用を実行する際に、リソース・リポジトリ・サーバをカスタマイズして、各リソースにカスタム ID を割り当てることができます。以下の例では、リソースをリソース・リポジトリに格納するときに、ランダムな UUID をこのリソースに割り当てます。

Class Definition

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.Json.Interactions
{

Method Add(pResourceObj As %DynamicObject, pResourceIdToAssign As %String = "",
           pHttpMethod = "POST") As %String
{
  //Assign a random UUID for each new resource's ID, except for when processing an
  //Update as Create (when a user uses the PUT method and explicitly defines the ID).
  if pHttpMethod '= "PUT" {
    set pResourceIdToAssign = $zconvert($system.Util.CreateGUID(), "L")
  }
  return ##super(pResourceObj, pResourceIdToAssign, pHttpMethod)
}
}
```

17.4 機能宣言書の変更

FHIR サーバの機能宣言書は、サーバの動作方法を記したクライアント向けのメタデータです。FHIR クライアントは、機能宣言書を取得して、サーバが予期する内容と FHIR 要求の処理方法を判断できます。FHIR サーバをカスタマイズするときに、機能宣言書を更新して FHIR クライアントがサーバのサポート内容に関する正確な記述を保持できるようにすることもできます。機能宣言書の更新には、次の 2 つのオプションがあります。

- 既存の機能宣言書を取得し、その JSON を編集して、サーバに戻します。簡単ではあるものの、この手法には限界があります。例えば、新しい検索パラメータの追加などの特定のアクションによって機能宣言書は自動的に再生成されるため、これらのアクションのいずれかの実行後に、カスタマイズした機能宣言書の復元が必要になる場合があります。詳細は、"機能宣言書の手動更新" を参照してください。
- 機能宣言書を生成するメソッドをオーバーライドして、InteractionsStrategy サブクラスを変更します。これにより、機能宣言書に対する制御が向上するため、再生成されても問題は発生しません。詳細は、"機能宣言書メソッドのオーバーライド" を参照してください。

17.4.1 機能宣言書の手動更新

REST クライアントまたはプログラムを使用して FHIR サーバの機能宣言書を取得し、テキスト・エディタまたはサードパーティのツールを使用してこれを編集して、サーバを新しいバージョンに更新できます。新しい検索パラメータの追加などの特定のアクションの実行後に、この手順を繰り返す必要がある場合があることに注意してください。したがって、必要に応じ、作成し直すのではなく改訂した機能宣言書のコピーを保存しておくことをお勧めします。

以下の例では、InterSystems サーバの IP アドレスが `172.16.144.98`、スーパーサーバ・ポートが `52782`、エンドポイントのベース URL が `/fhirapp/r4` であると想定しています。

- REST クライアントで機能宣言書を取得するには、`GET` 要求を *`base-url`*`/metadata` に送信します。例を以下に示します。

  ```
  GET http://172.16.144.98:52782/fhirapp/r4/metadata
  ```

- プログラムを使用して機能宣言書を取得し、JSON ファイルとして保存するには、以下のように入力します。

```
set strategy = ##class(HS.FHIRServer.API.InteractionsStrategy).GetStrategyForEndpoint("/fhirapp/r4")
set interactions = strategy.NewInteractionsInstance()
set capabilityStatement = interactions.LoadMetadata()
do capabilityStatement.%ToJSON("c:\localdata\MyCapabilityStatement.json")
```

機能宣言書を変更したら、改訂したバージョンを InterSystems ターミナルからサーバにプログラムによって送信します。以下の例では、`/fhirapp/r4` はエンドポイントのベース URL で、`MyCapabilityStatment.json` は改訂したバージョンです。`{}.%FromJson` メソッドは JSON ファイルを取り、それをダイナミック・オブジェクトに配置します。

```
set strategy = ##class(HS.FHIRServer.API.InteractionsStrategy).GetStrategyForEndpoint("/fhirapp/r4")
set interactions = strategy.NewInteractionsInstance()
set newCapabilityStatement = {}.%FromJSON("c:\localdata\MyCapabilityStatement.json")
do interactions.SetMetadata(newCapabilityStatement)
```

17.4.2 機能宣言書のメソッドのオーバーライド

特定の FHIR サーバの動作を変更すると、機能宣言書が自動的に再生成されるため、手動更新するのではなく、サーバの機能宣言書の生成に使用するメソッドをオーバーライドすることもできます。これには、IDE の開発タスクが必要ですが、生成プロセスの制御は向上します。これらのタスクは、`HS.FHIRServer.Storage.Json.InteractionsStrategy` のサブクラスを作成して、リソース・リポジトリが拡張済みであると想定しています。このサブクラスでオーバーライドが必要となるメソッドは、サーバのパブリッシャのように基本のメタデータを編集するのか、サーバの機能の記述を変更するのかによって異なります。

機能宣言書内で、サーバ名など、サーバの基本メタデータを変更するだけの場合は、機能宣言書の生成元である JSON テンプレートを変更できます。この JSON テンプレートは、エンドポイントの InteractionsStrategy クラスの `GetCapabilityTemplate()` メソッドにあります。サーバのメタデータ文字列を変更するには、以下の手順に従います。

1. `HS.FHIRServer.Storage.Json.InteractionsStrategy` のサブクラスに `GetCapabilityTemplate` メソッドを作成し、そのメソッドをオーバーライドします。
2. `HS.FHIRServer.Storage.Json.InteractionsStrategy.GetCapabilityTemplate()` の内容をサブクラスの `GetCapabilityTemplate()` メソッドにコピーします。
3. メタデータ文字列を編集し、サブクラスをコンパイルします。
4. コンソール設定ユーティリティを使用して、機能宣言書を更新します。詳細は、"コマンド行オプション" を参照してください。

リソースでサポートされている相互作用など、機能宣言書の内容を変更する場合は、InteractionsStrategy の `GetMetadataResource()` メソッドをオーバーライドする必要があります。オーバーライド・メソッドで `##super` を呼び出して `HS.FHIRServer.Storage.Json.InteractionsStrategy.GetMetadataResource()` を起動し、メソッドから返される機能宣言書を後処理することを強くお勧めします。返された機能宣言書をダイナミック・オブジェクトとして変更します。例えば、サブクラスは次のようになります。

```
Class Pkg.MyInteractionsStrategy Extends HS.FHIRServer.Storage.Json.InteractionsStrategy
{
  Method GetMetadataResource()
   {
    set MyCapabilityStatement = ##super()
    // manipulate MyCapabilityStatement as a DynamicObject
    return MyCapabilityStatement
   }
}
```

機能宣言書を生成するメソッドをオーバーライドしたら、コンソール設定を使用して機能宣言書を更新してください。詳細は、"コマンド行オプション" を参照してください。

18

カスタムの FHIR オペレーション

FHIR® サーバは、RESTful アプローチではなく RPC に似たアプローチを使用して、FHIR クライアントからの要求に基づいて特殊な機能を実行する FHIR オペレーションをサポートしています。これらは、`$everything` のような標準の FHIR オペレーションである場合も、カスタム・オペレーションである場合もあります。リソース・リポジトリを使用または拡張する FHIR サーバでは、既に特定の標準の FHIR オペレーションをサポートしています (完全なリストは、"サポートされる相互作用とオペレーション" を参照してください)。

以下に、FHIR オペレーションを FHIR サーバに追加するプロセスの概要を示します。

1. FHIR サーバのアーキテクチャのサブクラスを作成します。詳細は、"インストール前のサブクラスの作成" を参照してください。
2. `HS.FHIRServer.API.OperationHandler` のサブクラスを作成します。リソース・リポジトリを使用している場合は、`HS.FHIRServer.API.OperationHandler` ではなく `HS.FHIRServer.Storage.BuiltInOperations` サブクラスを作成して、`$everything` などの既定のオペレーションが失われないようにします。ベスト・プラクティスとして、オペレーションごとに別個のサブクラスを作成してから、そのすべてを継承するマスタ・クラスを作成することをお勧めします。
3. Interactions サブクラスで、`OperationHandlerClass` パラメータの値をオーバーライドし、作成したオペレーション・サブクラスのクラス名にします。
4. オペレーション・ハンドラ・サブクラスの各オペレーションのメソッドを記述します。
5. CapabilityStatement リソースにオペレーションを追加します。

以下のセクションでは、プロセスの最後の 2 つの手順を詳細に説明します。

18.1 カスタム・オペレーションのメソッドの記述

FHIR サーバでサポートされるオペレーションは、オペレーション・ハンドラ・サブクラスのメソッドに直接対応します。これらのメソッドの名前は、以下の構文に適合する必要があります。

`FHIR`*`Scope`*`Op`*`OperationName`*

この構文内の変数は以下のとおりです。

- Scope では、FHIR クライアントがオペレーションを追加する先のエンドポイントのタイプを指定します。可能な値は以下のとおりです。
  - `System` — "ベース" の FHIR エンドポイントに追加するオペレーションを指定します (例えば、http://fhirserver.org/fhir)。これらのオペレーションは、サーバ全体に適用されます。

  - Type ― FHIR エンドポイントにリソース・タイプと共に追加するオペレーションを指定します (例えば、http://fhirserver.org/fhir/Patient)。これらのオペレーションは、指定されたリソース・タイプのすべてのインスタンスで動作します。
  - Instance ― リソースの特定のインスタンスを指す FHIR エンドポイントに追加するオペレーションを指定します (例えば、http://fhirserver.org/fhir/Patient/1)。これらのオペレーションは、リソースの特定のインスタンスでのみ動作します。

- OperationName は、FHIR クライアントがサーバの呼び出しに追加する $ オペレーションです。

以下の表に、メソッド名と FHIR クライアントによって呼び出されるオペレーションの相関関係の例を示します。

メソッド名	REST クライアントによるオペレーションの呼び出し
FHIRSystemOpMyoperation	http://fhirserver.org/fhir/$myoperation
FHIRTypeOpValidate	http://fhirserver.org/fhir/Observation/$validate
FHIRInstanceOpEverything	http://fhirserver.org/fhir/Patient/1/$everything

オペレーションにハイフン (-) が含まれる場合は、メソッド名からハイフンを削除してください。例えば、システム全体のオペレーションが $my-operation である場合、メソッドを FHIRSystemOPMyOperation という名前にします。

以下に $everything のメソッド・シグニチャの例を示します。

```
ClassMethod FHIRInstanceOpEverything(pService As HS.FHIRServer.API.Service,
  pRequest As HS.FHIRServer.API.Data.Request,
  pResponse As HS.FHIRServer.API.Data.Response)
```

18.2 機能宣言書へのオペレーションの追加

FHIR サーバの機能宣言書には、サーバがサポートするすべてのオペレーションが含まれる必要があります。機能宣言書を新しいオペレーションで更新するには、次の 2 つの選択肢があります。

- 機能宣言書にオペレーションを手動で追加します。このアプローチには欠点が 1 つあり、新しい検索パラメータを追加したときなど、場合によっては機能宣言書が再生成されるため、手動による変更が再生成時に失われます。このプロセスの詳細は、"機能宣言書の手動更新" を参照してください。
- オペレーション・ハンドラ・サブクラスの AddSupportedOperations メソッドを変更して、機能宣言書の再生成時に、新しいオペレーションがその機能宣言書に自動的に追加されるようにします。このアプローチの詳細は、以下のセクションを参照してください。

以下の 2 段階の手順を使用して、新しいオペレーションを機能宣言書に自動的に追加できます。

1. オペレーションを、オペレーション・ハンドラ・サブクラスの AddSupportedOperations メソッドに追加します。コマンド行ユーティリティでサーバの機能宣言書を生成した場合、サポートされるオペレーションはこのメソッドから取得されます。例えば、$everything オペレーションをサポートするサーバのオペレーション処理クラスに、以下のようなメソッドが含まれます。

   ```
   ClassMethod AddSupportedOperations(pMap As %DynamicObject)
     {
       Do pMap.%Set("everything","http://hl7.org/fhir/OperationDefinition/patient-everything")
     }
   ```

オペレーション処理クラスのスーパークラスに既に何らかのオペレーションが含まれる場合は、必ず、サブクラスの AddSupportedOperations 内でそのスーパークラスの AddSupportedOperations メソッドを呼び出してください。例えば、オペレーション処理サブクラスのメソッドは以下のようになる場合があります。

```
ClassMethod AddSupportedOperations(pMap As %DynamicObject)
  {
    Do ##class(HS.FHIRServer.MySuperclass.Validate).AddSupportedOperations(pMap)
    Do pMap.%Set("everything", "http://hl7.org/fhir/OperationDefinition/patient-everything")
  }
```

各オペレーションのサブクラスと、そのすべてを継承するマスタ・クラスを作成している場合は、そのマスタ・クラスが、各オペレーションのサブクラスの AddSupportedOperations メソッドを呼び出すようにします。

2. コマンド行ユーティリティを使用して、機能宣言書を再生成します。

   a. InterSystems ターミナルから、FHIR サーバのネームスペースに変更します。以下に例を示します。

      ```
      set $namespace = "MyFHIRNamespace"
      ```

   b. インストールおよび構成ユーティリティを実行します。

      ```
      do ##class(HS.FHIRServer.ConsoleSetup).Setup()
      ```

   c. オプション Update the CapabilityStatement Resource を選択します。

   d. 構成するエンドポイントを選択します。

   e. 選択内容を確認します。

19

インターシステムズ FHIR クライアントのバイパス

サーバ側のアプリケーションを使用して内部 FHIR サーバに対して FHIR® 要求を行う場合、アプリケーションでは、通常、標準の FHIR クライアントが使用されます。これらの組み込みクラスの使用方法の詳細は、"FHIR クライアント" を参照してください。

ただし、カスタムの ObjectScript クラスを使用して、サービスを通じて要求せずに、リポジトリを操作できるようになりたい場合があります。例えば、サーバで要求を読み取り専用の相互作用に制限している場合でも、書き込みオペレーションを行いたい場合があります。その場合は、サービスをバイパスできます。

その他のケースでは、FHIR クライアントと REST ハンドラで使用されるのと同じメソッドを、ただし、カスタム・クラスから使用したい場合があります。詳細は、"`DispatchRequest` の直接呼び出し" を参照してください。

ObjectScript アプリケーションでは、リソースを検証することもできます。

19.1 サービスのバイパス

サーバ側アプリケーションは、サービス経由でプログラムによる要求を送信する代わりに、Interactions サブクラスのメソッドを直接呼び出すことができます。例えば、アプリケーションから、POST 要求をサービスに送信するのではなく、Interactions サブクラスの `Add` メソッドを直接呼び出すことができます。これは特に、サービスによって禁止されているアクションをサーバ側アプリケーションで実行する必要がある場合に便利です。例えば、サーバのメタデータでエンドポイントが読み取り専用として構成されている場合、サービスに対するプログラムによる要求でリソースを作成することはできません。ただし、サーバ側アプリケーションは、Interactions サブクラスへのメソッド呼び出しを使用し、リソースによってストレージ・ストラテジを更新すると、サービスによって強制されている制約を実質的にバイパスできます。

プログラムによる Interactions クラス・メソッドの呼び出しでは、FHIR データはダイナミック・オブジェクトとして渡されます。

19.2 `DispatchRequest` の直接呼び出し

ObjectScript アプリケーションは、`DispatchRequest( )` を直接呼び出すことにより、FHIR クライアントとしても動作できます。このメソッドは、標準の FHIR クライアントと内部 FHIR サーバの REST ハンドラによって使用されます。

19.2.1 GET リソース

ObjectScript アプリケーションは、サーバのサービスを使用して、リソースを取得できます。例えば、178.16.235.12 がインターシステムズ・サーバの IP アドレス、52783 がスーパーサーバ・ポートであるとした場合、REST 呼び出しは以下のようになります。

```
GET http://178.16.235.12:52783/fhirapp/namespace/fhir/r4/patient/1
```

ObjectScript を使用した同一のエンドポイントへのアクセスは、以下のようになります。

```
set url = "/fhirapp/namespace/fhir/r4"
set fhirService = ##class(HS.FHIRServer.Service).EnsureInstance(url)
set request = ##class(HS.FHIRServer.API.Data.Request).%New()
set request.RequestPath = "/Patient/1"
set request.RequestMethod = "GET"
do fhirService.DispatchRequest(request, .response)
```

この例では、応答は、ダイナミック・オブジェクトに表される JSON 応答を使用したデータ・オブジェクト (HS.FHIRServer.API.Data.Response) になります。

注釈　サーバに対する最初の要求で、EnsureInstance メソッドを呼び出して、FHIR サービスをインスタンス化する必要があります。要求の前に毎回この呼び出しを実行しても問題は発生しませんが、サービスが変更されているかどうかをチェックするのにごくわずかな時間がかかります。

19.2.2 POST リソース

プログラムによって FHIR サーバにデータを送信することもできます。以下の例では、アプリケーションで、ファイル MyPatient.json の JSON オブジェクトに記述される Patient リソースを作成するとします。ObjectScript コードは以下のようになります。

```
set url = "/csp/fhirapp/namespace/fhir/r4/"
set fhirService = ##class(HS.FHIRServer.Service).EnsureInstance(url)
set request = ##class(HS.FHIRServer.API.Data.Request).%New()
set request.RequestPath = "/Patient"
set request.RequestMethod = "POST"
set request.Json = {}.%FromJSON("c:\resources\MyPatient.json")
do fhirService.DispatchRequest(request, .response)
```

この例では、要求に格納されている JSON のソースは、外部ファイルではなくアプリケーション内のダイナミック・オブジェクトから取得された可能性があります。

19.3 FHIR データの XML としての処理

REST クライアントを使用して FHIR サーバで CRUD オペレーションを実行すると、FHIR サーバは、FHIR データを自動的に受け入れるか、受信要求に基づいて FHIR データを XML として返します。ただし、カスタムの ObjectScript クラスからプログラムによって CRUD オペレーションを実行する場合、FHIR サービスに送信されるすべてのデータは JSON 形式である必要があります。同様に、サービスによって返されるデータもすべて JSON 形式になります。FHIR サーバには、XML を JSON に、および JSON を XML に変換するためのヘルパー・メソッドが用意されています。

XML データを FHIR サービスに送信するには、XML をストリーム・オブジェクトに配置し、それを HS.FHIRServer.Service.StreamToJSON() メソッドに送信して、形式が XML になるように指定します。例えば、以下のコードは、XML ペイロードを、FHIR サービスに渡すことができる JSON 要求に変換します。

```
set url = "/csp/fhirapp/namespace/fhir/r4/"
set fhirService = ##class(HS.FHIRServer.Service).EnsureInstance(url)
set request = ##class(HS.FHIRServer.API.Data.Request).%New()
set request.Json= fhirService.StreamToJSON(MyStream,"XML")
```

FHIR サービスからの JSON 応答を XML に変換するには、HS.FHIRServer.Util.JSONToXML.JSONToXML() メソッドを使用します。

19.4 FHIR データのストリームとしての処理

HS.FHIRServer.Service.StreamToJSON() メソッドは、XML または JSON ストリームを JSON オブジェクトに変換し、それを要求の一部として FHIR サービスに渡すことができるようにします。FHIR サービスは、ストリームを直接処理することはできません。このメソッドは、ストリーム、およびストリーム内のデータの形式の 2 つの引数を受け付けます。例えば、以下のコード行は、JSON ストリームを、FHIR サービスに送信できる JSON オブジェクトに変換します。

```
set url = "/csp/fhirapp/namespace/fhir/r4/"
set fhirService = ##class(HS.FHIRServer.Service).EnsureInstance(url)
set request = ##class(HS.FHIRServer.API.Data.Request).%New()
set request.Json= fhirService.StreamToJSON(MyStream,"JSON")
```

XML ストリームの場合は、単に `XML` を 2 つ目の引数として渡します。

19.5 FHIR リソースの検証

リソースがダイナミック・オブジェクトとして表される限り、ObjectScript アプリケーションは、FHIR `$validate` オペレーションを使用せずに、FHIR サーバのメタデータに対し、プログラムを使用してリソースを検証することができます。例えば、以下のコードは Patient リソースをサーバの FHIR R4 メタデータに対して検証します。このメタデータには、Patient リソースのスキーマが含まれています。`LoadSchema` メソッドを呼び出すと、FHIR バージョンの一般名 (`R4` や `STU3` など) や、サーバのベース・メタデータ名 (`HL7v40` または `Hl7v30` など) を指定することができます。

```
// Put JSON representation of Patient resource into a dynamic object
set patient = ##class(%DynamicObject).%FromJSON("c:\localdata\myPatient.json")

//Validate the patient resource
set schema = ##class(HS.FHIRServer.Schema).LoadSchema("R4")
set resourceValidator = ##class(HS.FHIRServer.Util.ResourceValidator).%New(schema)

do resourceValidator.ValidateResource(patient)
```

20

バルク FHIR コーディネータ

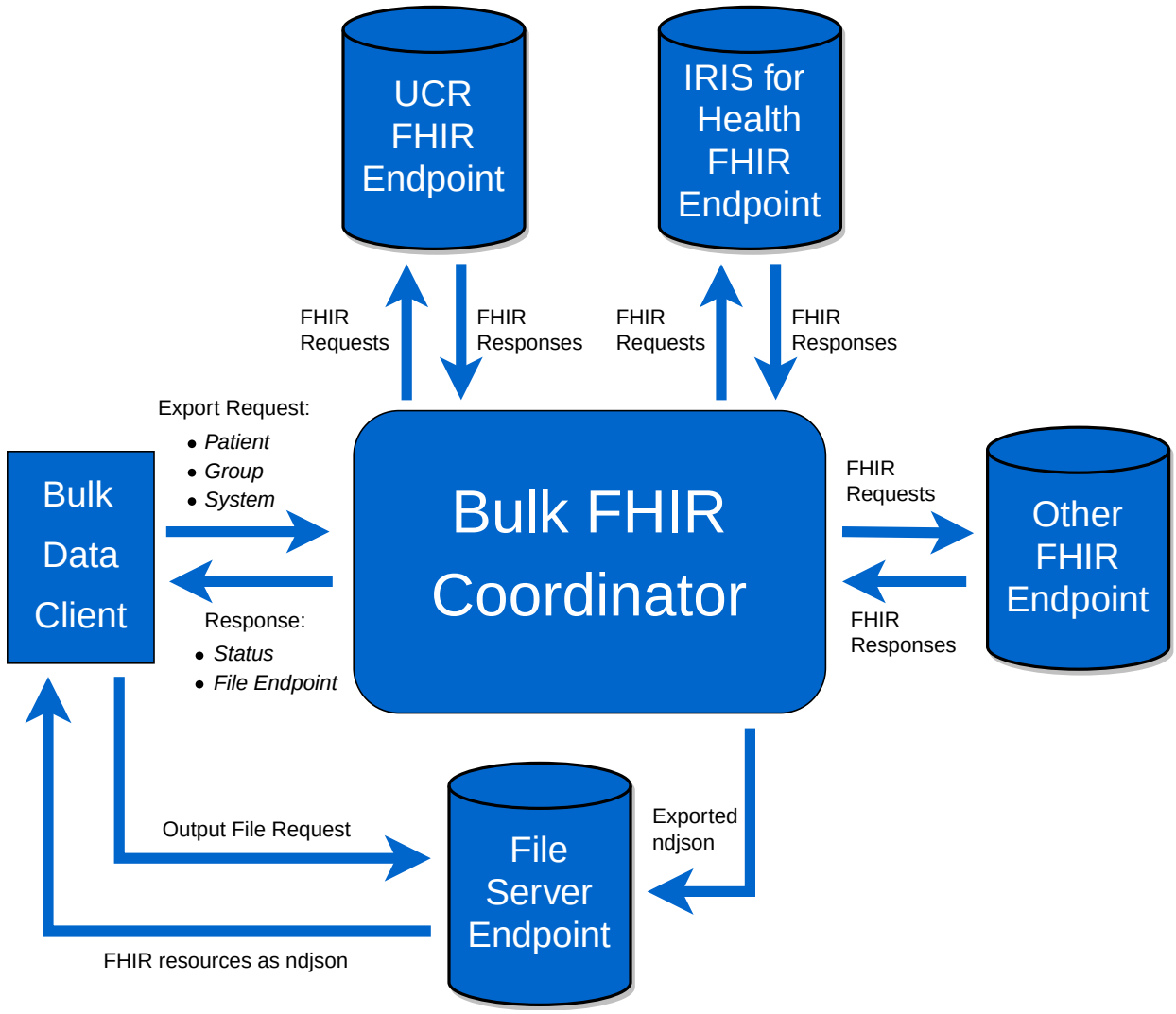
一般的なFHIR相互作用が、特定の患者に関する固有の情報を検索するのに対して、HL7® FHIR® bulk data 相互作用は、FHIRリソースサーバから患者全体にわたる大量のデータを抽出します。バルク FHIR の一般的な用途として、検査コホートの特定、公衆衛生、EHR から他の EHR へのデータ転送などがあります。

20.1 バルク FHIR コーディネータの概要

クライアントとの FHIR バルク・データの相互作用を簡潔にしてバルク・データ要求で FHIR サーバが過負荷にならないように、バルク・データ・クライアントと FHIR リソース・サーバ・エンドポイントの間では、インターシステムズのバルク FHIR コーディネータ (BFC) によってバルク・データ要求が仲介されます。ネイティブではバルク・データの相互作用をサポートしていない FHIR リソース・サーバへのバルク FHIR のエクスポートを、バルク FHIR コーディネータで容易にすることができます。

クライアントに代わり、バルク FHIR コーディネータが FHIR エンドポイントに FHIR リソースを要求し、それを受け取る場合、この処理はエクスポートとされます。

クライアントと FHIR エンドポイントとの相互作用をバルク FHIR コーディネータが仲介する様子を以下の図に示します。



バルク FHIR コーディネータと相互作用するには、BFC のホーム・ページを使用するか、バルク・データの REST クライアントを介します。

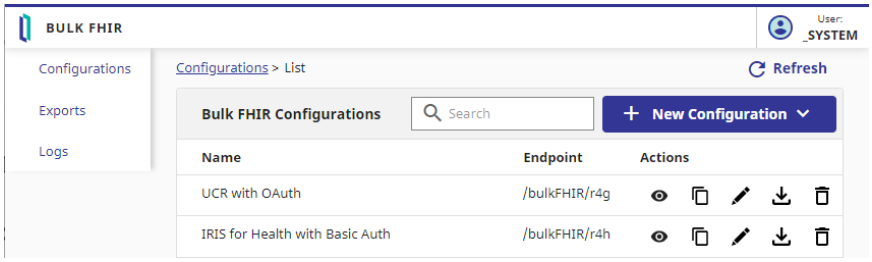
- BFC のホーム・ページでは一組の構成を入力できます。バルク FHIR の構成のそれぞれで FHIR リソース・サーバのエンドポイントを特定し、承認のタイプやファイルの場所など、バルク・データの相互作用で使用する各種パラメータを定義します。すべてのリソースを返すことができて、患者とグループのエクスポートで Patient/$everything オペレーションをサポートするあらゆるシステムを FHIR エンドポイントとして使用できます。例えば、InterSystems IRIS for Health、InterSystems HealthShare Health Connect、InterSystems HealthShare Unified Care Record があります。

  このホーム・ページでは、エクスポートの開始、エクスポート・ステータスの確認、エクスポートしたファイルのダウンロード、エクスポート・ログの確認もできます。

- REST クライアントを使用すると、HL7® FHIR® Bulk Data Export 仕様の説明にある各要求を実行できます。各バルク FHIR 構成は、次の 2 つのエンドポイントを提供します。
  - バルク FHIR エンドポイント ー エクスポートとステータスをサポートし、要求を削除します。
  - ファイル・ストレージ・エンドポイント ー ファイルのダウンロード要求をサポートします。

20.2 バルク FHIR コーディネータのホーム・ページ

バルク FHIR 構成を操作するには、[ホーム] → [Health] → [foundationNamespace] → [バルク FHIR コーディネータ] に移動します。



構成ごとに名前と一意のエンドポイントがあります。[構成] → [リスト] ページのアイコンを使用して、以下の各アクションを実行します。

アイコン	アクション	注
	新規構成	[編集] ページを開くか、JSON ファイルをインポートすることで、新しい構成を作成できます。
👁	エクスポートの表示	構成の [エクスポート] ビュー・ページが開き、進行中または完了したバルク FHIR エクスポートを確認して、新しいエクスポートを要求できます。
📋	コピー	上記と同様の構成の [編集] ページが開きますが、Name と Endpoint に _copy が付加されている点が異なります。[次へ] ボタンを使用しながら構成ページの手順を 1 つずつ実行し、必要な調整を実施します。[レビュー] ページで [構成] をクリックして新しい構成を保存します。
✎	編集	この構成の [編集] ページが開きます。[次へ] ボタンを使用しながら構成ページの手順を 1 つずつ実行し、必要な調整を実施します。[レビュー] ページで [構成] をクリックして変更を保存します。
⬇	ダウンロード	目的の構成の JSON 形式レコードを作成します。このファイルの名前は目的の構成と同じ名前になりますが、特殊文字はアンダースコアに置き換えられます。JSON 形式の構成をインポートするには、[新規構成] ボタンをクリックして [Import JSON] を選択し、参照コントロールを使用して目的のファイルを探します。
🗑	削除	テキスト・ボックスに表示されているとおりにエンドポイントの URL を入力し、それが間違いないことを確認して [削除] ボタンをクリックします。

注釈　　各構成で表示されるアクションはユーザのロールによって異なります。

20.3 バルク FHIR 構成の作成または編集

バルク FHIR 構成を作成または編集するには以下の手順に従います。

1. 管理者特権を持つユーザとして InterSystems IRIS for Health にログインします。
2. [ホーム] → [Health] → [foundationNamespace] → [バルク FHIR コーディネータ] に移動します。

3. [**バルク FHIR コーディネータ**] ページで、既存の構成に対して ✎ または ⧉ をクリックするか、[**新規作成**]、[**新規構成**] の順にクリックすることで、**作成/編集**ワークフローを呼び出します。

バルク FHIR 構成の**作成/編集**ワークフローは、以下の 5 つのページで構成されています。

1. 設定構成
2. 認証タイプ
3. Fetch
4. Storage Location
5. レビュー

作成/編集ワークフローの各ページで値を入力し、[**次へ**] をクリックして次のページへ移動します。最後のレビュー・ページで [**構成**] をクリックすると、バルク FHIR 構成が保存されます。以降の各セクションでは、さまざまな設定について説明します。

20.3.1 バルク FHIR の作成/編集ワークフロー：設定構成

バルク FHIR 構成を対象とした**作成/編集**ワークフローの [**構成設定**] ページには以下の各設定があります。

名前

使用する構成の一意な名前を入力します。主要なパラメータのいくつかを名前に使用すると、多数の構成からこの構成を区別しやすくなります。名前の指定は必須です。

Auto-start Exports

要求を受け取ると直ちにエクスポート・ジョブを開始するには、このオプションを選択します。エクスポートを開始するために手動での承認を必要とする場合は、このオプションの選択を解除します。既定では選択されています。

Authorized Users

この構成を使用したエクスポートの実行が許可されている、管理者ではないユーザの名前を、必要に応じてコンマ区切りで入力します。%HS_BFC_Exporter ロールのみを保有しているユーザがエクスポートを実行できるようにするには、その名前を承認済みユーザとしてここに記述する必要があります。ロールをマッピングする目的から、OAuth クライアントに関連するダミー・ユーザもこの対象になります。

注釈　%HS_BFC_Exporter ロールのみを保有しているユーザは、バルク FHIR コーディネータ・ホーム・ページの UI にアクセスできません。ただし、そのユーザにホーム・ページへの直接リンクを通知している場合、またはそのユーザの開始ネームスペースが BFC Foundation ネームスペースになるように構成されている場合は、この限りではありません。

BFC Endpoint

このバルク FHIR エンドポイントの URL を入力します。この値の入力は必須で、各種構成で一意とする必要があります。`/bulkFHIR/r4a` のように相対的な値とします。この構成を保存するときに、REST エンドポイントとして機能するこの URL を使用して Web アプリが作成されます。

Core FHIR Package

この構成を保存するときにその [Fetch Endpoint URL] から得られた読み取り専用フィールドです。その値は、例えば `hl7.fhir.r4.core@4.0.1` です。

Permitted Exports

許可するエクスポートのタイプを以下から選択します。

- [患者] ー すべての患者に関連する FHIR リソース群
- [グループ] ー 指定したグループ・リソースのすべてのメンバに関連する FHIR リソース群。`HS.BulkFHIR.Fetch.ODS.Adapter` 取得アダプタでは、グループ・リソースは Unified Care Record コホートとして把握されます。
- [システム] ー 患者との関連性の有無にかかわらず、すべての FHIR リソース。返されたリソースを `_type` パラメータで制限した用語データのエクスポートやサーバのバックアップなどのユース・ケースがサポートされています。

  注釈　`HS.BulkFHIR.Fetch.ODS.Adapter` 取得アダプタではシステムのエクスポートはできません。

Expire After

格納されている ndjson ファイルが失効するまでの期間 (分)。既定値は 1440 分 (1 日) です。失効した ndjson ファイルは、既定で 1 時間に 1 回実行される Bulk FHIR expiration task によってすべて削除されます。

Max file size

ndjson ファイルごとの最大サイズ (バイト)。エクスポートの際に、開いている ndjson ファイルのサイズがこの値に達すると、そのファイルは保存され、新しいファイルへのエクスポートが開始されます。既定値は 100 万バイトです。

Flush Interval

フラッシュ間隔 (分)。エクスポートでこの時間が経過すると、開いている ndjson ファイルは保存され、新しいファイルに対するエクスポートが開始されます。既定値は 60 分です。

Working Directory

ストレージ・ファイル・アダプタに渡す前の一時ファイルを格納するディレクトリ。

20.3.2 バルク FHIR の作成/編集ワークフロー : 承認の構成

作成/編集ワークフローの **[認証タイプ]** ページに表示される設定は、選択した Auth Adapter によって異なります。

Auth Adapter

このフィールドへの入力は必須です。ここでは、バルク・データ・クライアントとバルク FHIR コーディネータとの間で交わされる承認のタイプを定義します。[`HS.BulkFHIR.Auth.BasicAuth.Adapter`] または [`HS.BulkFHIR.Auth.OAuth.Adapter`] を選択します。InterSystems IRIS for Health には、OAuth 2.0 サーバがない場合にそれを自動的に構成するユーティリティが用意されています。

`BasicAuth` アダプタであれば、ほかに必要になる設定はありません。`OAuth` アダプタでは、以下の設定も入力する必要があります。

Issuer URL

既存の OAuth 2.0 サーバの URL。`OAuth` アダプタを使用している場合は、このフィールドへの入力が必須です。OAuth サーバは、IRIS for Health のインスタンス上などに置くことができます。IRIS for Health には、存在しない OAuth 2.0 サーバを作成するユーティリティが用意されています。

例えば、そのサーバが `https://example.org/oauth2` になります。

BFC Client Name

OAuth アダプタを使用している場合は、このフィールドへの入力が必須です。OAuth リソース・サーバとして機能するバルク FHIR コーディネータに対する OAuth クライアント構成の**アプリケーション名**を入力します。

バルク FHIR コーディネータ・エンドポイントに REST クライアントがトークンを提示すると、**BFC クライアント**では OAuth サーバを使用してそのトークンを検証します。

- **BFC クライアント**構成を定義していない場合、使用している OAuth サーバが動的クライアント登録をサポートしていれば、このバルク FHIR 構成を保存すると BFC クライアント構成が自動的に作成されます。
- OAuth サーバが動的クライアント登録をサポートしていない場合は、以下の手順を実行する必要があります。

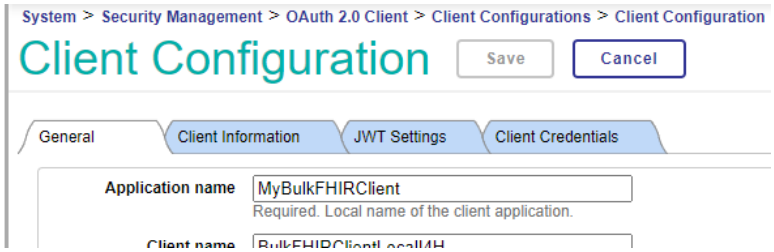
  1. OAuth サーバ上で OAuth サーバ管理者が BFC リソース・サーバに対してクライアントをプロビジョニングすることを要求します。
  2. **[ホーム]** → **[システム管理]** → **[セキュリティ]** → **[OAuth 2.0]** → **[クライアント]** → [issuerEndpoint] → **[クライアント構成]** へ移動することで、**[BFC Client Name]** の値を使用して、バルク FHIR コーディネータ・インスタンス上で OAuth クライアント構成を手動で追加します。

クライアント

この BFC エンドポイントからのバルク FHIR エクスポートの実行が承認されている OAuth クライアントのリスト。

[ホーム] → **[システム管理]** → **[セキュリティ]** → **[OAuth 2.0]** → **[クライアント]** → [issuerEndpoint] → **[クライアント構成]** で、OAuth クライアント構成と一致する OAuth クライアント記述を OAuth サーバに定義しておく必要があります。

OAuth クライアント構成ごとに**アプリケーション名**があります (**[一般]** タブに表示されています)。

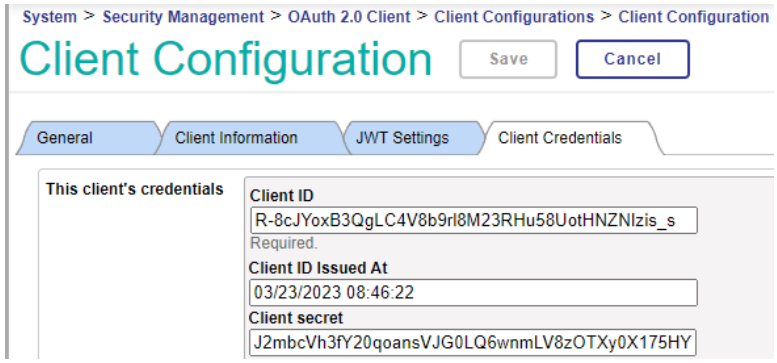


この BFC 構成を使用する OAuth クライアントを指定するには、name:authentication_method 形式のコンマ区切りリストを **[クライアント]** フィールドに入力します。

- *name* は、OAuth クライアント構成にある**アプリケーション名**です。
- authentication_method には、このクライアントが OAuth サーバへの認証に使用する Open ID Connect ワークフローを指定します。authentication_method の値は、`client_secret_post` または `private_key_jwt` とする必要があります。

注釈　入力したクライアント構成が存在しない場合、使用している OAuth サーバが動的クライアント登録をサポートしていれば、このバルク FHIR 構成を保存するとそのクライアント構成が自動的に作成されます。これらのクライアントを手動で作成することもできます。

各 OAuth クライアント構成には、**クライアント ID** と**クライアントの秘密鍵**も記述されています (**[クライアントの認証情報]** タブに表示されます)。



バルク・データ REST クライアントが BFC エンドポイントに要求を送信するとき、そのクライアントが提示したアクセス・トークンには client_id と client_secret が記述されます。アクセス・トークンの client_id は、BFC 構成の **[クライアント]** フィールドに示されている OAuth クライアント構成の**クライアント ID** との比較で検証されます。アクセス・トークンの client_secret は、OAuth クライアント構成の**クライアントの秘密鍵**との比較で検証されます。

重要　各 OAuth エクスポート・クライアントには、OAuth クライアント構成および名前が同じダミー InterSystems IRIS ユーザの両方が必要です。このダミー・ユーザは、適切なロールを OAuth クライアントに対応付けます。詳しい手順は、"ユーザの設定" を参照してください。

このダミー・ユーザは、ユーザのロールを OAuth クライアントに対応付ける手段としてのみ使用されます。これにより、REST エクスポート・クライアントが FHIR 相互作用でこの BFC エンドポイントに接続できます。多くの場合、このユーザは“有効ではない”ものとして作成されるので、このユーザの認証情報を使用して実際のユーザがログインすることはできません。

20.3.3 バルク FHIR の作成/編集ワークフロー：取得の構成

作成/編集ワークフローの [Fetch] ページでは以下の手順を実行できます。

1. 取得アダプタの選択
2. 取得アダプタの構成
3. 承認設定の構成

20.3.3.1 取得の構成：アダプタの選択

Fetch Adapter

このフィールドへの入力は必須です。[`HS.BulkFHIR.Fetch.PureFHIR.Adapter`] または [`HS.BulkFHIR.Fetch.ODS.Adapter`] を選択します。ODS アダプタは Unified Care Record ODS に固有です。

注釈　`HS.BulkFHIR.Fetch.ODS.Adapter` ではシステムをエクスポートできません。

20.3.3.2 取得の構成：アダプタの構成

[Adapter Configuration] 見出しに示されている設定は、ODS 取得アダプタに対してのみ表示される **Registry Webservice** 設定を除き、すべてが PureFHIR 取得アダプタと ODS 取得アダプタの両方に対して表示されます。

エンドポイント URL

FHIR エンドポイントの完全な URL。例えば、`https://example.org/fhir/r4`。このフィールドへの入力は必須です。

SSL 構成

HTTPS を使用して FHIR エンドポイントと通信する方法を記述した、InterSystems IRIS の SSL/TLS クライアント構成。

リソース・タイプ

この構成のエクスポート・オペレーションで扱う必要がある FHIR リソース・タイプをコンマ区切りで記述した既定のリスト。バルク・データ要求に `_type` クエリ・パラメータを使用することで、このリストをクライアントによってオーバーライドできます。このフィールドを空のままにすると、既定ですべてのリソース・タイプが扱われます。

Max Requests Per Second

FHIR エンドポイントに対して毎秒実行できる HTTP(S) 要求の最大数。この値は、構成のすべてのアクティブなエクスポート・オペレーションに共通で適用されるので、バルク FHIR コーディネータによって FHIR エンドポイントに発生する負荷を制限するために使用できます。秒あたり 10 件の要求が既定値です。

HTTP Timeout

データの取得で FHIR エンドポイントに対して実行する HTTP(S) 要求ごとのタイムアウト値 (秒)。エクスポートするデータが大量で、FHIR エンドポイントのページ・サイズが大きい場合、取得でタイムアウト・エラーが発生するのであれば、このタイムアウトを長くします。既定値は 180 秒です。

ワーカ・ジョブ

取得処理の実行が割り当てられているバックグラウンド・ワーカ・ジョブの数。4 件のジョブが既定値です。

Registry Webservice Credential Id

`HS.BulkFHIR.Fetch.ODS.Adapter` を使用するときは必ず指定します。Unified Care Record レジストリで Hub Web サービスを呼び出すときに使用する相互運用認証情報。

この認証情報は、Hub Web サービスの UCR サービス・レジストリ・エントリの Username Token Profile 設定と一致している必要があります。このサービス・レジストリ・エントリは baseURL`/services/HS.Hub.HSWS.WebServices.cls` を参照するので、これによって特定できます。多くの場合、このサービス・レジストリ・エントリの名前は HSREGISTRY です。HSREGISTRY サービス・レジストリ・エントリの Username Token Profile 設定の値は、一般的に `HS_Services` です。

実行時に Hub Web サービスを呼び出すときは、この認証情報のユーザ名とパスワードが使用されます。

Registry Webservice Endpoint URL

`HS.BulkFHIR.Fetch.ODS.Adapter` を使用するときは必ず指定します。Unified Care Record レジストリの Hub Web サービス の URL。通常は、以下のとおりです。

```
https://UCRHost:Port/csp/healthshare/registryNamespace/services/HS.Hub.HSWS.WebServices.cls
```

20.3.3.3 取得の構成 : 承認の構成

取得アダプタを構成した後、FHIR エンドポイントとの BFC 取得相互作用向けに承認設定を指定します。

Authorization Type

エクスポートの際にバルク FHIR コーディネータと FHIR エンドポイントとの間で使用する承認のタイプを定義します。基本的な認証では [`HTTP`]、X-API-Key ヘッダ認証では [`X-API`]、OAuth 2.0 では [`OAuth`] をそれぞれ選択します。

注釈
- 基本的な認証を選択した場合にのみ、[HTTP Credential Id] 設定が表示されます。
- X-API-Key 認証を選択した場合にのみ、[X-API Key Credential] 設定が表示されます。
- 他の設定は、OAuth 2.0 を選択した場合にのみ表示されます。

HTTP Credential Id

FHIR エンドポイントと通信する際に、基本的な認証でのみ使用する相互運用認証情報。

X-API Key Credential

FHIR エンドポイントと通信する際に、X-API-Key 認証でのみ使用する相互運用認証情報。BFC から HTTP 要求を FHIR エンドポイントに送信する際に、この認証情報のパスワードが X-API-Key ヘッダに記述されます。

OAuth Issuer URL

FHIR エンドポイントの OAuth サーバの発行者 URL。

この OAuth サーバが検出をサポートしている場合は、この BFC 構成を保存するときに、そのサーバ記述が作成されます。

クライアント名

エクスポートの際に、FHIR エンドポイントの OAuth サーバとの認証でバルク FHIR コーディネータが使用する OAuth クライアント構成の**アプリケーション名**。

FHIR エンドポイントの OAuth サーバで検出と動的クライアント登録がサポートされていれば、この BFC 構成を保存するときに、このクライアント構成が自動的に作成されます。また、**[ホーム]** → **[システム管理]** → **[セキュリティ]** → **[OAuth 2.0]** → **[クライアント]** → [FHIRServerIssuerEndpoint] → **[クライアント構成]** で、このクライアント構成を手動で作成することもできます。

Grant Type

FHIR エンドポイントの OAuth サーバからアクセス・トークンを取得するときに使用する OAuth 付与タイプ。

クライアント構成の **[必要な付与タイプ]** に応じて、このフィールドで使用できる値は以下のようになります。

- [`password`] ー リソース所有者のパスワード認証情報
- [`client_credentials`] ー クライアント認証情報

Fetch Token Scopes

FHIR エンドポイントの OAuth サーバからアクセス・トークンを取得するときに指定する OAuth スコープのコンマ区切りリスト。バルク FHIR コーディネータに対する元の要求でアクセス・トークンを使用していなかった場合にのみ適用します。例えば、`system/*.read` を指定するとすべてが許可されます。

重要 [Authorization Type] が [`OAuth`] である場合、患者またはグループのエクスポートでは最小限の `system/Patient.read` が必要です。これは、Patient コンパートメントと Patient コンパートメント外部の関連リソース (Practitioner など) の両方を返す取得で使用する Patient/$everything オペレーションをサポートするためです。このオペレーションでは、`_type` パラメータで Patient リソースをフィルタ処理する場合でも、取得する他のすべてのリソースに対応するスコープと共に `system/Patient.read` スコープが必要です。

Fetch Token Credential ID

付与タイプで基本的な認証の認証情報を必要とする場合に、FHIR エンドポイントの OAuth サーバでの認証に使用する相互運用認証情報。

20.3.4 バルク FHIR の作成/編集ワークフロー : ストレージの構成

バルク FHIR 構成を対象とした**作成/編集**ワークフローの [Storage Location] ページには以下の各設定があります。

Storage Adapter

このフィールドへの入力は必須です。[`HS.BulkFHIR.Storage.File.Adapter`] を選択します。

File URL

バルク・エクスポート・ファイルを提供する CSP アプリケーションの URL。異なる構成であっても同じネームスペースにあれば、使用する URL は同じです。例えば、`/file` や `/bulkfhir/file` です。構成を保存するときに CSP アプリケーションが作成されます。

ディレクトリ

エクスポートした FHIR リソースを収めた ndjson ファイルの格納先。指定しない場合は、既定で installDir**`/mgr/Temp/BulkFHIR/`**namespace**`/`** になります。このディレクトリには、セッションごとに番号付きの名前を持つサブディレクトリがあります。各セッション・サブディレクトリには、リソース・グループのディレクトリとファイルがあります。ネームスペースごとに独立したディレクトリを使用する必要があります。セッション識別子が衝突する可能性があるからです。

20.3.5 バルク FHIR の作成/編集ワークフロー : 構成のレビューと検証

バルク FHIR 構成を対象とした**作成/編集**ワークフローの **[レビュー]** ページには、他のページにある各設定の値が表示されます。これらの設定をレビューし、**[構成]** をクリックしてバルク FHIR 構成を保存します。**[構成]** をクリックすると BFC によって各設定が検証され、対処を必要とする問題があるとそれが表示されます。各設定が検証に適合すると、必要な OAuth クライアント構成とサーバ記述が BFC によってすべて自動的に構成されます。

20.4 JSON を使用したバルク FHIR 構成のインポート

構成をインポートするには、**[新規構成]** ボタンをクリックして [Import JSON] を選択し、参照コントロールを使用して目的のファイルを探します。

以下では、バルク FHIR 構成向け JSON 仕様をセクション別で具体的に示しています。

- 空白のままとする単純なプロパティは除外できます。
- JSON で使用されている角括弧は、値のコンマ区切りリストを入力できることを示しています。
- JSON フラグメントのテキストの色は以下を示しています。
  - 緑色のテキストは、引用符の中の文字列が想定されることを示します。
  - 赤色のテキストは、数値が想定されることを示します。
  - オレンジ色のテキストは、ブーリアン値である `true` または `false` が想定されることを示します。

次の各ページの JSON 例をその後に示します。

- [構成設定] ページ
- [認証タイプ] ページ
- [Fetch] ページ
- [Storage Location] ページ

20.4.1 [構成設定] ページの JSON 例

```
{
  "name": "MyBFCConfig",
  "auto_start": true,
  "authorized_users": [
    "MyBulkFHIRExportOAuthClient",
    "MyOtherClient"
  ],
  "endpoint_url": "/BFC-REST/a1",
  "core_fhir_package": "hl7.fhir.r4.core@4.0.1",
  "patient_export": true,
  "group_export": false,
  "system_export": true,
  "expire_after_mins": 1440,
  "max_file_size": 1000000,
  "flush_interval_mins": 60,
  "working_directory": "/myLocal/temp/workspace/",
```

20.4.2 [認証タイプ] ページの JSON 例

- 基本的な承認アダプタの JSON 例
- OAuth アダプタの JSON 例

20.4.2.1 基本的な承認アダプタ

```
  "auth_adapter": "HS.BulkFHIR.Auth.BasicAuth.Adapter",
  "auth_config": {},
```

20.4.2.2 OAuth アダプタ

```
  "auth_adapter": "HS.BulkFHIR.Auth.OAuth.Adapter",
  "auth_config": {
    "issuer_url": "https://MyOAuthServer:44473/oauth2",
    "bfc_client_name": "MyBFCResourceServerOAuthClient",
    "clients": [
      {
        "name": "MyBulkFHIRExportOAuthClient",
        "authentication_method": "client_secret_post"
      },
      {
        "name": "MyOtherClient",
        "authentication_method": "private_key_jwt"
      }
    ]
  },
```

20.4.3 [Fetch] ページの JSON 例

- ODS 取得アダプタの JSON 例
- 純粋な FHIR 取得アダプタの JSON 例
- 取得承認設定

20.4.3.1 ODS 取得アダプタ

```
  "fetch_adapter": "HS.BulkFHIR.Fetch.ODS.Adapter",
  "fetch_config": {
    "endpoint_url": "https://MyFHIRServer:44473/fhir/r4",
    "ssl_configuration": "MyFHIRServer_SSL",
    "resource_types": [
      "Patient",
      "Practitioner"
    ],
    "max_req_per_sec": 5,
    "http_timeout": 180,
    "worker_jobs": 4,
    "registry_webservice_credential_id": "HS_Services",
    "registry_webservice_endpoint_url": "https://MyODS:44473/csp/healthshare/hsregistry/services/HS.Hub.HSWS.WebServices.cls",
```

"fetch_config": { プロパティの終了部分は、"取得承認設定" を参照してください。

20.4.3.2 純粋な FHIR 取得アダプタ

```
32  "fetch_adapter": "HS.BulkFHIR.Fetch.PureFHIR.Adapter",
33  "fetch_config": {
34    "endpoint_url": "https://MyFHIRServer:44473/fhir/r4",
35    "ssl_configuration": "MyFHIRServer_SSL",
36    "resource_types": [
37      "Patient",
38      "Practitioner"
39    ],
40    "max_req_per_sec": 5,
41    "http_timeout": 180,
42    "worker_jobs": 4,
```

"fetch_config": { プロパティの終了部分は、"取得承認設定" を参照してください。

20.4.3.3 取得承認設定

- HTTP 取得承認の JSON 例
- X-API 取得承認の JSON 例
- OAuth 取得承認の JSON 例

HTTP 取得承認

以下の JSON フラグメントは "fetch_config": { プロパティの終了部分です。

```
43      "http_credential_id": "MyFHIRServerBasic"
44    },
```

X-API 取得承認

以下の JSON フラグメントは "fetch_config": { プロパティの終了部分です。

```
43      "x_api_key_credential_id": "MyFHIRServerXAPI"
44    },
```

OAuth 取得承認

以下の JSON フラグメントは "fetch_config": { プロパティの終了部分です。

```
43    "oauth_issuer_url": "https://MyFHIREndpointOAuthServer:44473/oauth2",
44    "client_name": "BulkFHIRClientLocalI4H",
45    "grant_type": "client_credentials",
46    "fetch_token_scopes": [
47      "user/*.read"
48    ],
49    "fetch_token_credential_id": "MyFHIRServerBasic"
50  },
```

20.4.4 [Storage Location] ページの JSON 例

```
51  "storage_adapter": "HS.BulkFHIR.Storage.File.Adapter"
52  "storage_config": {
53    "file_url": "/file",
54    "directory": "/myLocal/file/workspace/"
55  }
56 }
```

20.5 バルク FHIR ホーム・ページからのエクスポート実行

クライアントに代わり、バルク FHIR コーディネータが FHIR エンドポイントにある FHIR リソース群を要求した場合、このアクションはエクスポートと呼ばれます。

ここでは以下の方法について説明します。

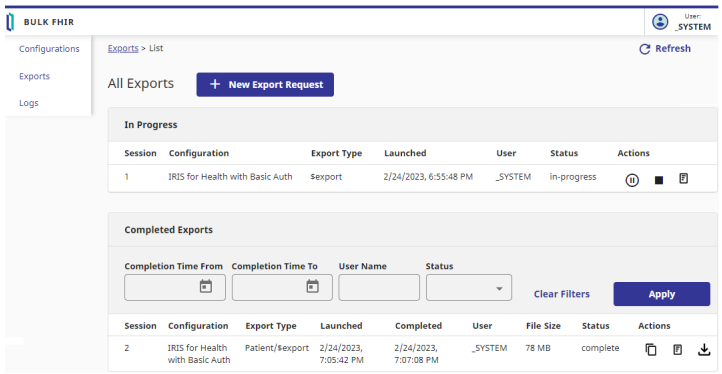
- [エクスポート] ページへのアクセス
- エクスポートの開始とそのステータスの確認

- 完了したエクスポートの ndjson のダウンロード
- エクスポート・ログの表示

20.5.1 [エクスポート] ページへのアクセス

バルク FHIR エクスポートを開始または検査するには以下の手順に従います。

1. 適切な特権を持つユーザとして InterSystems IRIS for Health にログインします。
2. **[ホーム]** → **[Health]** → [foundationNamespace] → **[バルク FHIR コーディネータ]** に移動します。
3. 以下のいずれかの方法で **[エクスポート]** ページに移動します。
   - **[エクスポート]** リンクをクリックし、**[エクスポート]** → **[リスト]** ページを開いてすべてのエクスポートを表示します。
   - をクリックし、**[エクスポート]** → **[表示]** ページを開いて特定の構成のエクスポートを表示します。



[エクスポート] ページでは、進行中のエクスポートと完了したエクスポートのステータスを確認できます。以下の各アクションも実行できます。

アイコン	アクション	注
	新規エクスポート要求	新規エクスポートを開始します。
⏸	一時停止	進行中のエクスポートを一時停止します。
▶	再開	一時停止していたエクスポートを再開します。
■	キャンセル	進行中のエクスポートをキャンセルします。
🗎	ログの表示	進行中のエクスポートまたは完了したエクスポートのログを表示します。[ログ] → [リスト] ページを開き、エクスポート・ログをフィルタ処理して表示できます。
🗐	コピー	完了したエクスポートの情報を使用して新しいエクスポートを作成します。
⬇	ダウンロード	[エクスポート] → [Exported Files] ページを開きます。タイプ別に特定リソースの ndjson ファイルを検索してダウンロードできるほか、エクスポート・エラーもダウンロードできます。

注釈　表示されるアクションはユーザのロールによって異なります。

20.5.2 エクスポート要求の開始

1. [エクスポート] ページで [New Export Request] をクリックしてエクスポートを開始します。
2. [構成] ドロップダウン・リストから BFC 構成を選択します。
3. [次へ] をクリックします。
4. [エクスポート] からエクスポートのタイプを選択します。[システム]、[グループ]、または [患者] を選択できます。
5. グループのエクスポートを選択した場合は、[Group ID] にグループ ID を入力します。Unified Care Report ODS では、コホート名がグループ ID になります。
6. 必要に応じ、[Since] フィールドに YYYY-MM-DD の書式で日付を入力するか、日付選択機能を使用して日付を選択します。[Add Time] をクリックして時刻を入力することもできます。
7. [Export Now] をクリックするとエクスポートが始まります。
8. [エクスポート] ページの [処理中] テーブルにエクスポートが行として表示されます。進行中のエクスポートを一時停止、再開、またはキャンセルできます。ログを表示して、エクスポートの進行状況を確認することもできます。

20.5.3 完了したエクスポートの ndjson のダウンロード

1. [エクスポート] ページの [Completed Exports] ペインで必要に応じてフィルタ値を入力し、[適用] をクリックすると完了済みエクスポートのリストがフィルタ処理されます。
2. [Completed Exports] テーブルの目的の行で ⬇ をクリックすると、そのエクスポートに存在するファイルの一覧が表示されます。エクスポートするファイルがリソース・タイプ別に分類されています。
3. 一覧に表示されるファイルを少なくするには、必要に応じて検索語を入力します。
4. ndjson ファイルをダウンロードするには ⬇ をクリックします。

20.5.4 エクスポート・ログの表示

バルク FHIR ワークフローの以下のコンポーネントで発生した各種イベントについて、エクスポート・ログに詳しい情報が記録されます。

コンポーネント	イベント・タイプ	詳細
BFC	session_action	アクション：作成、開始、一時停止、再開、完了、失敗（理由とスタック）
BFC	flush	理由 ： サイズ、間隔、finalize_session
取得	rest_request	パス、rate_limit_time、http_response_time、http_status（理由、スタック）
ストレージ	flush	理由 ： サイズ、間隔、finalize_session
ストレージ	file_access	クライアント、ファイル

セッションのエクスポート・ログを表示するには以下の手順に従います。

1. [エクスポート] ページで、[処理中] または [完了] が示されたセッションの行で 🗎 をクリックします。または、[ログ] をクリックしてすべてのセッションのログを表示します。
2. 必要に応じ、一覧に表示されるログを少なくするには、フィルタ値を入力して [適用] をクリックします。
3. 🗎 をクリックすると特定のログ・ファイルが表示されます。

20.6 REST クライアントからのバルク FHIR エクスポート

クライアントに代わり、バルク FHIR コーディネータが FHIR エンドポイントにある FHIR リソース群を要求した場合、このアクションはエクスポートと呼ばれます。

ここでは以下の方法について説明します。

- REST エクスポートの開始
- 進行中のエクスポートのステータス確認
- 完了したエクスポートの ndjson のダウンロード

20.6.1 REST クライアントからの要求のエクスポート開始

REST クライアントからのバルク FHIR エクスポートを開始するには、目的のオペレーションを指定して BFC エンドポイントに GET 要求を送信します。以下に例を挙げます。

- システム ― `GET https://`bfcEndpoint`/$export`
- 患者 ― `GET https://`bfcEndpoint`/Patient/$export`
- グループ ― `GET https://`bfcEndpoint`/Group/`groupID`/$export`

この BFC 構成で OAuth の **Auth アダプタ**を使用している場合は、以下を指定してアクセス・トークンを取得します。

- OAuth サーバのアクセス・トークン・エンドポイント：

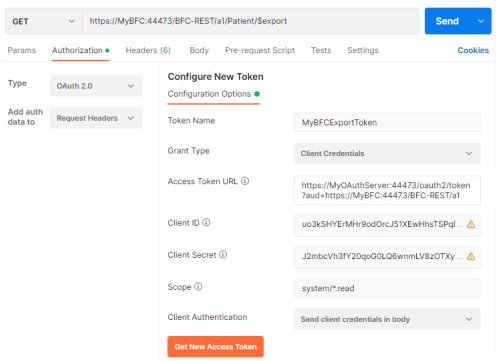
  issuerEndpoint`/token`

  必要に応じて対象ユーザ：

  `?aud=https://`*bfcEndpoint*

- BFC 構成の **[認証タイプ]** タブに挙げられているいずれかの OAuth **クライアント**のクライアント ID とクライアント秘密鍵。
- OAuth クライアントでサポートされている付与タイプ (InterSystems IRIS では、OAuth クライアント構成の **[一般]** タブで選択した**必須付与タイプ**のいずれかです)。
- スコープ。最小限必要なスコープは `system/Patient.read` です。`system/*.read` を指定するとすべてが許可されます。

OAuth を使用して患者をエクスポートする例を以下に示します。



$export では、以下のオプション・パラメータを使用できます。

_outputFormat

BFC では、改行区切りで JSON (ndjson) ファイルがエクスポートされます。

application/fhir+ndjson 値と、その省略名である application/ndjson と ndjson の値を使用できます。

_since

以下の“FHIR 瞬間時点”形式で指定した時間の経過後にリソースの状態が変化すると、そのリソースが応答に追加されます。

YYYY-MM-DDThh:mm:ss.sss+zz:zz

_type

エクスポートの対象とする FHIR リソース・タイプのコンマ区切りリスト。既定は、構成した取得アダプタでサポートされているすべてのリソース・タイプです。

20.6.2 REST クライアントからのエクスポートのステータス確認

最初の GET 要求に対する応答ヘッダには、以下の形式で URL を指定した CONTENT-LOCATION キーがあります。

bfcEndpoint/status/*sessionNumber*

CONTENT-LOCATION の URL へ定期的に GET 要求を送信して、バルク FHIR エクスポート・セッションのステータスを取得します。

以下のようなステータス応答が返されます。

202 Accepted

- BFC でエクスポートが処理されています。
- 応答ヘッダには、値を in-progress とした X-PROGRESS キーが追加されます。

200 OK

- エクスポートが完了したので、ファイルをダウンロードできます。
- 応答ヘッダには、BFC ファイル・サーバに ndjson ファイルが保持されている期間を示した EXPIRES キーが追加されます。
- 応答本文には、BFC ファイル・サーバ上に格納されている ndjson ファイルの URL が記述されます。返されるリソース・タイプごとに 1 つ以上のファイルがあります。


```
transactionTime : 2023-03-23T15:04:37Z
request : https://ub20fast1-91:44473/bulkFHIR/r4a/Patient/$export
output
    type : AllergyIntolerance
    url : https://ub20fast1-91:44473/file/2_AllergyIntolerance_0001.ndjson

    type : CarePlan
    url : https://ub20fast1-91:44473/file/2_CarePlan_0001.ndjson

    type : CareTeam
    url : https://ub20fast1-91:44473/file/2_CareTeam_0001.ndjson

    type : Claim
    url : https://ub20fast1-91:44473/file/2_Claim_0001.ndjson

    type : Claim
    url : https://ub20fast1-91:44473/file/2_Claim_0002.ndjson

    type : Claim
    url : https://ub20fast1-91:44473/file/2_Claim_0003.ndjson
```

500 Internal Server Error

- BFC でエラーが発生しました。

20.6.3 完了したエクスポートの ndjson のダウンロード

受信した応答ヘッダに `Status: 200 OK` があれば、BFC ファイル・サーバから目的のファイルをダウンロードできます。これらのファイルを取得するには、ファイルの URL ごとに GET 要求を送信します。

この BFC 構成で OAuth の **Auth アダプタ**を使用している場合は、以下を指定して新しいアクセス・トークンを取得します。

- BFC 構成の **[取得]** タブで特定した**付与タイプ**。
- OAuth サーバのアクセス・トークン・エンドポイント：

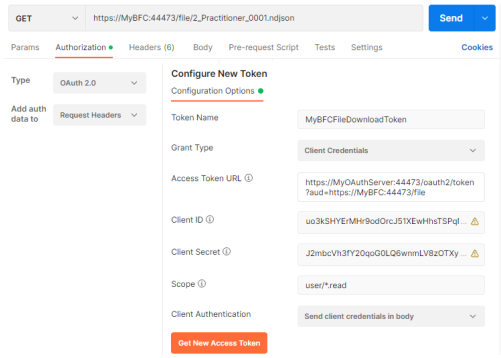
  issuerEndpoint`/token`

  必要に応じて対象ユーザ：

  `?aud=https://`*bfcFileEndpoint*

- BFC 構成の **[認証タイプ]** タブに挙げられているいずれかの OAuth **クライアント**のクライアント ID とクライアント秘密鍵。
- スコープ。ファイルのダウンロードでは一般的に `user/*.read` です。

以下に例を示します。



20.7 バルク FHIR ロール

バルク FHIR コーディネータは、以下のようにロールベースのアクセス制御を提供します。

ロール	許可	注
%HS_BFC_Exporter	ユーザが承認済みユーザとして指定されている構成上で患者またはグループのエクスポートを表示し、また実行します。 これらのエクスポートを一時停止、再開、またはキャンセルします。また、これらのエクスポートのログを表示し、ダウンロードします。	OAuth クライアント構成と照合するために作成するダミーの InterSystems IRIS ユーザそれぞれに、このロールを割り当てます。 %HS_BFC_Exporter は、OAuth クライアントにロールを対応付けるダミー・ユーザに使用することを主眼としています。したがって、このロールのみのユーザは、管理ポータルのメニューからバルク FHIR コーディネータのホーム・ページにアクセスできません。実際のユーザにこのロールを割り当てる場合は、BFC Foundation ネームスペースをそのユーザの開始ネームスペースとするか、そのユーザにポータルで BFC ホーム・ページへの直接リンクを通知します。 このロールと %HS_BFC_Export_Manage ロールを割り当てられたユーザは、すべての構成上で患者またはグループのエクスポートを開始でき、自身が開始したエクスポートのログを表示し、ダウンロードできます。また、ポータルのホーム・ページにもアクセスできます。
%HS_BFC_Export_Manage	すべての構成とすべてのエクスポートを表示します。 構成の作成、エクスポートの開始、エクスポート・ログの表示はできません。	%HS_BFC_Exporter と共に割り当てることで特権を拡張できます（上記を参照）。
%HS_BFC_Administrator	すべての構成を表示、作成、編集、コピー、削除します。 システムのエクスポートをサポートしているあらゆる構成上で、そのエクスポートを実行します。 すべてのエクスポートを表示、一時停止、停止、再開、キャンセルし、そのログを表示またはダウンロードします。 自身で開始したエクスポートをダウンロードします。	%HS_BFC_Download_Manage と共に割り当てることで、すべてのエクスポートのファイルをダウンロードできます。
%HS_BFC_Download_Manage	すべてのエクスポートのファイルをダウンロードします。	%HS_BFC_Administrator の特権を拡張するために使用します（上記を参照）。

[ホーム] → [セキュリティ] → [ロール] へ移動すると、上記の各ロールに関連付けられたリソースと特権を確認できます。[ホーム] → [セキュリティ] → [リソース] へ移動すると、カスタム・ロールの作成で使用できる %HS_BFC リソースをすべて確認できます。

20.8 バルク FHIR コーディネータ向け OAuth 2.0 サーバの作成

バルク FHIR REST クライアントとバルク FHIR コーディネータ間の認証で OAuth 2.0 を使用するものの、OAuth 2.0 サーバがない場合は、InterSystems IRIS for Health に用意されているユーティリティを使用できます。このユーティリティは、バルク FHIR コーディネータのエンドポイントで SMART バックエンド・サービス承認をサポートすることを主な目的として、ローカル・インスタンス上に OAuth 2.0 サーバを作成します。この OAuth サーバは、動的クライアント登録をサポートするように構成されます。

このユーティリティを適切に使用するには、以下の前提条件を満足する必要があります。

1. SSL/TLS に対応するように Web サーバを構成している。
2. 使用しているインスタンスに SSL/TLS 構成を作成している。
3. インストーラ・ウィザードの **[安全な通信の構成]** ダイアログで、安全な通信構成を作成し、有効にしている。
4. インストーラ・ウィザードで安全な通信を構成した後、バルク FHIR 構成を作成する Foundation ネームスペースを構成し、有効にしている。

この OAuth 2.0 サーバ・ユーティリティは、**HS.BulkFHIR.OAuth2Installer** クラスの 2 つのメソッドから構成されています。Foundation ネームスペースから以下のメソッドを呼び出します。

SetupOAuthServer()

バルク FHIR のローカル IRIS インスタンスに IRIS OAuth 2.0 承認サーバを構成し、さらにその OAuth サーバの発行者エンドポイントを指すサービス・レジストリ・エントリを作成します。このメソッドでは、これら 2 つのアイテムの値がクラス・パラメータの `OAuthSSLConfigName` と `OAuthIssuerServiceName` に依存しています。

引数：

- `pForceDelete`

  0 = 既存の OAuth サーバが見つかった場合は実行を中止し、失敗であることを返します (既定)。

  1 = 既存の OAuth サーバとそのクライアントを削除したうえで再作成します。

- `pVerbose`

  0 = メソッドの実行結果テキストを表示しません。

  1 = メソッドの実行結果テキストを表示します (既定)。

SetupServiceEntry()

現在の IRIS インスタンスでの OAuth サーバの発行者エンドポイントを指すサービス・レジストリ・エントリを、現在のネームスペースに作成します。このメソッドでは、これら 2 つのアイテムの値がクラス・パラメータの `OAuthSSLConfigName` と `OAuthIssuerServiceName` に依存しています。

OAuth サーバを既に目的どおりに設定済みで、別の Foundation ネームスペースにバルク FHIR 構成を作成する場合にのみ、このメソッドが必要です。

引数：

- `pVerbose`:

  0 = メソッドの実行結果テキストを表示しません。

  1 = メソッドの実行結果テキストを表示します (既定)。

注釈　バルク FHIR 構成を作成して保存するときに、OAuth 2.0 クライアント構成が自動的に設定されます。

20.9 バルク FHIR 設定チェックリスト

バルクFHIR 相互作用の構成では、さまざまな場所で多数の移動操作が必要です。必要なすべての構成が発生していることを確認するためのチェックリストを以下に示します。この確認により、バルク FHIR 相互作用の正常な動作を図ります。

- FHIR リソース・サーバ設定のチェックリスト
- バルク FHIR コーディネータ設定のチェックリスト
- REST クライアント設定のチェックリスト

20.9.1 FHIR リソース・サーバ設定のチェックリスト

- FHIR リソース・サーバごとにエンドポイントの URL を取得します。
- SSL/TLS 構成情報を取得します。
- OAuth 取得アダプタを使用している場合は、FHIR エンドポイントの OAuth サーバ・エンドポイントの URL を取得します。受け入れられる付与タイプを確認します。
- ODS 取得アダプタを使用している場合は、Unified Care Record レジストリの Web サービス・エンドポイントと SSL/TLS 構成情報を取得します。
- 指定の検索で返すことができるリソースの数が FHIR エンドポイントによって制限されている場合は、検索エラーを防止するために、この制限値を大きくすることを検討します。InterSystems IRIS for Health の場合、FHIR サーバを作成すると、[検索結果の最大数] 設定が既定で 1000 に設定されます。この値を大きくするには、**[ホーム]** → [Health] → **[FHIR 構成]** → **[サーバ構成]** → [endpoint] → **[構成]** → **[検索結果の最大数]** へ移動します。推奨値は FHIR サーバのコンテンツによって異なりますが、3000 であれば十分です。

20.9.2 バルク FHIR コーディネータ設定のチェックリスト

BFC 構成を作成する前に、以下の前提条件が成立していることを確認します。

- SSL/TLS 構成の作成
- 相互運用認証情報の作成
- OAuth の設定
- ユーザの設定
- ストレージの場所の設定

20.9.2.1 SSL/TLS 構成の作成

- 各 FHIR リソース・サーバおよび各 OAuth サーバとの通信で使用する SSL/TLS 構成を作成します。
- ODS 取得アダプタを使用している場合は、Unified Care Record Web サーバとの通信で使用する SSL/TLS 構成を作成します。

20.9.2.2 相互運用認証情報の作成

- HTTP 取得アダプタを使用している場合は、FHIR エンドポイントへの認証のための認証情報を作成します。

- X-API Key 取得アダプタを使用している場合は、FHIR エンドポイントへの認証のための認証情報を作成します。この場合は、API キーがこの認証情報のパスワードになります。
- OAuth 取得アダプタを使用していて、FHIR エンドポイントの付与タイプで基本的な認証の認証情報が必要な場合は、取得トークンの認証情報を作成します。

20.9.2.3 OAuth の設定

OAuth 2.0 を BFC 承認アダプタとして使用している場合、または FHIR エンドポイントで取得に OAuth 2.0 を必要とする場合は、OAuth を適切に設定する必要があります。この設定では、BFC 向け OAuth サーバ、OAuth を必要とする FHIR エンドポイントのサーバ記述、さまざまなクライアント構成を作成することがあります。

OAuth サーバの作成または特定

OAuth を BFC **承認アダプタ**として使用する場合は、SMART バックエンド・サービス承認をサポートするバルク FHIR コーディネータで使用する OAuth サーバの URL を指定する必要があります。OAuth サーバがない場合は、InterSystems IRIS for Health のユーティリティを使用して OAuth サーバを作成できます。

OAuth リソース・サーバとした BFC の OAuth クライアントの作成

OAuth を BFC の**承認アダプタ**として使用する場合は、その BFC を OAuth サーバ発行者エンドポイントに対する OAuth リソース・サーバとして、その BFC の OAuth クライアント構成が必要です。その**アプリケーション名**を記録しておきます。

OAuth サーバで動的クライアント登録がサポートされていれば、BFC 構成を保存するときに、OAuth クライアント構成が自動的に作成されます。

エクスポートで使用する OAuth クライアントの作成

OAuth を BFC の**承認アダプタ**として使用する場合は、OAuth サーバ発行者エンドポイントに対する OAuth クライアント構成が、バルク FHIR REST クライアントで使用するために必要です。各クライアントの**アプリケーション名**と**クライアント ID** を記録しておきます。

このような OAuth クライアント構成が [**クライアント**] フィールドに挙げられていて、OAuth サーバで動的クライアント登録がサポートされていれば、BFC 構成を保存するときに、OAuth クライアント構成が自動的に作成されます。また、これらの構成を手動で作成することもできます。

FHIR エンドポイント向けのサーバ記述と OAuth クライアントの作成

承認タイプが OAuth である FHIR エンドポイントごとに、FHIR エンドポイントの OAuth サーバに対する検出を使用して、BFC インスタンスにサーバ記述を作成します。動的クライアント登録を使用するか、手動でクライアント ID とクライアントの秘密鍵を入力することで、各 FHIR エンドポイントの OAuth サーバ発行者エンドポイントに対し、BFC の OAuth クライアント構成を作成します。

FHIR エンドポイントの OAuth サーバで発見と動的クライアント登録がサポートされていれば、BFC 構成を保存するときに、そのFHIR エンドポイントの OAuth サーバ向けにサーバ記述と BFC クライアント構成の両方が自動的に作成されます。

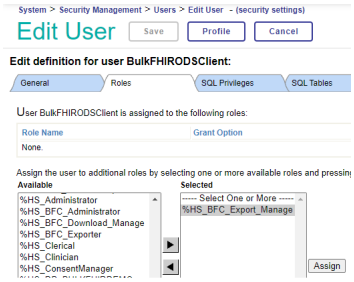
20.9.2.4 ユーザの設定

- %HS_BFC_Administrator ロールを持つ管理ユーザを作成します。
- OAuth エクスポート・クライアントごとにダミー・ユーザを作成します。このダミー・ユーザには %HS_BFC_Exporter 以上のロールが必要です。

  各 OAuth エクスポート・クライアントには、OAuth クライアント構成および名前が同じダミー InterSystems IRIS ユーザの両方が必要です。このダミー・ユーザは、適切なロールを OAuth クライアントに対応付けます。

  OAuth クライアントのダミー・ユーザを作成するには以下の手順に従います。

1. バルク FHIR コーディネータのインスタンスで、[**ホーム**] → [**システム管理**] → [**セキュリティ**] → [**ユーザ**] → [**新規ユーザ作成**] へ移動します。
2. 認証アダプタを構成したときに [**クライアント**] に入力したname文字列と同じ名前を [**名前**] フィールドに入力します。これは、OAuth クライアント構成で指定した**アプリケーション名**です。
3. [**パスワード**] フィールドと [**パスワード (再入力)**] フィールドの両方に、まったく同じランダムな文字列を入力します。このアカウントをログインには使用しない場合でも、InterSystems IRIS のユーザを作成するためにパスワードが必要です。
4. このユーザ・アカウントはログインに使用しないので [**ユーザ有効**] の選択を解除します。これにより、誰かがユーザとしてログインすることを防止できます。
5. [**保存**] をクリックします。
6. [**ロール**] タブで適切なユーザ・ロールを追加します。一般的には %HS_BFC_Exporter を追加します。ロールを追加するには、そのロールを [**利用可能**] ペインで選択し、▸ をクリックして [**選択済み**] ペインへ移動します。続いて、以下のように [**割り当てる**] をクリックします。



このダミー・ユーザは、ユーザのロールを OAuth クライアントに対応付ける手段としてのみ使用されます。これにより、REST エクスポート・クライアントが FHIR 相互作用でこの BFC エンドポイントに接続できます。

20.9.2.5 ストレージの場所の設定

- エクスポートに使用する一時作業ディレクトリを特定します。
- エクスポートで生成される ndjson ファイルを収めるうえで十分な容量のストレージ・ディレクトリを指定します。
- BFC 構成を保存するときに、指定したファイル URL を使用して CSP アプリが作成されます。

20.9.3 REST クライアント設定のチェックリスト

- 上記で説明したように、BFC エンドポイントの URL に対する動的クライアント登録を使用して、REST クライアントで使用する OAuth クライアント構成を作成します。
- OAuth を使用している REST クライアントからバルク FHIR エクスポートを開始するとき、またはそのステータスを確認するときは、以下と共にアクセス・トークンを提示します。
  - BFC 構成の [**取得**] タブで特定した**付与タイプ**。
  - OAuth サーバのアクセス・トークン・エンドポイント (issuerEndpoint`/token`) と、必要に応じて対象ユーザ (`?aud=https://`bfcEndpoint)。
  - BFC 構成の [**認証タイプ**] タブに挙げられているいずれかの OAuth **クライアント**のクライアント ID とクライアント秘密鍵。
  - スコープ。最小限必要なスコープは `system/Patient.read` です。`system/*.read` を指定するとすべてが許可されます。

- OAuth を使用している REST クライアントを使用して BFC ファイル・サーバから ndjson ファイルをダウンロードするときは、以下と共にアクセス・トークンを提示します。
  - BFC 構成の **[取得]** タブで特定した**付与タイプ**。
  - OAuth サーバのアクセス・トークン・エンドポイント (issuerEndpoint`/token`) と、必要に応じて対象ユーザ (`?aud=https://`bfcFileEndpoint)。
  - BFC 構成の **[認証タイプ]** タブに挙げられているいずれかの OAuth **クライアント**のクライアント ID とクライアント秘密鍵。
  - スコープ。ファイルのダウンロードでは一般的に `user/*.read` です。

21

従来の FHIR テクノロジ

従来の FHIR® テクノロジの使用方法の詳細は、"インターシステムズの旧ドキュメント・コレクション" で入手可能な FHIR の旧ドキュメントを参照してください。

21.1 従来の変換のアップグレード

インターシステムズ製品の SDA-FHIR 双方向変換をカスタマイズするストラテジは、従来の FHIR テクノロジでは異なっていました (2020.2 以前)。このセクションでは、従来の FHIR 実装で変換をカスタマイズするために開発されたコードを新しい FHIR アーキテクチャに変換する方法について説明します。

変換を行うためにアプリケーションによって呼び出される API が変わりました。従来の実装では、アプリケーションは `HS.FHIR.DTL.Util.API.HC.Transform` クラスのメソッドを呼び出し、変換を起動していました。新たな FHIR アーキテクチャでは、このクラスは廃止されたため、そのメソッドに対する直接呼び出しは機能しません。現在では、変換は、HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR および HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 クラスのメソッドによって呼び出されます。

従来の FHIR テクノロジでは、コールバック・オブジェクトを使用して、変換の実行方法を制御するカスタム・ロジックを実装していました。新しいアーキテクチャでは、カスタマイズは、変換 API クラスのサブクラスを作成し、そのメソッドをオーバーライドすることにより、実現されます。これらの変換メソッドのカスタマイズの詳細は、"変換 API クラスのカスタマイズ" を参照してください。

従来のコールバック・クラスからアップグレードする場合は、コールバック・メソッドのロジックを、新規変換クラスのオーバーライド可能なメソッドに移行する必要があります。以下のテーブルは、従来の `HS.FHIR.DTL.Util.API.HC.Callback.Default.SDA3ToSTU3` クラスのコールバック・メソッドと `HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR` の新しいオーバーライド可能なメソッド間のリレーションシップをまとめたものです。

従来のコールバック・メソッド	新しいオーバーライド可能なメソッド
`IsDuplicate`	`IsDuplicate`
`AssignResourceId`	`GetId`
`GetIdByIdentifier`	`GetId`
`GetPatientId`	`GetId`
`GetURLPrefix`	`GetBaseURL`

以下のテーブルは、従来の HS.FHIR.DTL.Util.API.HC.Callback.Default.STU3ToSDA3 クラスのコールバック・メソッドと HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 の新しいオーバーライド可能なメソッド間のリレーションシップをまとめたものです。

従来のコールバック・メソッド	新しいオーバーライド可能なメソッド
AssignEncounterNumber	GetIdentifier
AssignExternalId	GetIdentifier
GetSendingFacility	GetSendingFacility
GetSendingFacilityFromReference	GetSendingFacility

新しい変換クラスのメソッドの 1 つである GetDTL をオーバーライドして、従来の FHIR テクノロジ向けに記述されたカスタムの DTL クラスを選択することができます。この場合、GetDTL メソッドは古いメソッド GetDTLPackageAndClass を呼び出す必要があります。例を以下に示します。

```
Method GetDTL(source As HS.SDA3.DataType, DTL As %Dictionary.Classname = "") As %Dictionary.Classname
{
  // Get the standard product DTL class name for this SDA3 data type.
  Set className = ##super(source, DTL)

  Set className = ##class(HS.FHIR.DTL.Util.API.ExecDefinition).GetDTLPackageAndClass(className)

  Quit className
}
```

21.1.1 変換プロダクションのアップグレード

FHIR 相互運用プロダクションで変換の実行に使用されるビジネス・プロセス HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process および HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process が更新され、新しい変換 API を使用するようになりました。従来の実装で標準のビジネス・プロセスを使用していた場合は、アップグレード後に以下のタスクを完了してから、プロダクションを開始する必要があります。

- ビジネス・プロセスの FHIRMetadatSet 設定の値を指定します。
- TransmissionMode 設定が Batch に設定されている場合は、この設定を変更して transaction または individual.HS.FHIRServer.Interop.Operation を指定する必要があります。