FHIR Support in InterSystems Products

Version 2024.1
2024-05-02

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

FHIR Support in InterSystems Products
InterSystems Version 2024.1 2024-05-02

For Support questions about any InterSystems products, contact:

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

Table of Contents

List of Tables

1

InterSystems FHIR Components

InterSystems products include the following HL7® FHIR® technologies:

FHIR Server

A FHIR server is an application that receives and processes FHIR requests while leveraging an architecture that is capable of storing and retrieving FHIR resources from an internal repository. In InterSystems products, the out-of-box solution for a FHIR server uses the Resource Repository as its storage. Depending on your product's license, you might not be able to install a FHIR server with the Resource Repository. In this case, you should use the FHIR Interoperability Adapter to receive and process FHIR requests. When using the FHIR server, requests can be routed through an interoperability production before reaching the server's internal architecture, but it is not required; FHIR servers that do not use an interoperability production can be significantly faster.

FHIR Interoperability Adapter

When your application must receive and process FHIR requests, but does not need to store or retrieve resources from internal storage, your best option is to use the FHIR Interoperability Adapter rather than a FHIR server. The FHIR Interoperability Adapter installs the components needed to handle a FHIR request without installing the internal architecture of a FHIR server. The FHIR Interoperability Adapter always uses an interoperability production to process requests.

Transformations

InterSystems products can be used to transform healthcare data captured in a non-FHIR standard such as HL7v2 into FHIR using a set of pre-defined transformations that can be invoked from an interoperability production or directly from an ObjectScript application. Transformations that take FHIR as the input and translate it into another interoperability standard are also provided. At the core of these transformations is the ability to convert FHIR to and from SDA, which is the InterSystems clinical data format.

FHIR Client

Within InterSystems technology, a FHIR client is an interoperability business host or ObjectScript application that makes requests to a FHIR endpoint, whether it is the endpoint of an external FHIR server or the FHIR server architecture within the same InterSystems product. The FHIR client classes provide straightforward methods for performing FHIR interactions and operations on a FHIR server.

Amazon HealthLake Adapters

InterSystems products offer inbound and outbound adapters that allow an interoperability production to retrieve, create, delete and update FHIR resources in an Amazon HealthLake data store.

FHIRPath

FHIRPath is a language that allows you to navigate a FHIR resource to evaluate and extract data from its fields using a straightforward syntax. InterSystems products provide a subset of the FHIRPath functions and operations that you can use to evaluate a resource.

FHIR SQL Builder

The FHIR SQL Builder, or Builder, allows you to project data stored in a FHIR repository into a relational table that can be queried via InterSystems SQL. After analyzing the contents of a FHIR repository, you can select which resources and fields you would like to project into relational tables.

Bulk FHIR Coordinator

InterSystems products include a Bulk FHIR Coordinator that mediates the interaction between a client and a FHIR endpoint for HL7® FHIR® bulk data requests. You can enter a set of configurations. Each configuration identifies a FHIR endpoint, and defines the authorization type, file location, and other parameters to be used in the bulk FHIR interaction.

2

About FHIR

HL7® FHIR®, or Fast Healthcare Interoperability Resources, is a healthcare interoperability standard from HL7 that allows a multitude of systems to exchange healthcare information using agreed upon data models. In FHIR, these data models are simple, straightforward, simultaneously human and computer readable, and, when combined, robust enough to convey complex healthcare information.

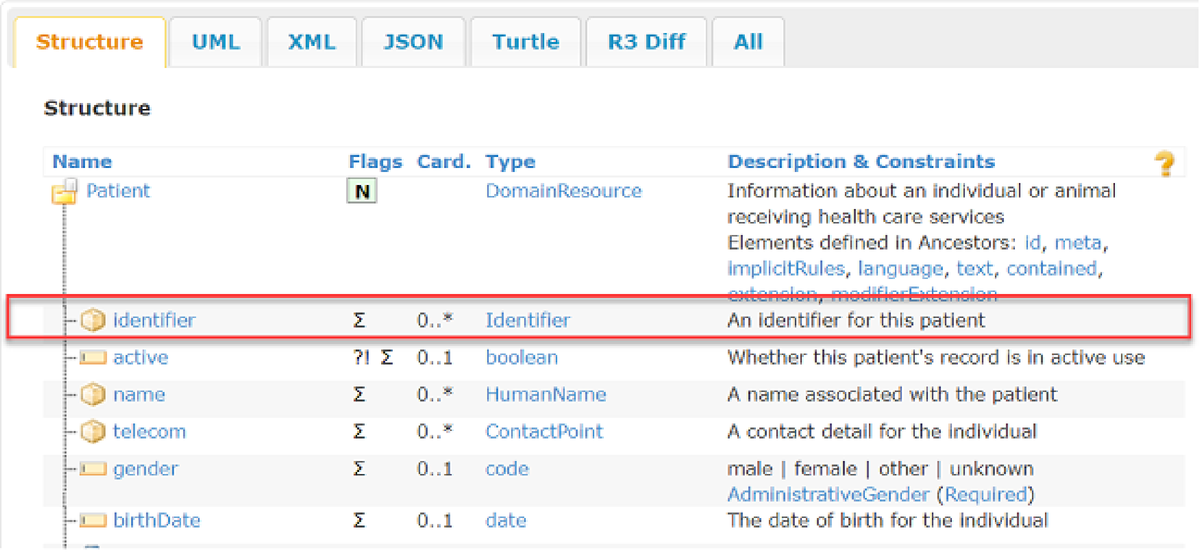
The following is a brief introduction to key concepts in FHIR; these concepts are described in detail in the official FHIR Specification.

2.1 FHIR Resources

FHIR is built on the concept of *resources*, which are discrete units of data represented as JSON or XML. For example, all data about a single patient can be encapsulated as a Patient resource, while information about a single doctor's visit can be captured in an Encounter resource. This Encounter resource would usually contain a reference to the Patient resource of the patient who visited the doctor, avoiding the need to include the patient's data in the Encounter resource itself. Because resources can be stored and retrieved individually using RESTful APIs, FHIR requires less bandwidth and computing resources than other interoperability standards. The ability to express a resource as JSON makes exchanging FHIR data even more lightweight.

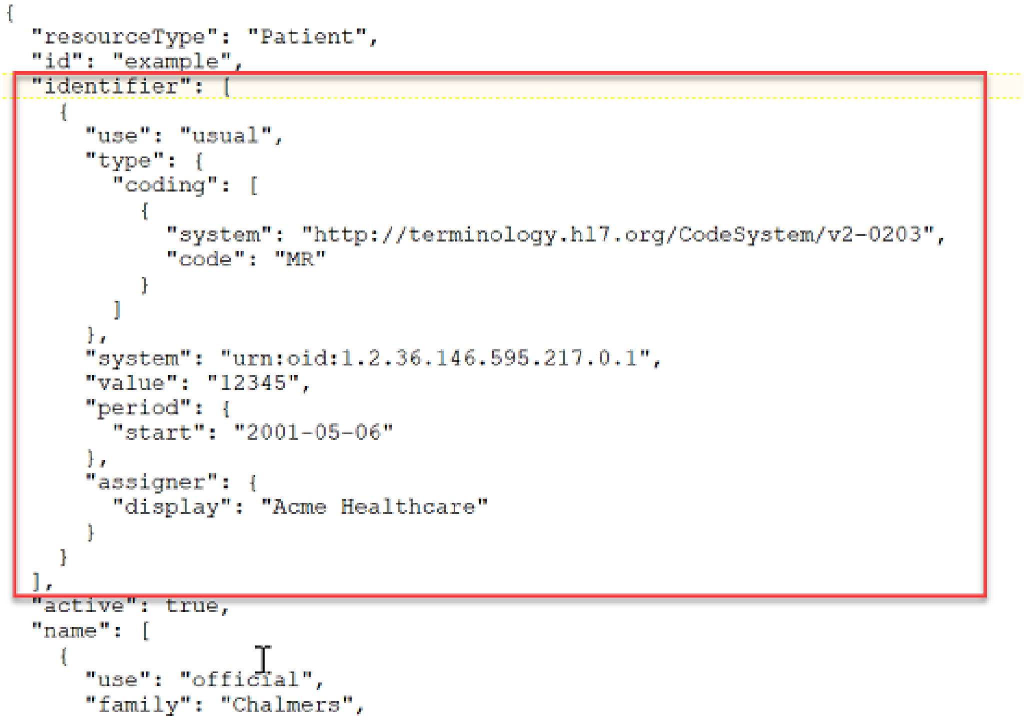
The base FHIR specification contains a page for every supported resource. For example, the Patient resource in the latest FHIR version is found at hl7.org/fhir/patient.html. Core information about a resource, for example what data fields belong in the resource and the data types of those fields, can be found in the Resource Content section of the specification page, which includes a Structure tab that explains each resource field. When starting out with FHIR resources, it is useful to compare a specific example of a resource with this structure (sample resources are available on the Examples tab of each resource page in the specification). A portion of the structure for the Patient resource looks like:

8.1.2 Resource Content



Structure

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual



```
{
  "resourceType": "Patient",
  "id": "example",
  "identifier": [
    {
      "use": "usual",
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
            "code": "MR"
          }
        ]
      },
      "system": "urn:oid:1.2.36.146.595.217.0.1",
      "value": "12345",
      "period": {
        "start": "2001-05-06"
      },
      "assigner": {
        "display": "Acme Healthcare"
      }
    }
  ],
  "active": true,
  "name": [
    {
      "use": "official",
      "family": "Chalmers",
```

For a description of the symbols and icons used on a resource’s **Structure** tab, see Resource Formats.

FHIR also uses resources to define elements of the standard itself. This metadata, known as *conformance resources*, defines things like the valid fields of a resource, the search parameters that can be used to retrieve a resource from a FHIR server, and the codes used within a particular healthcare environment.

For a list of resources currently found in the base FHIR specification, see the Resource Index.

2.2 FHIR Adaptations

FHIR is intended to be adapted for specific healthcare environments and implementations, and provides straightforward strategies for extending and constraining the FHIR standard for these purposes. It often said that FHIR follows a 80/20 rule; the base FHIR specification contains 80% of what your healthcare environment needs, while custom constraints and extensions provide the remaining 20%. Often, a FHIR server conforms to a standard, published Implementation Guide that represents a complete implementation of FHIR for a specific ecosystem. For example, the US Core Implementation Guide sets the standard for using FHIR in healthcare environments in the United States. Of course, a healthcare environment can extend the base FHIR specification, US Core, or another Implementation Guide to meet its own unique needs.

At the heart of a FHIR adaptation are FHIR *profiles*, which extend or constrain a specific resource. For example, the US Core Implementation Guide contains a unique profile for the Patient resource, another profile for the Observation resource, and so on. At a technical level, each profile is defined by a StructureDefinition conformance resource. According to the FHIR specification, the term "profiling" should be reserved for the act of using these StructureDefinitions to configure resources for a particular implementation.

An adaptation of FHIR can contain more than resource profiles. For example, an Implementation Guide can contain codes and search parameters that are unique to a healthcare environment. Similar to profiles, these assets are defined with conformance resources like ValueSet and SearchParameter.

A coherent collection of profiles and other conformance resources is known as a FHIR *package*. The contents of a package can vary widely; it can contain an entire Implementation Guide or a single custom profile. In InterSystems products, you configure a FHIR server to support a particular healthcare ecosystem by adding a package to a FHIR endpoint.

2.3 RESTful APIs

Though FHIR can be used in messaging and document-based frameworks like traditional healthcare interoperability standards, its innovation is the ability to use RESTful API calls to work with healthcare data. Using HTTP verbs like GET and POST, a FHIR client can store, delete, update, and retrieve FHIR resources as needed. These actions that a FHIR client can take on resources are known as *interactions*. For more information about RESTful APIs and supported interactions, see RESTful API in the FHIR specification.

FHIR also allows FHIR clients to use *operations* to perform functions on the FHIR server. Because they invoke functions on the server, these operations are more like RPC calls than RESTful ones. For example, the standard `$validate` operation invokes a function on the server that checks whether a resource conforms to a profile. A healthcare environment can implement custom operations to perform a variety of actions at the request of a FHIR client.

2.4 Searching for FHIR Resources

Search is a very powerful FHIR interaction. Because the healthcare data is stored as individual resources, FHIR clients can use complex queries to retrieve only the data they need without having to parse through unrelated data. These queries are performed with a GET HTTP verb and can leverage search parameters to narrow the results to those resources that meet certain criteria. In its simplest form, a search can retrieve all resources of a certain type without specifying a search parameter. For example, the following RESTful API call would retrieve all Patient resources:

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

You can add search parameters to the API call using the `?` character. For example, a search could use the `name` search parameter to find Patient resources that have a specified value in their name field. The API call to retrieve these Patient resources might be:

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

Multiple search parameters can be chained together using the `&` character. For example, the following API call can further limit the results by adding the gender of the patient:

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

The FHIR specification contains many other standard search parameters that can be used to perform powerful and complex queries. For details, see Search in the FHIR specification. You can find the search parameters for a specific resource on the resource’s page in the specification.

3

FHIR Server: An Introduction

Many implementations can use an out-of-the-box FHIR server that stores and retrieves resources from the InterSystems database using the Resource Repository. This FHIR server can be customized without using an interoperability production or developing an entirely new backend. For details about what HL7® FHIR® interactions are supported by a server that uses the Resource Repository, see Supported Interactions and Operations

FHIR requests can be routed through an interoperability production before reaching the server’s infrastructure, but this is not a requirement. FHIR servers that do not use an interoperability production can be significantly faster.

Though less frequent, it is possible to build a FHIR server with an entirely custom backend; this implementation leverages the same internal architecture used by the Resource Repository, but you must develop your own FHIR processing logic.

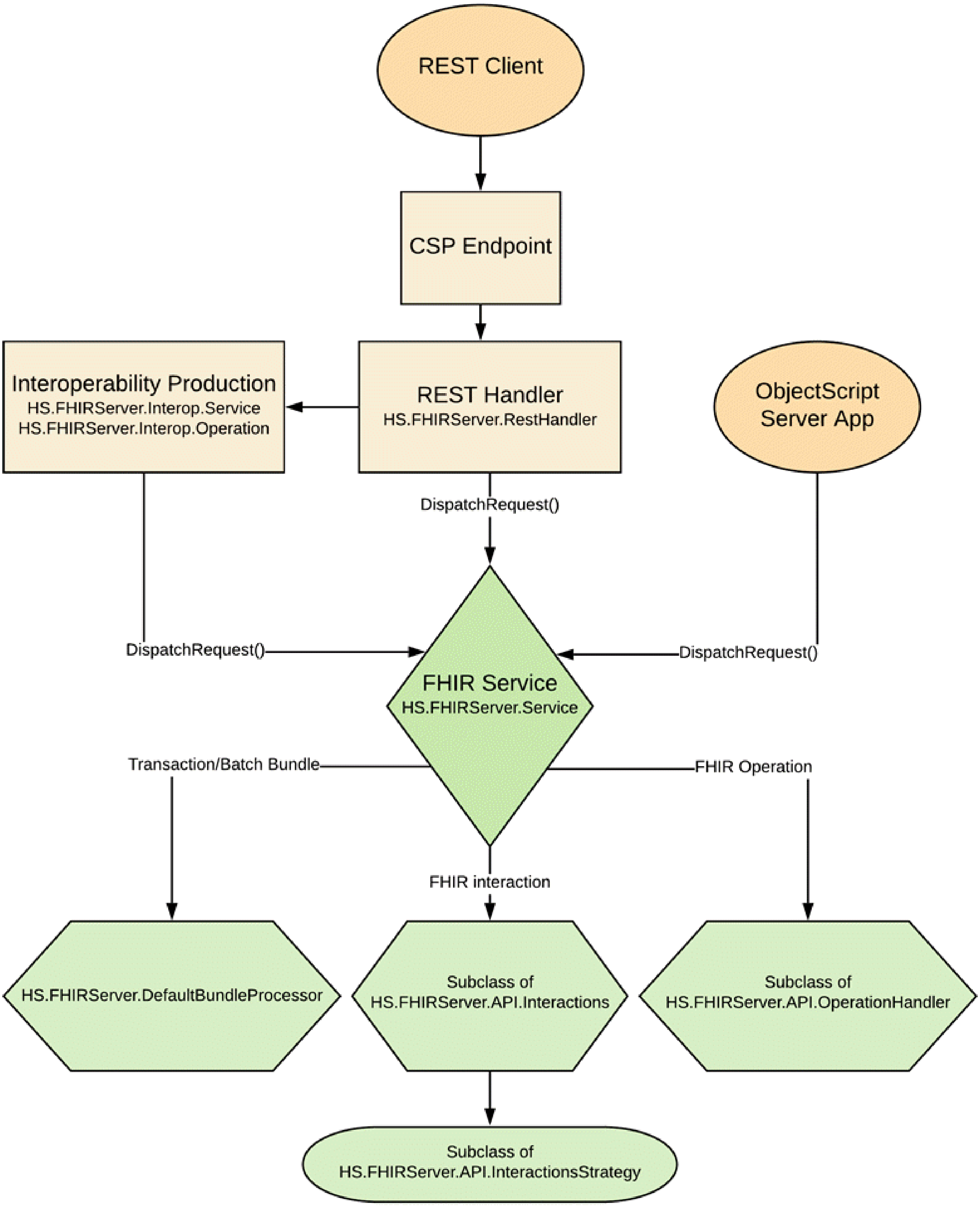
If your InterSystems product is not licensed to install the FHIR server, you can use the FHIR Interoperability Adapter to receive and process FHIR requests through an interoperability production.

3.1 Architecture

FHIR servers using the Resource Repository or a custom backend use the same architecture. Tracing a FHIR request through the FHIR server provides a good overview of the major architectural features of these servers. First, the FHIR request must reach the *Service*, which ensures that the request conforms to the server's FHIR metadata standards and then routes it to the appropriate component to handle the request. The FHIR request can reach this Service in three ways: from a REST handler, through an interoperability production, or from an ObjectScript FHIR Client. This Service is unrelated to a business service in an interoperability production.

What the Service does with the request depends on the type of request:

- If the request contains an HTTP method and endpoint that correspond to a FHIR interaction, the Service forwards it to the method of the *Interactions* class that handles that type of FHIR interaction. For example, requests with a `read` interaction are sent to the `Read()` method of the Interactions class. This Interactions class executes the FHIR interaction, using the *InteractionsStrategy* class to process the interaction according to the overall purpose of the FHIR server.
- For FHIR operations, the Service forwards the request to a special class designed to perform operations. FHIR servers using the Resource Repository offer out-of-the-box support for certain FHIR operations.
- If the request contains a bundle of type `transaction` or `batch`, the Service forwards the request to a special class that unpacks the bundle to perform the individual HTTP operations.


REST Client
CSP Endpoint
Interoperability Production
HS.FHIRServer.Interop.Service
HS.FHIRServer.Interop.Operation
REST Handler
HS.FHIRServer.RestHandler
ObjectScript Server App
DispatchRequest()
DispatchRequest()
DispatchRequest()
FHIR Service
HS.FHIRServer.Service
Transaction/Batch Bundle
FHIR Operation
FHIR interaction
HS.FHIRServer.DefaultBundleProcessor
Subclass of HS.FHIRServer.API.Interactions
Subclass of HS.FHIRServer.API.OperationHandler
Subclass of HS.FHIRServer.API.InteractionsStrategy

3.1.1 More About the Service

The Service is a singleton class that allows only one instance of itself to be instantiated for an endpoint. This instantiation occurs when the first FHIR request is sent to the Service by the REST Handler or Business Operation; once instantiated, the Service exists until the process ends. For server applications making FHIR requests programmatically, the app must call **HS.FHIRServer.Service.EnsureInstance()** to retrieve the Service before sending the first request.

In most cases, the Service class (HS.FHIRServer.Service) is ready to uphold the endpoint's FHIR standard and route requests without being subclassed. Custom logic that determines how the FHIR server behaves is written into the Interactions and InteractionsStrategy subclasses, not the Service.

The methods that manage the Service, including creating a new Service for an endpoint and deleting a Service, belong to the subclass of HS.FHIRServer.API.RepoManager.

3.1.2 More About the InteractionsStrategy

The InteractionsStrategy class dictates the overall strategy for the FHIR server. It is the FHIR server application's backend, creating and implementing the environment in which the FHIR data is processed. The InteractionsStrategy superclass is HS.FHIRServer.API.InteractionsStrategy.

In many cases, the InteractionsStrategy is the "storage strategy" for how the FHIR server stores and retrieves FHIR resources. For example, the Resource Repository is implemented by a subclass of HS.FHIRServer.API.InteractionsStrategy that creates the resource and index tables used to store and retrieve the FHIR data. In applications that are not storing FHIR data, the strategy might set up an environment that communicates with an external FHIR server or any other custom logic that works with the server's FHIR data.

An InteractionsStrategy is associated with a subclass of HS.FHIRServer.API.RepoManager that manages the Services that use the InteractionsStrategy.

3.1.3 More about the Interactions Class

While the InteractionsStrategy class is the backend of the application, it uses the Interactions class to actually execute the FHIR interactions received by the Service. During this process, the Interactions class often calls methods in the InteractionsStrategy class, especially for structures and logic that are common to the entire FHIR server strategy. Because of their interdependent relationship, the Interactions class and InteractionsStrategy class are subclassed together in a unified approach. The Interactions superclass is HS.FHIRServer.API.Interactions.

The methods in the Interactions class that are called by the Service when processing a FHIR request can also be called directly from a server-side ObjectScript application. For example, a server-side application could call the **Add()** method of the Interactions class rather than sending a `POST` request to the Service. In bypassing the Service, the server application can bypass any restrictions placed on the FHIR server by the Service's metadata. For example, the server application could populate the FHIR server's storage even though the endpoint is read-only for requests going through the Service.

The Interactions class also keeps track of which specialized classes the Service should use to perform FHIR operations, process bundles, and validate FHIR data. The Service obtains the name of these classes from the Interactions object when it needs to take action.

3.2 Resource Repository

The Resource Repository is the default storage strategy for a FHIR server, allowing you to install a fully functioning FHIR server without further development tasks. It automatically stores FHIR data received by the server as dynamic objects that encapsulate the JSON data structures of the FHIR data. To install a FHIR server that uses the Resource Repository, select HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy as the Interactions Strategy Class during installation.

Note: Prior to version 2024.1, the Resource Repository architecture was implemented using classes from the HS.FHIRServer.Storage.Json package. These legacy architecture classes are still supported in this version; however, they provide a limited set of features compared to the current classes, described in this documentation.

If you have upgraded an instance with a preexisting Resource Repository to this version from a version prior to 2024.1, see JSON Legacy SQL Strategy for a comparison of supported features and instructions for upgrading your Resource Repository from the legacy classes to the current classes.

For a list of the FHIR interactions and operations that are available for a FHIR server that uses the Resource Repository, see Supported Interactions and Operations.

The Resource Repository consists of the following architectural classes:

Architectural Component	Resource Repository Class
Interactions	HS.FHIRServer.Storage.JsonAdvSQL.Interactions
InteractionsStrategy	HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy
RepoManager	HS.FHIRServer.Storage.JsonAdvSQL.RepoManager

You can subclass the Resource Repository to customize the FHIR server. For more information, see Customizing a FHIR Server.

4

Installing and Configuring a FHIR Server

The Management Portal provides a **Server Configuration** page that allows you to install a new FHIR server and then configure it. Alternatively, you can install and configure a server programmatically.

The FHIR server must be installed in a Foundation namespace; multiple FHIR servers can be installed in the same Foundation namespace.

Important: Before installing a FHIR server, you must consider whether you want to customize it now or in the future. In many cases, a FHIR server using the Resource Repository cannot be customized unless you subclass the InteractionsStrategy *before* creating the endpoint. For example, modifying how bundles are processed or post-processing search results requires you to subclass the Resource Repository. For information about preparing for these customizations before installing the FHIR server, see Pre-Installation Subclassing.

To install a new FHIR server from the Management Portal:

1. Open the Management Portal and switch to the Foundation namespace where you want the FHIR server installed. If you do not have a Foundation namespace, follow the creating a foundation namespace procedure to create and activate a foundation namespace before you begin.
2. Navigate to **Health** > *MyNamespace* > **FHIR Configuration**. If you do not see the **FHIR Configuration** menu, make sure you are using a Foundation namespace.
3. Select the **Server Configuration** card.
4. In the **Endpoints** pane, click **Add Endpoint** to create a new FHIR Endpoint.
5. Select a core FHIR package. Each package corresponds to a version of the HL7® FHIR® standard which the endpoint will support. So, for example, to configure a FHIR endpoint that supports FHIR R5, select the `hl7.fhir.r5.core@5.0.0` package.
6. Review the endpoint URL that has been autogenerated according to your choice of the core FHIR package. You can change the endpoint's URL, but ensure that it begins with a slash (`/`).
7. If you want the endpoint to support additional packages, select them from the **Additional Packages** drop-down list. For more information about packages, see Profiles and FHIR Adaptations.
8. Select the InteractionsStrategy for the endpoint. The default interactions strategy is the Resource Repository (HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy), which stores FHIR data as JSON in dynamic objects. If you created a custom InteractionsStrategy, select it from the list.
9. By default, data for each endpoint in a namespace is stored in two separate databases. If you do not want to maintain separate databases, clear the **Use separate databases for FHIR resource storage** field; in this case, all FHIR data is stored in the namespace's common database files. If you use separate databases. you can accept the default locations or

specify your own. The Resource History database contains previous versions of a resource; because these are not accessed as frequently, you could put this database on a slower, less expensive disk.

10. Select **Add**.

If you prefer to use a command-line interface to install a FHIR server, see Command Line Options.

If you plan to mirror your FHIR server, see Mirroring Considerations for Healthcare Products for special instructions.

4.1 Configuring a FHIR Server

Once you have installed a FHIR server, you can configure its settings using the **Server Configuration** page of the Management Portal. These configuration settings can also be modified programmatically by setting the properties of the server's ConfigData object.

To configure the FHIR server:

1. Navigate to **Health** > *MyNamespace* > **FHIR Configuration**. Make sure you are in the FHIR server's namespace.
2. Select the **Server Configuration** card.
3. Choose the endpoint of the FHIR server that you are configuring.
4. When the page expands, scroll down and select the **Edit** button.
5. Configure the settings, using the following descriptions as a guide.

Setting	Description
Enabled	Specify whether the endpoint is enabled. A disabled endpoint rejects requests from FHIR clients.
Default Search Page Size	Search result page size to use when a search does not contain a `_count` parameter.
Max Search Page Size	Maximum search result page size to prevent an excessive user-specified page size.
Max Search Results	Maximum number of resources that can be selected by a search before the server responds to the query with an error. This number only includes resources selected by the actual search; it does not include resources included via an `_include` search parameter. This value does not affect the size of pages returned by a search. Overly broad searches that select large numbers of resources take a lot of system resources to fulfill, and are probably more broad than the client actually needs.
Max Conditional Delete Results	Maximum allowable number of resources to delete via conditional delete. If the conditional delete search finds more than this number of resources, then the conditional delete as a whole is rejected with an HTTP 412 Precondition Failed error.

Setting	Description
FHIR Session Timeout	Maximum number of seconds between requests to the service before any session data is considered stale.
Default Prefer Handling	Specifies what happens by default when a search request contains an unknown parameter. Specify `lenient` to ignore the unknown parameter and return a bundle in which the OperationOutcome resource identifies the issue. Specify `strict` to reject the search request and return an error. A FHIR search request that includes the prefer header overrides this default.
OAuth Client Name	Specifies the application name that the FHIR server, as an OAuth resource server, uses to contact the OAuth 2.0 authorization server when needed. For more information about OAuth 2.0 support, see OAuth 2.0 Authorization.
Required Resource	If you specify an InterSystems security resource, FHIR clients must have privileges to the resource to perform interactions on the server. For more information, see Adding Authorization Requirements.
Service Config Name	To route FHIR requests through an interoperability production before reaching the FHIR server, enter the package and name of the business service that will receive the requests. Unless the business service has a custom name, this entry is `HS.FHIRServer.Interop.Service`. For more details, see Interoperability Productions.
Allow Unauthenticated Access	Allows all FHIR requests to reach the server, ignoring authentication and authorization strategies.
New Service Instance	Instantiates a new Service object for every FHIR request.
Include Tracebacks	The FHIR server responds to a FHIR request by sending a stack trace in an OperationOutcome resource.
SMART on FHIR Capabilities	A comma-delimited list of the endpoint's SMART on FHIR capabilities. This list does not control the functionality of the endpoint; rather, it specifies the capabilities that are returned in the JSON document when a client appends `/.well-known/smart-configuration` to the endpoint's URL. For more details about SMART on FHIR capabilities retrieved with Well-Known URIs, see FHIR Authorization Endpoint and Capabilities Discovery using a Well-Known Uniform Resource Identifiers (URIs).

If you prefer to use a command-line interface to configure a FHIR server, see Command Line Options.

Note: If you expect to post Bundles containing 10,000 or more entries, you should increase the value of the Web Gateway Server Response Timeout parameter to avoid server timeouts interrupting your data loads.

4.2 Deleting a FHIR Endpoint

By default, using the Management Portal to delete an FHIR server endpoint also deletes the FHIR data associated with the endpoint. However, if you want to delete an endpoint but retain all of its FHIR data, you can use the command line interface to decommission the endpoint rather than delete it. For more information about using the command line interface to decommission an endpoint, see Command Line Options.

To delete an endpoint:

1. Navigate to **Health** > *MyNamespace* > **FHIR Configuration**. Make sure you are in the FHIR server’s namespace.
2. Select the **Server Configuration** card.
3. Choose the endpoint that you are deleting.
4. Select the Trash Can icon.

4.3 Installing Programmatically

For applications that need to install a FHIR server programmatically rather than using the Management Portal, the server must be installed first, then configured.

The FHIR server must run in a foundation namespace, therefore creating a foundation namespace is a prerequisite to installing the FHIR server. Once you have a foundation namespace, the following methods of HS.FHIRServer.Installer must be called in order:

HS.FHIRServer.Installer method	Description
InstallNamespace()	Prepares an existing foundation namespace for the FHIR server; it does not create a new foundation namespace. If called without an argument, the installer assumes the active namespace is a foundation namespace and prepares it for the FHIR server.
InstallInstance()	Installs an instance of a FHIR Service into the current namespace. This method requires the following arguments: • Unique URL of the FHIR endpoint. Be sure the URL begins with a slash (`/`). • Classname of the FHIR server's InteractionsStrategy. • List of FHIR packages, for example, the package for an Implementation Guide like US Core. For details, see pPackageList parameter. There are also optional parameters that can be passed to **InstallInstance()**. For complete details on these optional parameters, see **InstallInstance()**

4.3.1 `pPackageList` Parameter

The `pPackageList` parameter of the **InstallInstance()** method accepts a list of FHIR packages that have been loaded into the system. Often, a package corresponds to a specific Implementation Guide, but can also be the core metadata for a version of FHIR. By passing a list of packages to `InstallInstance`, you can configure an endpoint to support one or more packages. For more about packages, see Profiles and FHIR Adaptations.

To obtain a list of the packages that can be passed into the `pPackageList` parameter, use the **HS.FHIRMeta.Storage.Package.GetAllPackages()** method. For example, the following code displays the identifiers of the available packages

ObjectScript

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

The result might look like:

```
hl7.fhir.r5.core@5.0.0
hl7.fhir.r4.core@4.0.1
hl7.fhir.us.core@3.1.0
hl7.fhir.r3.core@3.0.2
```

You could then pass in some of these package identifiers as arguments to the `pPackageList` parameter using `$lb`. For example:

ObjectScript

```
 Do ##class(HS.FHIRServer.Installer).InstallInstance(
            myURL,
            strategyClass,
            $lb("hl7.fhir.r5.core@5.0.0"))
```

For details about the APIs used to create FHIR packages, see Package APIs.

4.3.2 Programmatic Install Example

The following ObjectScript code example installs a FHIR server that supports two packages and uses the default storage strategy (Resource Repository).

ObjectScript

```
Set appKey = "/myfhirserver/fhir/r5"
Set strategyClass = "HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy"
Set metadataPackages = $lb("hl7.fhir.r5.core@5.0.0")

//Install a Foundation namespace and change to it
Do ##class(HS.Util.Installer.Foundation).Install("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 Configuring Programmatically

Once you have installed a FHIR server, it can be configured programmatically using the HS.FHIRServer.Installer.UpdateInstance() method. This method accepts several arguments that configure the server, including one that accepts the server's HS.FHIRServer.API.ConfigData object, which contains most of the server's configuration options. For a list of these configuration options, see the class reference. In addition to the options defined with the ConfigData object, a few of the server's settings (Service Config Name, OAuth Client Name, and Enabled) are specified using a dedicated parameter of the **UpdateInstance()** method.

The following code configures an existing FHIR server using the **UpdateInstance()** method.

ObjectScript

```
Set appKey = "/fhirendpoint/r5"

//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)
```

Note: Like all InterSystems IRIS APIs that act on code in a repository, HS.FHIRServer.Installer.UpdateInstance() locks the repository to prevent simultaneous configuration activities and holds the lock until configuration is complete. Before performing configuration tasks on your FHIR server using methods other than InterSystems IRIS APIs, execute the **Lock()** method of the HS.FHIRServer.Repo class to lock the repository explicitly, as follows: `##class(HS.FHIRServer.Repo).Lock()`. If you completely override an InterSystems IRIS method, remember to use the **Lock()** method to prevent conflicts.

4.5 Command Line Options

Developers who prefer a command line interface to the Management Portal can use Console Setup in the InterSystems Terminal to perform many of the same actions that are available in the user interface. To run the Console Setup, open the InterSystems Terminal and run:

ObjectScript

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

The following sections describe each option that is available in the Console Setup.

Create FHIRServer Endpoint

Installs a new FHIR server endpoint. You are presented with the following prompts:

- `Choose the Storage Strategy` — `Json` is the Resource Repository.
- `Choose the FHIR version for this endpoint` — Select the version of the core FHIR specification that your endpoint supports.
- `Enter any package numbers` — Packages that have been imported are listed as possibilities. The endpoint can support multiple packages; to specify more than one package, separate the numbers by commas. You can add additional packages later, but you might need to run additional steps if you wait. Use the `Upload a FHIR Metadata Package` option to add a package to the list.
- `Do you want to create the default repository endpoint` — Press `Enter` if you want to accept the default URL of the endpoint. If you want your endpoint to have a different URL, specify `N`, and enter the URL (be sure the URL begins with a slash).
- `Enter the OAuth Client Name for this Endpoint` — If you are using OAuth 2.0 to secure the endpoint, enter the Client Name of the FHIR server. For more information, see OAuth 2.0 Authorization.
- `Do you want to create separate database files for your FHIR data?` — If you specify `yes`, FHIR data for the endpoint is stored separately from the FHIR data of other endpoints in the same namespace. If you specify `no`, all FHIR data is stored in the namespace’s database files, even if you have multiple endpoints. If you are creating separate database files, you can accept the default locations or specify alternate locations. The Versions Database contains previous versions of a resource; because these are not accessed as frequently, you could put the Versions Database on a slower, less expensive disk.

Add a profile package to an endpoint

Adds a FHIR package to an existing endpoint so it can support the package’s profiles, search parameters, and other conformance resources. The FHIR package (an NPM-like package) that contains the conformance resources must be uploaded before you can use this option. You can use the `Upload a FHIR Metadata Package` option to import the FHIR package. Some common packages, for example the US Core Implementation Guide, are already available.

If the package contains new search parameters, you must run the `Index new SearchParameters for an Endpoint` option when you are done.

Display a FHIRServer Endpoint Configuration

Displays the current configuration options of the FHIR server. To modify these configuration options, use the `Configure a FHIRServer Endpoint` option.

Configure a FHIRServer Endpoint

Allows you to configure the FHIR server endpoint by providing values for each configuration option. For a description of each configuration item, see Configuring a FHIR Server.

Decommission a FHIRServer Endpoint

Deletes a FHIR server endpoint, but retains the FHIR data that has been collected by the endpoint. The SQL tables containing the FHIR data are retained. If you want to delete the endpoint *and* all of the FHIR data, use the `Delete a FHIRServer Endpoint` option.

Delete a FHIRServer Endpoint

Deletes a FHIR server endpoint *and* deletes the endpoint's FHIR data. If you want to delete the endpoint, but retain the FHIR data that has been collected by the endpoint, use the `Decommission a FHIRServer Endpoint` option.

Update the CapabilityStatement Resource

Updates the Capability Statement of the FHIR server. For more details, see Modifying the Capability Statement.

Index new SearchParameters for an Endpoint

When you add new search parameters to an *existing* endpoint using a published or custom package, FHIR clients can use the new parameter to retrieve resources added to the repository *after* you applied the package. However, resources that existed *before* you added the new search parameter will not be returned until you re-index the endpoint. If an endpoint has collected a large volume of FHIR data, this option can take a long time to run as it re-processes all existing resources.

Upload a FHIR metadata package

Used to import a FHIR package of JSON files that define conformance resources. You must use this option before the package can be applied to an endpoint. For information about preparing a custom FHIR package for uploading, see Creating a Custom Package.

Delete a FHIR metadata package

Deletes a package from the list of available packages that can be applied to an endpoint. This does not delete the FHIR package's JSON files from your local system. You cannot delete packages that have been applied to an endpoint.

4.6 Configuring the Profile Validation Server

When you create a FHIR endpoint, an external server named FHIR_Validation_Server is created to perform back-end functions related to profile validation. This server requires a Java 11 development kit. If your *JAVA_HOME* environment variable does not point to a Java 11 directory, you can use the Management Portal as follows:

1. If necessary, install a supported Java 11 JDK. Make a note of the directory where it has been installed.
2. In the Management Portal, navigate to **System Administration** > **Configuration** > **Connectivity** > **External Language Servers**.
3. If the FHIR_Validation_Server is running, click **Stop**.
4. Enter edit mode by clicking **FHIR_Validation_Server**.
5. On the **Edit External Language Server** page, in the **Java Home Directory** field, enter the path to your Java 11 directory.

6. Click **Save**.
7. Restart the FHIR_Validation_Server by clicking **Start**.
8. To ensure good performance when you execute validation operations related to previously-imported profiles (including those that are automatically imported), in the Terminal application, switch to your FHIR-enabled namespace and execute the following command:

   ```
   do ##class(HS.FHIRServer.Installer).InitializeProfileValidator()
   ```

Note: Do not set the *JAVA_HOME* environment variable directly to enable the FHIR_Validation_Server; doing so could affect other applications and processes that may rely on the previous value of *JAVA_HOME*.

4.7 Optimizing Search Performance

For a FHIR server which uses or extends the Resource Repository, you can optimize the performance of search interaction responses by running the Tune Table utility on the SQL search tables generated for that endpoint. By default, the names of these tables begin with `HSFHIR`. You can also set custom selectivity values for these tables manually.

When you install a new FHIR server which uses or extends the Resource Repository, a set of default selectivity values are assigned to searchable elements based on the element's data type. These default selectivity values allow the server to select more efficient query plans when it retrieves resources in response to a search request. Selecting prospective results by beginning with the most selective query parameter minimizes the number of resources each subsequent selection operation must evaluate.

For example, consider the following search request:

```
GET [base]/Patient?family=halifax&gender=male
```

Probably, the Resource Repository contains fewer patients with the family name Halifax than patients who are male. Therefore, it is probably most efficient to find the Patients with the family name Halifax first, because then the operation to find males will only have to search through the small subset of patients named Halifax.

If you have upgraded to this version from a version prior to 2023.1 and you are using a preexisting FHIR server, you can set these default selectivity values for your search operations using the SetDefaultSearchTableSelectivities() method. Invoke this method in the Terminal, providing the relative path for your FHIR endpoint, as in the following example:

Terminal

```
do
##class(HS.FHIRServer.Storage.SearchTableBuilder).SetDefaultSearchTableSelectivities("/csp/healthshare/hsods/fhir/r5")
```

However, for most preexisting FHIR servers, selectivity values generated by Tune Table are more useful than the default selectivity values set by this method.

Important: The SetDefaultSearchTableSelectivities() method overwrites all existing optimizations made manually or using the Tune Table utility.

5

Supported FHIR Interactions and Operations

When using the Resource Repository storage strategy provided with the FHIR server, the server supports the following interactions and operations. If your custom FHIR server extends the Resource Repository, it also supports these interactions and operations by default.

5.1 Interactions

HL7® FHIR® interactions are the set of actions that a FHIR client can take on resources. These interactions can be grouped according to whether they act upon an instance, a type, or the whole system. An instance is a specific instance of a resource, for example, `Patient/1` refers to an instance of a Patient resource with an `id` of `1`. A type refers to a particular FHIR resource, for example, a Patient or Observation.

The following table summarizes the support for FHIR interactions in the Resource Repository, or a custom FHIR server that has extended the Resource Repository. Click on an interaction to see how it is defined in the HL7 REST API and how to use it.

Interaction	Level of Support
create	Fully supported, including conditional create.
read	Conditional read is not supported.
vread	Conditional read is not supported.
update	Fully supported, including conditional update.
patch	Supported for JSON patch documents only. Conditional patch is supported.
delete	Fully supported, including conditional delete.
history	Supported for instance interactions only, not type or system. For example, `GET [baseURL]/Patient/1/_history` is supported, but not `GET [baseURL]/Patient/_history` or `GET [baseURL]/_history`. The `_count` and `_at` parameters are not supported. Paging is not supported.
batch	Fully supported
transaction	Circular references within the bundle are not supported.
search	Supported with some limitations. For details, see Search Interaction.

5.1.1 Search Interaction

FHIR clients use the search interaction to retrieve resources from the Resource Repository. For full details about the search interaction, refer to FHIR specification. This section summarizes the default support for the search interaction when the FHIR server is using or extending the Resource Repository.

Note: Prior to version 2024.1, the Resource Repository's search interaction was implemented using a different search strategy. This legacy strategy is still supported in this version; however, they provide a limited set of features compared to the current strategy, described in this documentation.

If you have upgraded an instance with a preexisting Resource Repository to this version from a version prior to 2024.1, see JSON Legacy SQL Strategy for a comparison of supported features and instructions for upgrading your Resource Repository from the legacy strategy to the current strategy.

5.1.1.1 General Support Notes

Keep in mind that a FHIR server using or extending the Resource Repository:

- *Does not* support searching across multiple resource types across compartments. For example `GET [base]?_id=1` is not supported.
- *Does* support the use of the wildcard character (`*`) to search across multiple resource types within a compartment. For example: `GET [base]/Patient/100000001/*` is supported. Within a compartment, the search can use the wildcard in conjunction with any search parameters common to all the resource types which the search is targeting. This is especially useful if you use the `_type` parameter. For example: `GET [base]/Patient/100000001/*?status=final&_type=Observation,DiagnosticReport` is supported because both the Observation and DiagnosticReport resource type include the `status` element.

Note: The wildcard syntax provides a different result than the `$everything` operation. `GET [base]/Patient/100000001/*` returns any and all resources associated with the specified Patient—including, for example, a Patient's DiagnosticReport resources. By contrast, `GET [base]/Patient/100000001/$everything` returns all resources associated with the Patient resource as well as resources associated with those resources. Compared with the previous search, this search would also include Practitioner resources associated with the Patient's DiagnosticReport resources.

5.1.1.2 Search Parameter Types

Each search parameter has a search parameter type that determines how the parameter behaves.

Parameter Type	Level of Support
composite	Not supported
date	Fully supported
number	Fully supported
quantity	Fully supported
reference	Fully supported*
string	Fully supported
token	Fully supported
uri	Fully supported

* For canonical references, chained search is not supported. The use of the search result parameters `_include` and `_revinclude` for canonical references is also not supported.

5.1.1.3 Parameters

The following summarizes FHIR server support for standard search parameters when retrieving resources from the Resource Repository.

Parameter	Level of Support
`_content`	Not supported
`_filter`	Not supported
`_has`	Fully supported as described in the official specification
`_id`	Fully supported as described in the official specification
`_lastUpdated`	Fully supported as described in the official specification
`_list`	Not supported
`_profile`	Fully supported as described in the official specification
`_query`	Not supported

Parameter	Level of Support
`_security`	Fully supported as described in the official specification
`_source`	Fully supported
`_tag`	Fully supported as described in the official specification
`_text`	Not supported
`_type`	Fully supported.

5.1.1.4 Modifiers

Modifiers can be added to the end of a parameter to affect the results of the search.

Modifier	Level of Support
`:above`	Supported for URI
`:below`	Supported for URI
`:code-text`	Not supported
`:contains`	Fully supported (strings and URIs)
`:exact`	Fully supported
`:identifier`	Fully supported
`:in`	Not supported
`:iterate`	Fully supported
`:missing`	Not supported
`:not`	Not supported
`:not-in`	Not supported
`:of-type`	Fully supported
`:text`	Supported for references and tokens, not supported for strings
`:text-advanced`	Not supported
`:[type]`	Fully supported

5.1.1.5 Prefixes

When using search parameters of type number, date, and quantity, you can add a prefix to the parameter's value to affect what resources match the search. For example, `[parameter]=le100` returns values that are less than or equal to 100.

Prefix	Level of Support
`eq`	Fully supported
`ne`	Fully supported
`gt`	Fully supported
`lt`	Fully supported
`ge`	Fully supported
`le`	Fully supported
`sa`	Fully supported
`eb`	Fully supported
`ap`	Fully supported

5.1.1.6 Search Result Parameters

Search result parameters help manage the resources returned by a search.

Search result parameter	Level of Support
`_contained`	Not supported
`_containedType`	Not supported
`_count`	Fully supported as described in the official specification
`_elements`	Fully supported as described in the official specification
`_graph`	Not supported
`_include`	Fully supported as described in the official specification
`_maxresults`	Not supported
`_revinclude`	Fully supported as described in the official specification
`_score`	Not supported
`_sort`	Fully supported as described in the official specification
`_summary`	Supports `_summary=count`, `_summary=data`, and `_summary=text`. For details, see the official specification.
`_total`	Not supported

5.2 Operations

For FHIR servers using or extending the default Resource Repository, the following operations are supported:

Operation	Level of Support
`$everything`	Fully supported for Patient and Encounter. Not supported for EpisodeOfCare, Group, MedicinalProduct, or MedicinalProductDefinition.
`$lastn`	Fully supported
`$validate`	Partially supported. See "Profile Validation".

5.2.1 Operation Query Parameters

For specific operations, certain operation query parameters are supported:

Operation	Query Parameter
`$everything`	• `_since` is supported for Patient and Encounter, and accepts a dateTime value. • `_type` is supported for Patient and Encounter. See "Recursive Behavior of the _type Operation Query Parameter for $everything" for details.
`$lastn`	`max` is supported.

5.2.1.1 Recursive Behavior of the _type Operation Query Parameter for $everything

When a list of resource types is provided in the `_type` query parameter for the `$everything` operation, the compartment search will return only resources of the type listed. Recursive resource reference retrieval in the compartment will skip over references to resource types that are not specified in the `_types` parameter. Some examples illustrate how the `_type` query parameter for `$everything` operates on the Patient compartment:

1. `/Patient/123/$everything?_type=DiagnosticReport,Observation` — returns DiagnosticReport and Observation resources but *not the Patient resource*.
2. `/Patient/123/$everything?_type=Observation` — returns the patient's Observation resources, even though the referring DiagnosticReport resources *are not included*, because Observation is also in the Patient compartment.
3. `/Patient/123/$everything?_type=Practitioner` — returns nothing. Practitioner is *not* in the Patient compartment, and no other resource type that could refer to Practitioner was specified.
4. `/Patient/123/$everything?_type=Patient,DiagnosticReport,Practitioner` — returns the Patient resource, all of the DiagnosticReport resources, and *only* the Practitioner resources directly referred to by the returned DiagnosticReport resources.

5.2.2 Profile Validation

Note: Profile validation is currently designed to work only with FHIR version R4. Later releases will target R5.

Intersystems IRIS for Health supports profile validation by implementing part of the FHIR standard for the `$validate` operation, which checks a resource against the most recent version of a specified profile.

The following query syntax options are supported:

- You can specify the profile in the query URL:

  Table 5–1:

Query URL	`POST <FHIR Endpoint>/<Resource Type>/$validate?profile=<Profile URL>|<Profile Version Number>`
Body	Resource details in XML or JSON format

  <Profile Version Number> is required. Note that the character separating *<Profile URL>* from *<Profile Version Number>* is a pipe, not a slash.

- You can provide the profile in the query body, optionally specifying a supported mode:

  Table 5–2:

Query URL	`POST <FHIR Endpoint>/<Resource Type>/<Optional Resource ID>/$validate`
Body	A `Parameters` block, which must include the resource details, and which may include the mode and a profile, in XML or JSON format.

  Providing a profile is optional. If no profile is provided, validation is performed based on the core schema for the resource type.

  `<Optional Resource ID>` may be required or forbidden, based on the value of the `mode` parameter, as follows:

  Table 5–3:

Value of mode	*ID Required?*	*Checks Performed by $validate*
`create`	Forbidden	`$validate` operation confirms that no resource ID is included, and compares the potential new resource to the profile or the core schema for the resource type, if no profile is provided.
`update`	Required	`$validate` operation checks that a resource ID is included in the request URL and matches the ID in the query body, that the resource URL resource type matches the resource included in the query body, and compares the potential outcome of the update to the profile or the core schema for the resource type, if no profile is provided.
`delete`	Required	`$validate` operation checks to ensure that a resource ID is included in the request URL. This ID is required for the deletion operation, but no profile validation occurs.
`profile`	Required	`$validate` operation checks to ensure that a resource ID is included in the request URL, and that a profile is specified either in the request body (in a `Parameters` resource) or as a query parameter in the request URL.
unspecified	Forbidden	`$validate` operation checks the resource in the query body against the profile or the core schema for the resource type, if no profile is provided.

5.3 Migrate from Pre–2020.1 Resource Repository

For FHIR servers developed using InterSystems IRIS for Health 2019.4 or earlier, the data in the old Resource Repository must be migrated before using the new FHIR server architecture. To migrate your FHIR data:

1. In the Management Portal, switch to the namespace of your pre-2020.1 FHIR server, and then create a STU3 endpoint.
2. Open the InterSystems Terminal and navigate to the namespace of your pre-2020.1 FHIR server.
3. Run:

   ObjectScript

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

4. Select the STU3 endpoint and confirm the migration.

6

FHIR Profiles and Adaptations

The HL7® FHIR® standard is intended to be adapted for specific healthcare environments and implementations. At the core of these adaptations are FHIR *profiles*, which define the allowable fields of a specific resource. These profiles extend or constrain the resource definitions that are found in the base FHIR specification. Profiles and other FHIR artifacts are achieved through *conformance resources*; for example, profiles are defined by StructureDefinition resources, search parameters are defined by SearchParameter resources, codes are defined by ValueSet and CodeSystems resources, and so on.

In most cases, a complete, robust FHIR adaptation is defined by an *Implementation Guide*, which is a coherent collection of conformance resources that includes documentation explaining the adaptation-specific profiles and other artifacts. Most commonly, these Implementation Guides are distributed as NPM-like packages that are downloadable from distribution sites. In InterSystems products, you control what a FHIR server supports by adding a FHIR *package* of conformance resources to an endpoint, even when the package does not contain an entire Implementation Guide.

An InterSystems FHIR endpoint can support multiple FHIR packages. For example, a FHIR endpoint can support the package of the US Core Implementation Guide while simultaneously supporting a unique Patient profile or search parameter from a custom package. This allows FHIR clients to search and use resources that conform to all of the supported packages.

In adherence to the FHIR specification, an InterSystems FHIR server does not automatically verify whether a resource that it receives from a FHIR client conforms to a supported profile. The FHIR client asserts that a resource conforms to one or more profiles using the `meta` element of the resource, but the FHIR server does not check whether that assertion is true. A FHIR client can use the `_profile` search parameter to retrieve resources that claim to conform to a profile.

Because FHIR servers support variations of the core FHIR specification, it is important that FHIR clients be able to determine exactly what is acceptable and possible with the FHIR server. To meet this need, every FHIR server must provide a Capability Statement that identifies the APIs, FHIR operations, search parameters, and resources that it supports. FHIR clients can retrieve this Capability Statement with a call to `GET [EndpointBaseURL]/metadata`.

6.1 Working with FHIR Packages

Within InterSystems products, a FHIR package is a collection of conformance resources, like StructureDefinitions and SearchParameters. In this way, packages contain the profiles for a healthcare environment. A package can contain the standard conformance resources for a version of FHIR or it can extend or constrain a version of FHIR for a specific purpose. These packages are distributed and imported as NPM packages of JSON files. The contents of a package can vary widely; it can be used to distribute a national Implementation Guide (for example, US Core) or be limited to a Patient profile that is unique to a health network. In some cases, you might need to configure an endpoint using a standard, published package that can be downloaded from a distribution site. In other cases, you might develop your own package that contains custom profiles and search parameters.

If you need to work with packages programmatically, see Package APIs.

6.1.1 Importing Packages

Before you can configure an endpoint to support an Implementation Guide or custom package, you need to import the published or custom package using the Management Portal. Some standard packages are available by default (for example, US Core), and do not need to be imported before applying them to an endpoint.

To import a package:

1. Make sure the JSON files of the package are on your local machine. If you are importing a published package, download it from the distribution site to your local machine. For additional requirements for a custom package, see Creating Custom Packages.
2. In the Management Portal, navigate to **Home** > **Health** > *MyFHIRNamespace* > **FHIR Configuration**.
3. Select the **Package Configuration** card.
4. Make sure that the dependencies of the new package have already been imported. You can review which packages have been imported by looking at the left hand navigation bar of the **Package Configuration** page.
5. Select **Import Package**.
6. Select the directory that contains the package's JSON files. Do not select the individual files.
7. Select **Import**.

The package of profiles and other artifacts that were contained in the FHIR package are now available for an endpoint.

6.1.2 Uninstalling Packages

You can remove a package from the FHIR server's namespace if it is not a dependency of another package and has not been applied to an existing endpoint. Uninstalling a package does not delete the local JSON files that were used to import the package. To uninstall a package:

1. In the Management Portal, navigate to **Home** > **Health** > *MyFHIRNamespace* > **FHIR Configuration**.
2. Select the **Package Configuration** card.
3. Select the package from the left hand navigation bar.
4. Select **Uninstall Package**. You cannot uninstall a package that has been applied to an endpoint. In addition, you cannot uninstall a package that is a dependency of another package.

6.1.3 Creating Custom Packages

You can use a custom package to configure your FHIR endpoint to support a custom profile or search parameter. For example, to add a custom search parameter, define a SearchParameter resource in a JSON file on your local machine. Then, create a file called package.json in the same directory. At a minimum, this file must include the name, version, and dependencies of the package. For example, the package.json file might look like:

```
{
  "name":"myorg.implementation.r5",
  "version":"0.0.1",
  "dependencies": {
    "hl7.fhir.r5.core":"5.0.0"
  }
}
```

Once you have JSON files with conformance resource definitions and a package.json file in a directory, you are ready to import the new package.

6.1.4 Applying Packages to an Endpoint

When you create a new FHIR endpoint, you can select a package that the endpoint will support. Only those packages that have been imported are available when creating the endpoint; InterSystems products come with a few published packages already imported.

You can also apply a new package to an *existing* endpoint. To add a package to an existing endpoint:

1. In the Management Portal, navigate to **Home** > **Health** > *[MyFHIRNamespace]* > **FHIR Configuration**.
2. Select the **Server Configuration** card.
3. Select the endpoint from the list.
4. Select **Edit**.
5. Use the **Additional Packages** drop-down list to select the package. If you do not see the package in the list, make sure you have imported the package.
6. Select **Update**.

Important: If you are applying a package to an existing endpoint, and the package has new search parameters, the new parameters cannot be used to retrieve pre-existing resources until you re-index the endpoint. For details, see Re-indexing an Endpoint

6.1.5 Re-indexing an Endpoint

When you add new search parameters to an *existing* endpoint using a published or custom package, FHIR clients can use the new parameter to retrieve resources added to the repository *after* you applied the package. However, resources that existed *before* you added the new search parameter will not be returned until you re-index the endpoint. Until the endpoint is re-indexed, a FHIR client that uses the new search parameter receives an OperationOutcome stating that the search results might be incomplete.

Once you have applied a package that contains new search parameters, an option to re-index the endpoint appears next to the endpoint's URL on the Server Configuration page (**Health** > **FHIR Configuration** > **Server Configuration**). If the repository has a lot of pre-existing resources, it can take a significant amount of time to re-index the endpoint.

6.1.6 Package APIs

If your implementation needs to work with packages directly without using the user interface, you can leverage the following API methods.

Importing Packages

The InterSystems FHIR server uses packages to determine which FHIR profiles and other assets it supports. While InterSystems products come with pre-loaded packages that correspond to base FHIR versions and popular Implementation Guides, you can also import new packages by specifying a directory that contains the JSON files that define conformance resources like StructureDefinition and ValueSet. For more information about FHIR packages, see Working with Packages.

The API for importing a new package so it can be added to an endpoint is **HS.FHIRMeta.Load.NpmLoader.importPackages()**. For example, the following code would import a custom package:

ObjectScript

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

Listing Available Packages

To obtain a list of the packages that have been imported into the namespace, use the **HS.FHIRMeta.Storage.Package.GetAllPackages()** method. For example, the following code displays the identifiers of the available packages:

ObjectScript

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

Specifying a Package when Creating an Endpoint

The `pPackageList` parameter of the `InstallInstance()` method allows you to specify the packages you want applied to a new endpoint. For more details, see installing a FHIR server programmatically.

Adding Packages to an Existing Endpoint

If you need to add a package to an existing endpoint, you can leverage the **HS.FHIRServer.Installer.AddPackagesToInstance()** method.

Uninstalling a Package

You can use **HS.FHIRMeta.Load.NpmLoader.UninstallPackage()** to remove a package from the FHIR server's namespace if it is not a dependency of another package and has not been applied to an existing endpoint. Uninstalling a package does not delete the local JSON files that were used to import the package. You can determine the id of the package you want to uninstall by Listing Available Packages. As an example, the call to uninstall a package might look like:

ObjectScript

```
 do ##class(HS.FHIRMeta.Load.NpmLoader).UninstallPackage("myorg.r5@1.0.0")
```

6.2 Custom Search Parameters

Adding a custom search parameter to an endpoint consists of creating a custom package with the SearchParameter resource and applying it to the endpoint. To complete the process:

- Use a text editor or third-party tool to create a SearchParameter JSON file.
- Put the JSON file and a package.json file into a file directory so it can be imported as a custom package. For details, see Creating Custom Packages.
- Import the package.
- Apply the package to your endpoint.
- If you applied the package to an existing endpoint, you might need to re-index the endpoint.

6.3 Extensions

The FHIR server accepts a resource with extensions as long as it is well-formed according to the syntax for extensions defined by the base FHIR specification. In adherence to the FHIR specification, the FHIR server does not automatically verify whether those extensions are valid or conform to the profile specified in the resource’s `meta` field.

For information about adding custom search parameters for an extension, see Custom Search Parameters.

7

FHIR Interoperability Adapter

Not all solutions require a FHIR server that routes requests to an internal repository. For example, an implementation may need to receive an HL7® FHIR® request and forward it to an external FHIR server without ever storing its payload in an InterSystems product. In cases where you need to process a FHIR request without leveraging the internal repository of a FHIR server, you can use the FHIR Interoperability Adapter to receive requests into an interoperability production. For Health Connect implementations that are not licensed to install a FHIR server with a repository, incoming FHIR requests are processed by installing a FHIR interoperability Adapter.

Installing the FHIR Interoperability Adapter creates a new interoperability REST endpoint that uses special business hosts to process FHIR requests in a production. Note that this interoperability REST endpoint does not appear with the FHIR server endpoints in the Management Portal.

7.1 Installing an Adapter

To install a FHIR Interoperability Adapter:

1. Create a namespace with an interoperability production.
2. Open the InterSystems Terminal and change to the namespace that you just created. For example, enter:

   ObjectScript

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

3. Run the following command, specifying the URL of the interoperability REST endpoint:

   ObjectScript

   ```
   set status = ##class(HS.FHIRServer.Installer).InteropAdapterConfig("/MyEndpoint/r5")
   ```

   The URL of the Adapter's endpoint must start with a slash (`/`).

4. To ensure the command executed successfully, enter:

   ObjectScript

   ```
   write status
   ```

   The response should be `1`.

5. If this was the first FHIR Interoperability Adapter created for the namespace, navigate to **Interoperability** > **List** > **Productions**, open your production, and do one of the following:
   - If you see an **Update** button, select it.
   - If the **Update** button does not appear and you are ready to test your production, select **Start** to start the production.

7.2 Adapter Components

Installing the FHIR Interoperability Adapter creates:

- A web application with the specified URL
- A new business service in the interoperability production called `InteropService`. If you install multiple Adapters, they all use the same `InteropService` business service. If you want an Adapter to use a different business service, see Using a Custom Business Service.
- A new business operation in the interoperability production called `InteropOperation`. This is a placeholder business operation that can be extended or replaced according to your use case. Until you modify `InteropOperation` to implement custom functionality, it returns an `501 Unimplemented` error when a FHIR request is received by the new interoperability REST endpoint.

  Note: When using a production, you must explicitly set the *ContentType* property of an HS.FHIRServer.Interop.Response to create the HTTP response Content-Type header. Setting the ResponseFormatCode in the HS.FHIRServer.API.Data.Response is not sufficient.

For details about other production components that can be used in conjunction with the FHIR Interoperability Adapter, see Interoperability Productions.

7.3 Using a Custom Business Service

By default, if you install multiple FHIR Interoperability Adapters, they all share the same business service, `InteropService`. If you want requests received by the adapter endpoints to be routed to different business services, you need to create a subclass of the REST handler and specify it as the Dispatch Class of the Adapter's CSP application. This process of using a custom business service consists of the following steps:

1. Using an IDE, create a subclass of HS.FHIRServer.HC.FHIRInteropAdapter.
2. Use your subclass' `ServiceConfigName` parameter to specify the name of the custom business service that will receive the FHIR requests.
3. In the Management Portal, navigate to **System Administration** > **Security** > **Applications** > **Web Applications**.
4. Select the URL of your FHIR Interoperability Adapter.
5. Using the **Enable** field on the **General** tab, specify the name of your subclass in the **Dispatch** text box.
6. Select **Save**.

7.4 Security

Security of the interoperability REST endpoint depends on the security settings of the web application created for the FHIR Interoperability Adapter. For example, you can configure the web application to require that the user making the FHIR request have privileges to an InterSystems resource. The security settings for the web application are available by navigating to **System Administration** > **Security** > **Applications** > **Web Applications**. The web application is identified by the URL of the interoperability REST endpoint. For details about the security settings, see Edit a Web Application.

The FHIR Interoperability Adapter does not provide extensive OAuth 2.0 support. If a request to the adapter contains an OAuth 2.0 token, it is examined with basic tests that determine if the token is in the Authorization header, is non-blank, and is on a secure connection. Unlike a FHIR server, it does not examine the token's contents like scope and patient context value. If the token passes the adapter's basic tests, it is added to the request message in the AdditionalInfo property of HS.FHIRServer.API.Data.Request.

8

Interoperability Productions for FHIR

InterSystems healthcare products provide built-in business hosts that you can use to create an interoperability production that accepts and/or sends out HL7® FHIR® requests. Business processes that transform SDA to FHIR and FHIR to SDA are also available.

To explore some of the FHIR implementations that are possible using an interoperability production, see Use Cases.

Note: The InterSystems FHIR server does not require an interoperability production; by default, FHIR requests received by an endpoint's REST handler are sent directly to the FHIR server's Service. FHIR servers that do not use an interoperability production can be significantly faster.

8.1 Accepting FHIR Requests

FHIR implementations can accept FHIR requests into an interoperability production is two ways:

- For FHIR servers (implementations that leverage the internal architecture and storage of an InterSystems product), the FHIR request can be sent through HS.FHIRServer.Interop.Service on its way to the repository. For details, see Accepting FHIR Server Requests.
- For implementations that do not use a FHIR server, you can use the FHIR Interoperability Adapter to accept FHIR requests into an interoperability production. For details, see FHIR Interoperability Adapter.

8.1.1 Security for Incoming Requests

A FHIR request can enter an interoperability production using a FHIR Interoperability Adapter or a FHIR Server that uses a production. Security is handled differently depending on which feature you are using to receive the request. If your production is using the FHIR interoperability, see Adapter Security. If your production is using a FHIR server, see Server Security

8.1.2 Accepting FHIR Server Requests

The built-in business service HS.FHIRServer.Interop.Service is designed to receive FHIR requests that have been sent to a FHIR server endpoint. Once configured, the endpoint's REST handler routes the request to HS.FHIRServer.Interop.Service rather than the FHIR server's Service.

Setting up an endpoint to route FHIR server requests through an interoperability production is a two-step process:

1. Create an interoperability production and add the HS.FHIRServer.Interop.Service business service.

2. Configure an endpoint's **Service Config Name** field so it specifies the name of the business service that has been added to the interoperability production.

These steps can be taken in any order as long as, when the setup is complete, the name of the business service in the endpoint's configuration matches the name in the interoperability production.

Note: This two-step process assumes you are using a FHIR server; if your implementation does not leverage the internal architecture and repository of a FHIR server, use the FHIR Interoperability Adapter to accept FHIR requests.

8.1.2.1 Creating the Interoperability Production

When the Foundation namespace for the FHIR server endpoint is created, the installation process also creates an interoperability production that should be used as the FHIR production. You need to modify the production to add the required business service that the endpoint uses to route requests through the production.

Interoperability productions that receive FHIR requests from the REST handler must include the HS.FHIRServer.Interop.Service business service. You can give the business service a custom name, but make sure that name matches the one specified for the endpoint's **Service Config Name** option.

8.1.2.2 Configuring the Endpoint

After installing a FHIR server endpoint, the endpoint can be configured to use an interoperability production at anytime, including before the production has been created. Specifying the name of the business service while configuring the endpoint does not automatically create the business service in your production.

To configure an existing endpoint so FHIR requests are routed through a production:

1. Navigate to **Health** > *MyNamespace* > **FHIR Configuration**. Make sure you are in the FHIR server's namespace.
2. Select the **Server Configuration** card.
3. Select the endpoint.
4. Select **Edit**.
5. In the **Service Config Name** field of the **Interoperability** section, specify the name of the business service of the production through which FHIR requests will be routed. For example, if the business service does not have a custom name, specify `HS.FHIRServer.Interop.Service`
6. Select **Update**.

8.2 Sending FHIR Requests

Within an interoperability production, business operations are responsible for making sure a FHIR request is sent to a FHIR endpoint. This request can originate from a variety of sources, for example, from an external FHIR client accessing an InterSystems endpoint or from a business process that transforms HL7 messages into FHIR requests. Regardless of its origin, there are two business operations available to send requests:

Business Operation Class	Description
HS.FHIRServer.Interop.Operation	Sends a FHIR request to the internal Service of an InterSystems FHIR server in the local namespace. Except in rare cases where a custom architecture has been implemented, Health Connect users cannot use this business operation unless they have a license to use the Resource Repository. This business operation identifies the correct InterSystems FHIR server based on the URL of its endpoint, which is included in the SessionApplication property of the request message. If the message originated from a request sent to the FHIR server's endpoint through the REST Handler, the endpoint's URL is already part of the message. If the message was sent from the business process that transforms SDA to FHIR (HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process), the server is identified by the **FHIREndpoint** setting of the business process.
HS.FHIRServer.Interop.HTTPOperation	Sends a FHIR request to an internal or external FHIR endpoint over HTTP. If you are using a built-in business host to send the request to this business operation, use that business host's **TargetConfigName** setting. The default HTTP address of the FHIR endpoint is specified with the business operation's **ServiceName** setting, which refers to an entry in the Service Registry. This default is overridden if a request includes an AdditionalInfo item named `ServiceName`, which specifies a Service Registry entry pointing to the alternate endpoint.

If a *built-in* business host (such as HS.FHIRServer.Interop.Service) sends a request message (HS.FHIRServer.Interop.Request) to the HS.FHIRServer.Interop.HTTPOperation business operation, the request is sent over HTTP without custom code. However, if a FHIR payload is *formulated within a custom business host* that needs to put the payload into a FHIR request, you should instantiate an interoperability FHIR client to send the message. Similarly, if your custom business host needs to retrieve FHIR data from an endpoint, your production should use the FHIR client.

8.3 Interoperability FHIR Client

InterSystems technology provides a FHIR client object that simplifies the process of formulating a FHIR request from within a custom business host and sending it to a FHIR endpoint over HTTP. The business operation, HS.FHIRServer.Interop.HTTPOperation, that is used by the FHIR client to send the request over HTTP must be added to the interoperability production. Once the production is configured, your custom business host can use the FHIR client by instantiating HS.FHIRServer.RestClient.Interop, then calling the methods that correspond to FHIR interactions and operations.

Not all productions that send out FHIR requests over HTTP need to instantiate the interoperability FHIR client. For example, if SDA is being transformed into FHIR using HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process, the FHIR forwarded from this business process to HS.FHIRServer.Interop.HTTPOperation is sent out via HTTP without the FHIR client. However, when a FHIR payload is formulated by a custom business host within a production, the recommended method of sending it to a FHIR endpoint over HTTP is to instantiate the FHIR client.

When instantiating the FHIR client within the context of a custom business host, the call to the **CreateInstance( )** method must contain the following parameters:

- `pServiceName` — Name of an entry in the Service Registry that points to a FHIR endpoint. This value overrides the **ServiceName** setting of the HS.FHIRServer.Interop.HTTPOperation business operation.
- `pTargetConfigName` — Name of the HS.FHIRServer.Interop.HTTPOperation business operation.
- `pHostObj` — Object instance of the business host that is instantiating the FHIR client. You can use `$this` to specify the current business host object that is instantiating the FHIR client.

For example, to instantiate a FHIR client within a business host with only the required arguments, enter:

ObjectScript

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

The **CreateInstance( )** method also accepts optional arguments that specify the value of the FHIR `prefer` header and send an OAuth token with the request.

Once the FHIR client has been instantiated, you can use it to send requests and perform operations. For details on using the FHIR client's methods to perform these actions, see Interactions and Operations.

Note: The interoperability FHIR client class (HS.FHIRServer.RestClient.Interop) can also be used by a standalone ObjectScript application that needs to send a FHIR request through an interoperability production. In this case, the HS.HC.Util.BusinessService business service must be added to the production along with HS.FHIRServer.Interop.HTTPOperation. Instantiating the client is similar, but for standalone applications, the call to `CreateInstance()` should not include an argument for the `pHostObj` parameter.

8.4 Transformations

You can add built-in business processes to your production to invoke SDA-FHIR transformations. For example, a production could consume HL7 messages, use a business process to convert the HL7 to SDA, and then use the built-in SDA-FHIR business process to convert the SDA to FHIR. The production running these transformations must be in a Foundation namespace.

For more information about SDA-FHIR transformations using the built-in business processes, see Transformation Business Processes.

8.5 Use Cases

The following use cases provide examples of how to use the built-in interoperability components to work with FHIR resources.

- Proxy Server

- Transforming HL7 into FHIR
- Production-Based InterSystems FHIR Server

Proxy Server

InterSystems healthcare products can be used as a proxy server that accepts FHIR requests from an external FHIR client and forwards them to an external FHIR endpoint, then routes responses from the FHIR endpoint back to the external client. In this scenario, the FHIR client might be unaware that the InterSystems product is not the server that is accepting and producing FHIR, and the request or response can be manipulated within the production as needed.

You could implement a simple proxy server by:

- Installing the FHIR Interoperability Adapter.
- Adding HS.FHIRServer.Interop.HTTPOperation to the production and editing the **ServiceName** setting to specify the external FHIR endpoint.
- Editing the **TargetConfigName** of `InteropService` to point to HS.FHIRServer.Interop.HTTPOperation.

Of course, there are variations on the proxy server use case. For example, you could also add multiple HS.FHIRServer.Interop.HTTPOperation business operations and use a business process to determine which external FHIR endpoint should be the target of the proxy server.

If you wanted to store FHIR data in an internal InterSystems repository along with sending it out to an external FHIR endpoint, you would need to start with a FHIR Server rather than the FHIR Interoperability Adapter. In this case, you could send requests to external endpoints using HS.FHIRServer.Interop.HTTPOperation, but also use HS.FHIRServer.Interop.Operation to store the data internally.

Transforming HL7 into FHIR

InterSystems healthcare products simplify the process of extracting clinical data from incoming HL7 messages and transforming that data into FHIR resources. Once transformed into FHIR, the clinical data can be forwarded to external FHIR endpoints or stored in an internal FHIR repository that can be queried by FHIR clients. A basic interoperability production that transforms HL7 messages into FHIR resources would include:

- Adding a built-in business service that accepts HL7 messages into the production, for example, EnsLib.HL7.Service.HTTPService.
- Using a business host to transform the HL7 into SDA (the InterSystems intermediary data format). The following code added to a business process is enough to transform the HL7 into SDA:

  ```
  do ##class(HS.Gateway.HL7.HL7ToSDA3).GetSDA(request,.con)
  set streamContainer = ##class(ENS.StreamContainer).%New()
  set streamContainer.Stream = con
  set sc = ..SendRequestSync("SDAToFHIRProcess",streamContainer,.pResponse)
  ```

  For more information about this transformation method, see Data Transformations in InterSystems Healthcare Products.
- Adding the HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process business process to the production; this business process transforms SDA into FHIR.
- Modifying the **TargetConfigName** setting of the business host that contains the HL7–to-SDA transformation method to specify the name of HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process.

Once the HL7 data has been transformed into FHIR, it can be sent to an external FHIR endpoint or, in the case of FHIR server, stored in an internal Resource Repository. You control where the FHIR data is forwarded by adding a business operation that performs a specific function. For details about these business operations, see Sending

FHIR Requests. If you are using the business operation that forwards requests to the internal storage of the FHIR server, use the **FHIREndpoint** setting of HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process to specify the InterSystems FHIR server's endpoint.

For a hands-on example of integrating HL7 message with a FHIR server, see FHIR R4 Integration QuickStart.

Production-Based InterSystems Server

By default, requests to an InterSystems FHIR server do not go through an interoperability production, however you may want to use a production in some cases. For example, you may want to use a production during development to leverage message tracing and other advantages of productions, then make a small modification to send requests directly to the server's Service when it goes live. In an alternate use case, you might want to manipulate the FHIR requests using a business process before they reach the InterSystems FHIR server.

In its simplest form, a production-based FHIR server consists of configuring the production as described in Accepting FHIR Server Requests, then adding HS.FHIRServer.Interop.Operation as described in Sending FHIR Requests. Once both business hosts are added to the production, modify the **TargetConfigName** setting of HS.FHIRServer.Interop.Service to specify the name of the HS.FHIRServer.Interop.Operation business operation.

If your aim is to use a production during development, then switch to a FHIR server that sends a request directly to the Service, simply reconfigure the Server's endpoint by removing the value in the **Service Config Name** field when the server goes live.

9

SDA-FHIR Transformations

InterSystems provides transformations that convert SDA objects into HL7® FHIR® resources (and vice-versa) using the data transformation language (DTL). SDA is an intermediary clinical format that makes it easier to go from one standard to another. For example, rather than transform HL7v2 to FHIR directly, a system can convert HL7v2 to SDA and then SDA to FHIR. For more information about SDA, see SDA: InterSystems Clinical Data Format.

The bi-directional SDA-FHIR transformations can provide useful functionality in many different use cases, including:

- Taking content from an SDA-aware system and providing it to a FHIR system.
- Taking content from an SDA-aware system and storing it in a FHIR repository.
- Taking content from multiple SDA-aware systems and normalizing it for use or storage in a FHIR system.
- Taking content from a FHIR system and providing it to an SDA-aware system.

Note: Transformations between SDA and FHIR are available for FHIR R4 and earlier, but not supported for FHIR R5.

You have two options for invoking the DTL transformations that convert SDA objects into FHIR resources and vice-versa. You can invoke the DTL transformations by adding a built-in business process to an interoperability production, or you can call the transformation APIs directly, for example, from a custom business process.

9.1 Transformation Business Processes

You can use built-in business processes to invoke SDA-FHIR transformations in an SDA to FHIR Production or FHIR to SDA Production. For example, a production could consume HL7 messages, use a business process to convert the HL7 to SDA, and then use the built-in SDA-FHIR business process to convert the SDA to FHIR.

For more information about the underlying transformation code used by the built-in business processes, see Transformation APIs. These APIs can be called directly from a custom business process.

9.1.1 SDA to FHIR Productions

A built-in business process, HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process, can be added to a production to transform SDA objects and containers to FHIR bundles. This production must exist in a Foundation namespace.

Once added to the production, the business process:

- Accepts an SDA container as input and loops through each contained object.
- Converts the SDA container to FHIR content, in the form of a FHIR `Bundle` resource.

- Forwards the FHIR content to the business host specified by the TargetConfigName setting.
- Receives a response from the business host.
- Returns a response (based on what it received) to the business host that originally called it.

The business process in the SDA to FHIR production calls a method of the HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR class to perform the transformation. For details about how this class handles the transformation, see Transformation Details.

9.1.1.1 Adding the Business Process

To begin, open your production in the Production Configuration window of the Management Portal and add the HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process business process. Once added, you can modify the business process settings that impact the transformation. For an introduction to adding business processes to an interoperability production, see InterSystems IRIS Basics: Connecting Systems Using Interoperability Productions.

9.1.1.2 Business Process Settings

Settings of HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process that influence SDA to FHIR conversions include:

- TargetConfigName — Specifies the business host to which HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process sends its output. This setting is located in the **Basic Settings** section of the **Settings** tab in the Production Configuration window.
- TransmissionMode — Specifies how the business process transmits the FHIR bundle for further processing:
  - transaction — The business process sends the bundle of resources in a single interaction and the processing succeeds or fails for the whole of the bundle; if processing any single resource fails, processing for the other resources (and the entire bundle) stops. This is the default.
  - individual — The business process sends each resource from the bundle separately as its own interaction.

  This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- FullTransactionResponse — If selected, the FHIR request message that this process sends is created with a "PREFER" header value set to "return=representation". Per the FHIR spec, this header indicates to a FHIR server that every created or updated resource should be returned in its entirety as it is saved (i.e., with any modifications applied by the server). Whether the server actually does this depends on the server. In general, this setting should be left unchecked except during debugging or if the FHIR client has a specific need to receive back the created/updated resources, as requesting this information is likely to increase response time from the FHIR server. This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- FHIRFormat — Specifies whether the content is in XML or JSON format. This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- FormatFHIROutput — Specifies whether or not content is formatted for readability. If selected, this setting has a performance impact, and as such should be enabled only during development and testing. This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- CallbackClass — Deprecated.
- ValidResourceRequired — Deprecated.
- OutputToQuickStream — If selected, the FHIR payload sent by this business process is placed in an HS.SDA3.QuickStream object, and the id of the QuickStream object is placed in the QuickStreamId property of the request message. If left unselected, the FHIR output from the transformation is placed in the Payload property of the request message. This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- TransformClass — Specifies name of the class that performs the transformation. If you subclass HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR to customize the transformation behavior, you need to specify the name of that subclass.

- FHIRMetadataSet — Specifies the version of the outgoing FHIR based on a package. All available packages appear in the drop-down list.
- FHIREndpoint — Specifies the endpoint of a FHIR server. This setting is required if your business process is sending the outgoing FHIR to an `HS.FHIRServer.Interop.Operation` business operation on its way to the FHIR server's Service.

9.1.1.3 Assigning a Patient ID

You can use the AdditionalInfo property of the SDA message sent to HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process to assign an ID to the Patient resource that is created by the SDA-FHIR transformation. When the SDA message contains an AdditionalInfo item named `PatientResourceId`, the transformation takes the value of `PatientResourceId` and assigns it to the Id field of the generated Patient resource.

Note: The underlying class, HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR, used by the transformation business process contains a method that can be overridden to assign Ids to resources, including patient resources. For more information, see Customizing Transformation API Classes.

9.1.1.4 Messages

The request message to HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process is either Ens.Container or HS.Message.XMLMessage.

There is no response message from HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process. It returns a success or failure status instead.

9.1.2 FHIR to SDA Productions

A built-in business process, HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process, can be added to a production to transform FHIR resources and bundles into SDA objects and containers. This production must exist in a Foundation namespace.

Once added to the production, the business process:

- Accepts a FHIR resource or bundle as input.
- Converts the FHIR content to an SDA container.
- Forwards the container to the business host specified by the TargetConfigName setting.
- Receives the response from the business host.
- Returns a FHIR response (based on what it received) to the business host that originally called it.

The business process in the SDA to FHIR production calls a method of the HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 class to perform the transformation. For details about how this class handles the transformation, see Transformation Details.

9.1.2.1 Adding the Business Process

To begin, open your production in the Production Configuration window of the Management Portal and add the HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process business process. Once added, you can modify the business process settings that impact the transformation. For an introduction to adding business processes to an interoperability production, see First Look: Connecting Systems Using Interoperability Productions.

9.1.2.2 Business Process Settings

Settings of HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process that influence FHIR to SDA conversions include:

- TargetConfigName — Specifies the business host where the XMLMessage that includes the SDA3 stream is sent after it is transformed from FHIR by the DTL transformation. This setting is located in the **Basic Settings** section of the **Settings** tab in the Production Configuration window.
- CallbackClass — Deprecated.
- OutputToQuickStream — By default, the output of HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process is an HS.Message.XMLMessage object that contains the SDA3 stream produced by the DTL transformation. If this setting is checked, the SDA3 stream is placed in a separate HS.SDA3.QuickStream object, and the QuickStreamID of the QuickStream object is placed in the AdditionalInfoItem property of the XMLMessage. If this setting is not selected, the SDA3 stream is placed in the ContentStream property of the XMLMessage. This setting is located in the **Additional Settings** section of the **Settings** tab in the Production Configuration window.
- TransformClass — Specifies name of the class that performs the transformation. If you subclass HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 to customize the transformation behavior, you need to specify the name of that subclass.
- FHIRMetadataSet — Specifies the version of the incoming FHIR based on a package. All available packages appear in the drop-down list.

9.2 Transformation APIs

Your application has access to both SDA to FHIR APIs and FHIR to SDA APIs.

9.2.1 SDA to FHIR APIs

The APIs that your code uses to transform SDA to FHIR are found in HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR. Your application can call the `TransformStream()` or `TransformObject()` method, depending on whether the SDA is in a %Stream.Object or an SDA object.

Both of these methods return a transformation object (HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR) that contains the FHIR output in its bundle property, which is of type `%DynamicObject`. This Bundle contains all of the resources generated by the transformation with all references resolved.

Your code could serialize this bundle property from a dynamic object to JSON or XML with the following code. It assumes that `SDA3ToFHIRObject` is the transformation object returned by one of the transformation methods.

ObjectScript

```
Set stream = ##class(%Stream.TmpCharacter).%New()
Set metadataSetKey = "R5"
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 Using the `TransformStream` Method

The `TransformStream()` method of HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR takes in SDA as a `%Stream` and transforms it into a FHIR bundle. Its signature is:

Class Member

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

Parameters:

- `stream` — A `%Stream` representation of an SDA object or Container.
- `SDAClassname` — The classname for the object contained in the stream (for example, HS.SDA3.Container).
- `fhirVersion` — The version of FHIR produced by the transformation. For example, `STU3` or `R5`.
- `patientId` — If this optional parameter is specified, the `Id` field of the generated Patient resource will have the specified value.
- `encounterId` — If this optional parameter is specified, the `Id` field of the generated Encounter resource will have the specified value. This parameter is ignored if the `stream` parameter is a SDA Container because a Container can have multiple encounters, making it impossible to determine which FHIR Encounter should be given the specified resource id.

9.2.1.2 Using the `TransformObject` Method

The `TransformObject()` method of HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR takes in SDA as a container or object class and transforms it into a FHIR bundle. Its signature is:

Class Member

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

Parameters:

- `source` — The SDA container or SDA object class that will be converted into FHIR.
- `fhirVersion` — The version of FHIR produced by the transformation. For example, `STU3` or `R4`.
- `patientId` — If this optional parameter is specified, the `Id` field of the generated Patient resource will have the specified value.
- `encounterId` — If this optional parameter is specified, the `Id` field of the generated Encounter resource will have the specified value. This parameter is ignored if the `stream` parameter is an SDA Container because a Container can have multiple encounters, making it impossible to determine which FHIR Encounter should be given the specified id.

9.2.1.3 Transformation Details

The following describes the default behavior of SDA to FHIR transformations. For an introduction to methods that can be overridden to customize transformation behavior, see SDA to FHIR Overridable Methods.

- The incoming stream or object is broken down into individual streamlets, which are in turn transformed into STU3 resources.
- By default, UUIDs are generated and assigned to the `fullUrl` field of the Bundle resource. In this case, the resource itself does not have an `Id`. If you would rather provide a resource id, override the `GetId` method. In this case, the value for `fullUrl` is *`baseURL/resourceType/id`* and the resource references are *`resourceType/id`*.
- The methods do not modify incoming URLs at all by default. This behavior can be overridden with the `GetBaseURL()` method: for example, if you are posting to a specific repository, you can provide the URL prefix for the repository.

- Resources will contain references to other resources regardless of the mechanism used to assign IDs.
- Patient and Encounter references will be added to all available resources using the Patient and Encounter streamlets. Encounter references can be made successfully only if the `EncounterNumber` fields in the SDA streamlets are used. If they are empty, no references will be generated.
- In the case of shared resources such as Organization, Practitioner, or Medication, a hash of the first 32 kilobytes of each resource is added to a hash table. Each subsequent shared resource is checked for duplication by searching the hash table for a direct match. If a match is found, the resource will be marked as a duplicate. This behavior can be changed by overriding the `IsDuplicate()` method.
- Each resource is validated before being added to the Bundle. If a resource fails validation, an error is thrown and processing stops, which means the Bundle is not returned. This default behavior can be changed by overriding the `HandleInvalidResource()` method.
- When one or more SDA properties do not map to a FHIR resource field in the target schema, the transformation maps the SDA data to a FHIR extension. For more information, see FHIR Extensions.
- For details about how a specific SDA object or property maps to the target FHIR resource or field, see Understanding SDA-FHIR Mappings.

9.2.2 FHIR to SDA APIs

The APIs that your code uses to transform FHIR to SDA are found in HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3. This class contains multiple APIs that can be used to transform FHIR to SDA, depending on your use case.

In most cases, if your application needs to transform a single FHIR resource or bundle, it should call the class method `TransformStream()` or `TransformObject()`, depending on whether the FHIR is in a %Stream.Object or a dynamic object. However, in cases where you are transforming multiple FHIR bundles or resources in succession, it might be more efficient to instantiate the transformation class once and then call the `Transform()` method multiple times.

All of these transformation methods return a transformation object (HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3) that contains the SDA output in its container property, which is of type HS.SDA3.Container. The transformation object's object property contains the last SDA container or object that was generated by the transformation. If the last input was a bundle, the object property is an SDA container; if the last input was an individual resource, `object` is an SDA object.

9.2.2.1 Using the `TransformStream` Method

The `TransformStream()` method of HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 takes in a FHIR resource or bundle represented as a `%Stream` and transforms it into an SDA Container. Resource references are honored only if a FHIR bundle is passed to the method. Its signature is:

Class Member

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

Parameters:

- `stream` — A `%Stream` representation of the FHIR resource or bundle.
- `fhirVersion` — The version of the FHIR resource or bundle being transformed. For example, “STU3” or “R4”.
- `fhirFormat` — Specifies the format of the FHIR resource or bundle. Acceptable values are “JSON” and “XML”.

9.2.2.2 Using the `TransformObject` Method

The `TransformObject()` method of HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 takes in a FHIR resource or bundle as a dynamic object and transforms it into an SDA Container. Resource references are honored only if a bundle is passed to the method. Its signature is:

Class Member

```
ClassMethod TransformObject(source As %DynamicObject,
                            fhirVersion As %String) {}
```

Parameters:

- `source` — The FHIR resource or bundle represented as a dynamic object.
- `fhirVersion` — The version of the FHIR resource or bundle being transformed. For example, “STU3” or “R4”.

9.2.2.3 Using the `Transform` Method

The `Transform()` method of HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 takes in a FHIR bundle as a dynamic object and transforms it into an SDA Container. Resource references are honored only if a bundle is passed to the method.

`Transform()` is the method called by the class methods that transform FHIR into SDA. You might want to call it directly if you are transforming multiple FHIR resources in succession so you do not need to instantiate a HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 object every time. For example, the following code would transform a Patient resource, Encounter resource, and Observation resource using the same transformation object:

ObjectScript

```
 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)
```

The signature of the `Transform()` method is:

Class Member

```
ClassMethod Transform(source As %DynamicObject) {}
```

Parameters:

- `source` — The FHIR resource or bundle represented as a dynamic object.

9.2.2.4 Transformation Details

The following is an overview of the default behavior of FHIR to SDA transformations. For an introduction to methods that can be overridden to customize transformation behavior, see FHIR to SDA Overridable Methods.

- An incoming FHIR Bundle is broken down into individual resources, and those resources transformed into SDA3 streamlets.
- If a resource referenced by another resource within the incoming FHIR bundle is not present in the bundle, the transformation of the bundle continues. To change this behavior, override the `HandleMissingResource()` method.
- When a transformation is attempting to convert a reference to an object, no object will be created in the SDA streamlet if:
  - A subtransformation exists but the referenced resource has no values for any of the elements with mappings.

  - There is no subtransformation from the referenced resource type to the datatype in the SDA3 object.

- The `EncounterNumber` field on an Encounter streamlet will be populated starting at 1 and incremented for each encounter that is processed. Any subsequent resources that reference that Encounter resource, when transformed to SDA3, will perform a lookup based on the resource ID and will find the encounter number it should use. The assignment of encounter numbers can be overridden with `GetIdentifier()` method. It can be useful to access the contents of the resource being converted in order to determine what EncounterNumber should be returned. The instance property %currentReference contains a FHIR reference object that can be passed into the instance method `GetResourceFromReference()` in order to obtain the resource as a dynamic object.
- Similar to encounter numbers, `ExternalID` values for HealthConcern and Goal resources are populated starting at 1 by default. This behavior can be overridden with the `GetIdentifier()` method.
- The value of the SDA `Container:SendingFacility` property is set as follows: if the Patient's `managingOrganization` field contains a reference to an Organization, and that Organization is in the Bundle, it is used. Otherwise, the patient identifiers are searched for an MRN with an assigning authority, and that assigning authority is used. If neither of these items is found, the string `FHIR` is used. This behavior can be overridden in the `GetSendingFacility()` method.
- SDA3 extensions are not used. If a field does not exist in SDA3, the content will be dropped.
- If a Bundle comes in without a Patient resource, an error will be thrown. Other than that, no validation will be performed on the container. It will simply be returned as is.
- To view information about containment relationships, refer to the FHIR Annotations (**Health** > **FHIR Annotations**) in the Management Portal for the Bundle resource.
- For details about how a specific FHIR resource or field maps to an SDA object or property, see Understanding SDA-FHIR Mappings.

9.3 Understanding SDA-FHIR Mappings

Whether you use the transformation API or a built-in business process to perform an SDA-FHIR transformation, you can use the **FHIR Annotations** tool to understand exactly how the SDA or FHIR data was transformed into the target format. The tool gives you an overview of which SDA object was mapped to a particular FHIR resource (or vice-versa) while providing the ability to drill down into the mapping to understand exactly how the properties of the SDA object resulted into fields of a FHIR resource (or vice-versa). When using the **FHIR Annotations** tool, SDA properties are referred to as fields, for example, mappings are referenced as being field-to-field rather than property-to-field. You can also explore how lookup tables were used to map codes between SDA and FHIR, learn more about the data types involved in the transformation, and discover which ObjectScript methods were used in the transformation.

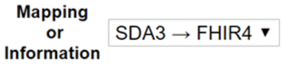
To understand the logic behind mappings, see Mapping Conventions.

9.3.1 Accessing the FHIR Annotations Tool

To access the **FHIR Annotations** tool:

1. Log in to the Management Portal as a user with the **`%Ens_EDISchemaAnnotations`** role.
2. Navigate to **Health** —> *MyFHIRnamespace*.
3. Expand the **Schema Documentation** menu option and click **FHIR Annotations**.

To begin exploring the mappings, use the **Mapping or Information** drop-down list to select the source and target of the transformations. For example, if you are interested in SDA3 to FHIR R4 mappings, select **SDA3 —> FHIR4**.

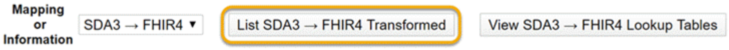


Mapping or Information: SDA3 → FHIR4

While using the **FHIR Annotations** tool to explore the SDA-FHIR mappings, you can select the **Help** and **FAQ** buttons to obtain guidance on using and interpreting the user interface of the tool. In addition, hover text is available over many of the elements of the user interface.

9.3.2 Mappings Overview

Before drilling down into the details of a particular mapping (including field-to-field mappings), it can be useful to gain an overview of all the mappings between SDA objects and FHIR resources. To view a list of how objects and resources map to each other, select **List <transform> Transformed**.

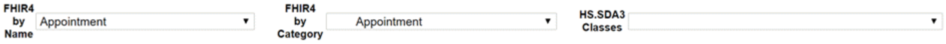


Mapping or Information: SDA3 → FHIR4

List SDA3 → FHIR4 Transformed
View SDA3 → FHIR4 Lookup Tables

9.3.3 Mapping Details

If you are interested in the details of how a specific SDA object or FHIR resource is mapped to the target format, you can select the object or resource from the drop-down lists. For example, to view the mapping of the Appointment resource to SDA3, select **Appointment** from the **FHIR4 by Name** drop-down list.



FHIR4 by Name: Appointment

FHIR4 by Category: Appointment

HS.SDA3 Classes:

Each mapping is presented in a table that shows all of the SDA field-FHIR field mappings, cardinality of the source field, data type of the source field, and other useful information. To discover more details about the elements in the table, you can:

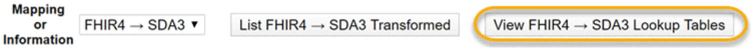
- Hover over each element in the table to obtain additional information.
- Click the links to open more details about that element, including the icons in the **Actions** column. For example, you can click a data type to explore how that data type is mapped.
- Click the Mapping Definition icon ( ) to drill down into the technical details of the mapping. Once the Mapping Definition opens, you can click the FHIR data types to bring them up in the official FHIR specification. You can also view technical details like cardinality, default values, and the subtransformation or class method used by the mapping. In some cases, there are additional notes that help explain the mapping.

The following is a legend of the icons in the **Actions** column of the mapping table:

Icon	Meaning
👓	View the detailed Mapping Definition.
🔍	Mapping uses a subtransformation or class method.
fx	Mapping uses a Condition to Set this Field to control whether the mapping is used.
⚙	Mapping uses a FHIR extension as its target.
✓	Mapping assigns a default value to the target when the source contains missing or invalid data.
📥	When the mapping is used, the transformation appends data to an existing target object, rather than creating a new target object.
▦	View mappings for the subfields of the FHIR resource field.

9.3.4 Lookup Table Mappings

The **FHIR Annotations** tool allows you to view the lookup tables that map codes between SDA and FHIR. For example, you can discover that the code A in the Status property of a HS.SDA3.Alert object maps to the `in-progress` code of the FHIR `event-status` value set. To explore the lookup tables used to map codes from the source to the target, select **View <transform> Lookup Tables**.


Mapping
or
Information
FHIR4 → SDA3 ▾
List FHIR4 → SDA3 Transformed
View FHIR4 → SDA3 Lookup Tables

To customize a lookup table, see Customizing Lookup Tables.

9.3.5 Mapping Conventions

This section explains the logic behind the SDA-FHIR mappings.

9.3.5.1 Field-to-Field Mappings

Most mappings are field-to-field: The mapping finds a data value in a source field and assign that value to a target field. For example, the value of a SDA property is assigned to a FHIR field.

9.3.5.2 Conditional Mappings

Some field-to-field mappings are conditional; the value is assigned to the target field only if certain conditions are met. The **FHIR Annotations** tool shows the label **Condition to Set this Field** when it presents this information. The DTL <if> element controls this in the code.

9.3.5.3 Literal Values

Among the defined mappings are mappings of literal values to target fields. One purpose of these mappings is to provide values for required target fields when the source object definition contains no fields that could provide the data required by the target.

Often, mappings of this type are defined conditionally, to be used only when needed.

9.3.5.4 Excluded Fields

SDA fields that contain metadata without clinical significance are not mapped to a FHIR field. For example, the UpdatedOn property is not transformed into FHIR.

In addition, SDA fields that are marked as Not Used in the class reference are not transformed into FHIR. For example, the ExternalId property of `LabResultItem` is not mapped to a field in the Observation resource.

9.3.5.5 Mapping Single to List

When the source field is single but the target field is a list, the transformation maps the source item to the first entry in a target list. After the transformation, the list contains only one entry. This feature is handled automatically during code generation for transformations. Single to List does not require special attention in the mapping definitions.

9.3.5.6 Mapping List to Single (Values)

When the source field is a list of values, and the target field is limited to a single value, the transformation concatenates the list of values into a single value, separating each value in the list with a semicolon and space.

9.3.5.7 Mapping List to Single (Objects)

For SDA to FHIR: when the incoming SDA is a list of objects, and the target FHIR has only one object, the mapping table contains two mapping entries for the source list field:

- One mapping maps the source list field to the target single field. The transformation generated from this mapping simply places the first list entry into the target field.
- The other mapping maps the source list field to the target FHIR extension that contains the full list of objects. The FHIR extension URL is the full source field name, including the resource name, but using all-lowercase text separated by hyphens.

For FHIR to SDA: when the incoming FHIR has a list of objects, and SDA has one object, the transformation uses the first object and drops all the others.

9.3.5.8 Mapping SDA CodeTableDetail to a FHIR Code

Transformations map an SDA **`CodeTableDetail`** (or one of its subclasses) to a FHIR coded object such as **`Coding`** or **`CodeableConcept`** as follows:

1. The Code value is mapped to the code field.
2. The Description is mapped to the display field.
3. If there is an OriginalText field, it is mapped to the text field.

9.3.5.9 Mapping Coded Values to FHIR using Lookup Tables

The mapping consults a lookup table to find the entry that maps code values from the source schema (SDA or FHIR) to code values in the target schema (FHIR or FHIR DSTU2) for this mapping.

If the mapping cannot find the lookup table, or cannot find a matching entry in the lookup table and it has a non-empty default value defined, it applies its default value to the code field. Otherwise, the target receives no value from this mapping.

If the mapping is SDA to FHIR, and the source field contains a non-empty value, then by convention there are two mapping entries for this source field. Both entries execute under the same Condition to Set this Field:

- One entry does the lookup to retrieve the value to assign to the target field.
- The other stores the original source field value in a string-valued FHIR extension.

In either case, if there is a Description or OriginalText along with the Code value, it is mapped to FHIR where applicable.

9.3.5.10 Mapping a FHIR Code to SDA CodeTableDetail

When a FHIR primitive code or coded object such as **`Coding`** or **`CodeableConcept`** does not use a lookup to transform the code value from FHIR to SDA, it is transformed to SDA **`CodeTableDetail`** (or one of its subclasses) as follows:

- CodeableConcept.text is transformed to HS.SDA3.CodeTableTranslated.OriginalText
- CodeableConcept.coding.display (or Coding.display) is transformed to HS.SDA3.CodeTableDetail.Description
- CodeableConcept.coding.code (or Coding.code, or simply code) is transformed to HS.SDA3.CodeTableDetail.Code
- GetCodeforURI of CodeableConcept.coding.system (or Coding.system) is transformed to HS.SDA3.CodeTableDetail.SDACodingStandard
- CodeableConcept.coding.version (or Coding.version) to HS.SDA3.CodeTableDetail.CodeSystemVersionId

9.3.5.11 Mapping FHIR Coded Values to SDA using Lookup Tables

If you want a mapping to use a code lookup table for FHIR to SDA, the mapping table contains two mapping entries for the source field:

- One of the two entries consults a lookup table to find the entry that maps a FHIR code value to an SDA Code.
- The other mapping entry in the pair takes over when the lookup table entry is unavailable or does not provide a match. It maps the source FHIR code value (unchanged) into an SDA **`CodeTableDetail`** object, as described above. That is, if the FHIR code was inside a **`Coding`** or **`CodeableConcept`** object, the FHIR code, display, system, version, and text values all are mapped appropriately into SDA **`CodeTableDetail`** fields.

9.3.5.12 Mapping SDACodingStandard

When the transformation encounters the SDACodingStandard property of an SDA object, it checks to see if the SDACodingStandard value is in the OID registry, and does one of the following:

- If the SDACodingStandard value is an entry in the OID registry that includes a URL, the transformation sets the `system` field of the FHIR Coding resource to the URL.
- If the SDACodingStandard value is an entry in the OID registry that does not define a URL, the transformation sets the `system` field of the FHIR Coding resource to the OID.
- If the SDACodingStandard value is not an entry in the OID registry, the transformation stores the value in a FHIR extension.

9.3.5.13 Mapping String Values to Numeric Values

When the target is FHIR, and a string value is mapped to a numeric value, the string may contain non-numeric text such as units of measurement or instructions. To handle this, there are two mapping entries for the source list field:

- One of the two entries always assigns the source string value to a FHIR extension that consists of one string-valued field.
- The other mapping entry tests the source string value to see if it is numeric. If so, it maps this numeric value to the target numeric field.

9.3.5.14 Multi-Part Literal Values for FHIR Code Objects

For some FHIR target fields that are **`Coding`** or **`CodeableConcept`** objects, a set of mappings from literal values forms a multi-part value that is assigned to the field when needed. The full set of fields that such an object can contain are: `code`, `system`, `display`, `text`, `version`, and `userSelected`.

Where this is the case, the DTL annotation element for the code field explains that this code resides within a **`Coding`** or **`CodeableConcept`** object that is receiving a multi-part literal value. The FHIR Annotations show that the set of literal value mappings relating to this code all have the same value in the Condition to Set this Field.

9.3.5.15 Mapping to FHIR Extensions

When the target of a transformation is FHIR, one or more SDA properties might not have a corresponding field in the target FHIR schema. In that case, transformations map the SDA data to a FHIR extension. The URL prefix for the extension is `http://intersystems.com/fhir/extn/sda3/lib`. The full URL is the full SDA property name, including the resource name, but using all-lowercase text separated by hyphens.

For example, the FHIR extension for the SDA property HS.SDA3.Administration:AdministeredAmount is:

- Extension name: `administration-administered-amount`
- Full URL for the FHIR extension: `http://www.intersystems.com/fhir/extn/sda3/lib/administration-administered-amount`

9.3.5.16 Mapping SDA CustomPairs

The transformations support the legacy CustomPairs property in SDA classes of type HS.SDA3.SuperClass.

CustomPairs is a collection of objects of type HS.SDA3.NVPairs, each of which has two properties, Name and Value. When the transformation code encounters this property in customer SDA data, and the target is FHIR, the collection is mapped to a FHIR extension that contains a Parameters resource. This Parameters resource is a collection of paired fields: name and valueString.

In the example below, the customized SDA Encounter object has an SDA CustomPairs collection with three members, each with the name `PlanOfCareInstructionsText`:

```
{
   "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 Customizing Transformations

Each SDA-FHIR transformation uses a Data Transformation Language (DTL) class to map SDA objects to FHIR resources, and vice versa. You can customize these DTLs using the DTL Editor.

If you want to implement more advanced custom transformation behavior, you can subclass the appropriate transformation API class and override its methods. For more information, see Customizing Transformation APIs. For information about upgrading to the new customization architecture from a legacy FHIR implementation, see Upgrading Pre-2020.2 Transformations

InterSystems products also provide a mechanism for customizing lookup tables used by the transformations.

You customize a transformation within a specific namespace, not for the entire instance, so you can have different customizations in each namespace. If you want multiple namespaces to have the same customized transformations, you must repeat the customization process for each namespace.

9.4.1 Implementing Custom DTLs

The strategy for customizing a DTL that the transformation uses to convert SDA to FHIR (and vice-versa) involves creating a copy of the standard DTL and then modifying it. After you manually specify the package of custom DTL, the transformation will automatically select the custom DTL instead of the standard one.

9.4.1.1 Specifying a Package for Custom DTLs

Before customizing DTLs, you need to specify a single package for all customized DTL classes. InterSystems recommends naming the class package: `HS.Local.FHIR.DTL`. Once you have decided on the package that will be used for all custom DTLs, you need to use the InterSystems Terminal to specify this package. To specify the custom DTL package:

1. Open the InterSystems Terminal.
2. Change to namespace that contains the SDA-FHIR transformations. For example:

   ObjectScript

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

3. To check if a custom DTL package already exists, enter:

   ObjectScript

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

4. If the custom DTL package does not already exist, enter the following command, replacing `HS.Local.FHIR.DTL` with the name of your custom DTL package:

   ObjectScript

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

5. To check that the package was defined successfully, enter:

ObjectScript

```
write status
```

The response should be: `1`.

9.4.1.2 Creating the Custom DTL

You create a custom DTL by saving a copy of the existing standard DTL and then editing it. The package and name of the custom DTL must conform to naming standards so the transformation knows to use the custom DTL rather than the standard one. To create a custom DTL:

1. Open the Management Portal and navigate to the FHIR namespace.
2. Select **Interoperability** > **List** > **Data Transformations**.
3. Find the name of the transformation that you want to customize. For example, transformations from SDA to FHIR STU3 are prefixed with `HS.FHIR.DTL.SDA3.vSTU3` while transformations from FHIR STU3 to SDA are prefixed with `HS.FHIR.DTL.vSTU3.SDA3` .
4. Double-click the name of the transformation you want to customize to open it in the DTL Editor.
5. Open the InterSystems Terminal.
6. To obtain the required name for the customized DTL class, enter the following in the Terminal:

   ObjectScript

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

   Where *`standard_class_name`* is the full name of the transformation that you are customizing, including packages. It is the name of the transformation that you have open in the DTL Editor. You can view the name on the **Transform** tab, but do not include the `.dtl` extension.
7. Be sure to make note of the response in the Terminal. You need to give your customized DTL class this name.
8. In the DTL Editor, click **Save As** .
9. In the **Package** field, enter the package from the name of the customized DTL class that appeared in the Terminal. For example, if the customized class name in the Terminal was HS.Local.FHIR.DTL.SDA3.vSTU3.Address.Address, enter `HS.Local.FHIR.DTL.SDA3.vSTU3.Address` (without the actual class name).
10. In the **Name** field, enter the name of the customized class. For example, if the customized class name in the Terminal was HS.Local.FHIR.DTL.SDA3.vSTU3.Address.Address, enter `Address`.
11. Enter a description and click **OK**.

9.4.1.3 Copying Custom Class to Mirror Members

If your environment uses mirroring and the package of your customizations resides in a non-mirrored database, you must copy the customized DTL class to the custom package on each mirror member. For example, if you defined the package for customized classes as HS.Local.FHIR.DTL, then you must copy the customized DTL class to HS.Local.FHIR.DTL on each mirror member because `HS.Local` resides in the HSCUSTOM namespace, which is not mirrored. If your custom package resides in a mirrored database, no further action is required.

9.4.2 Customizing Transformation API Classes

The transformation API classes contain several methods that can be overridden to implement custom transformation behavior. To override a method, subclass HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR or

HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 and write your custom method. For example, if you want to select a DTL based on a condition, you can override the `GetDTL()` method. The following is a brief introduction to the overridable transformation methods.

9.4.2.1 SDA to FHIR Overridable Methods

The following methods of the HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR class can be overridden to implement custom transformation behavior.

`GetDTL`

Specifies the DTL class used to transform a given SDA object. You do not need to override this method to use a custom DTL; if you specified a custom DTL package, the `GetDTL()` method finds the custom DTL before using the standard one. However, you can override this method if you want to select a DTL from multiple possibilities based on a condition.

`IsDuplicate`

Override this method to change how the transformation checks whether a generated resource that is referenced by another resource in the bundle already exists. For example, you might want to relax what is needed to identify a shared resource like Organization, Practitioner, or Medication as a duplicate. By default, the first 32 kilobytes of a shared resource are added as a hash in a hash table. For each subsequent reference to a shared resource, the transformation determines whether the referenced resource is a duplicate by searching the hash table for a direct match of the JSON.

If the `IsDuplicate()` method determines that a referenced resource already exists, it is not included in the bundle output.

`ResourceLookup`

By default, only the bundle created by the transformation is searched for a specified resource when the `ResourceLookup()` method is called. However, you can override this method, for example, if you want the application to search for the specified resource in a repository as well as in the bundle output.

`GetReference`

When transforming SDA that has a reference to another streamlet, this method returns the reference to the FHIR resource that is created for the referenced SDA object. For example, when an `EncounterNumber` is passed to this method, it returns a reference to the FHIR Encounter resource that corresponds to the SDA Encounter that was referenced by the specified `EncounterNumber`. Override the method to generate a custom reference to the specified FHIR resource.

`GetId`

By default, an individual resource is not assigned an `id` when the transformation produces a bundle. Override the `GetId()` method to assign resources in the bundle an `id`. In this case, the value for the `fullUrl` field of the bundle is *`baseURL/resourceType/id`* and the resource references in the bundle are *`resourceType/id`*.

`GetBaseURL`

Override the `GetBaseURL()` method to change the URL prefix of each resource. For example, if you are posting FHIR resources to a specific repository, you can provide a URL prefix that identifies the repository.

HandleInvalidResource

The transformation validates each resource before adding it to the Bundle output. Override the `HandleInvaidResource()` method to customize what happens to a resource that fails validation. By default, an error is thrown and processing stops, which means the Bundle is not returned.

9.4.2.2 FHIR to SDA Overridable Methods

The following methods of the HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3 class can be overridden to implement custom transformation behavior.

GetDTL()

Specifies the DTL class used to transform a given FHIR resource. You do not need to override this method to use a custom DTL; if you specified a custom DTL package, the `GetDTL()` method finds the custom DTL before using the standard one. However, you can override this method if you want to select a DTL from multiple possibilities based on a condition.

GetResourceFromReference()

This method controls where the transformation looks for a resource that has been referenced by another resource in the bundle. For example, you could override the method to find the referenced resource in a repository rather than in the same bundle.

GetSendingFacility()

Override this method to customize how the value of the SDA SendingFacility property is set.

By default, the SendingFacility property is set as follows: if the Patient’s `managingOrganization` field contains a reference to an Organization, and that Organization is in the Bundle, it is used. Otherwise, the patient identifiers are searched for an MRN with an assigning authority, and that assigning authority is used. If neither of these items is found, the string `FHIR` is used.

GetIdentifier()

Override this method to customize how certain identifiers are assigned to SDA properties.

For example, this method can be customized to assign values to the EncounterNumber field of an Encounter streamlet. In this case it can be useful to access the contents of the resource being converted in order to determine what EncounterNumber should be returned. The instance property `%currentReference` contains a FHIR reference object that can be passed into the instance method `GetResourceFromReference()` in order to obtain the resource as a dynamic object. By default, the value of the EncounterNumber properties are assigned sequentially, starting at `1`.

Overriding this method can also be useful for assigning the ExternalID value for the SDA HealthConcern or Goal. By default, the value of ExternalID properties are assigned sequentially, starting at `1`.

HandleMissingResource()

By default, if a resource that is referenced by another resource within the incoming FHIR bundle is not present in the bundle, the transformation of the bundles continues. To change what happens when there is a missing resource in the bundle, override the `HandleMissingResource()` method.

9.4.3 Customizing Lookup Tables

The FHIR Annotations tool allows you to explore the lookup tables that are used by transformations to map codes in the source data format to codes in the target format. You can customize these lookup tables by using a InterSystems Terminal utility or by manually modifying a JSON file that contains the lookup tables.

9.4.3.1 Using the Terminal Utility to Customize a Lookup Table

InterSystems provides a Terminal utility that leads you through the process of customizing a lookup table. To run the customization utility:

1. Open the InterSystems Terminal.
2. To change to the FHIR namespace, enter:

   ObjectScript

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

   where `Myfhirnamepace` is the FHIR namespace you have created.
3. To start the utility, enter:

   ObjectScript

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

4. Enter the Mapping Source for the lookup table you are customizing. For example, if you are customizing a lookup table that maps values from SDA3 to STU3, enter `SDA3` .
5. Enter the Mapping Target for the lookup table you are customizing. For example, if you are customizing a lookup table that maps values from SDA3 to STU3, enter `STU3`.
6. Enter the number that corresponds to Mapping Source Value Set in the lookup table you want to customize.
7. If only one lookup table with the Mapping Source Value Set exists, the Mapping Target Value Set is selected automatically and you can skip to the next step. If not, enter the number that corresponds to the Mapping Target Value Set you want to customize.
8. Select the code-to-code mapping you want to edit. If you want to add a new code-to-code mapping in the lookup table, enter `+` .
9. If you are editing the target value of a code-to-code mapping, enter the new target value for the mapping.

   If you want to edit the source value of the code-to-code mapping, you must enter `-` to delete the entire code-to-code mapping, then re-run the utility to add a new mapping with the correct source and target values.

9.4.3.2 Editing Lookup.json to Customize a Lookup Table

Rather than using the Terminal utility, you can customize lookup tables by adding, deleting, or editing key-value pairs in a JSON file that contains all of the lookup tables used by transformations. Before beginning, you must make a custom copy of the supplied Lookup.json file and put it into a namespace-specific directory under the custom directory.

Creating a Custom Lookup.json File

To create a custom JSON file in a durable location that will be used by transformations when accessing lookup tables:

1. Create a *custom* directory for your FHIR *namespace* in a durable location:

- In a kit installation:

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

- In a container:

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

  where *<MYFHIRNAMESPACE>* is the name of your FHIR namespace in all capital letters. For example, if the namespace that contains your FHIR production is called `Myfhirnamespace` , create a directory called MYFHIRNAMESPACE .

2. Copy Lookup.json from the <install-dir>\dev\fhir\lookup directory to the new custom namespace directory you created.

   You can now begin to edit the lookup tables in the new copy of Lookup.json .

Editing Custom Lookup.json File

To begin customizing a lookup table, you must gather four pieces of information:

- Mapping Source
- Mapping Target
- Mapping Source Value Set
- Mapping Target Value Set

These values can be found in the FHIR Annotations in the Management Portal. To access these values:

1. Open the Management Portal and navigate to your FHIR namespace.
2. From the Home page, select **Health** > **Schema Documentation** > **FHIR Annotations** .
3. In the first drop-down list, select the type of transformation that contains the lookup table you are customizing. For example, if you are interested in how SDA3 and FHIR STU3 codes map to each other in a lookup table, select **FHIR3–>SDA3**.

   Make note of the Mapping Source and Mapping Target. The first interface format in the transformation pair is the Mapping Source. The second interface format is the Mapping Target. For example, if you select **FHIR3—>SDA3** , vSTU3 is the Mapping Source and SDA3 is the Mapping Target.
4. Click the **View <transformation> Lookup Tables** button, where the full name of the button depends on which transformation pair you selected.
5. Using the **View Lookup Tables** dialog, use the drop-down lists to note the Mapping Source Value Set and Mapping Target Value Set. The Mapping Source Value Set is the name in the left-hand drop-down list. The Mapping Target Value Set is the name in the right-hand drop-down list.

Now that you have the Mapping Source, Mapping Target, Mapping Source Value Set, and Mapping Target Value Set, you can edit a lookup table by adding, deleting, or editing the appropriate key-value pair in the custom Lookup.json file.

The top-level key-value pair in Lookup.json corresponds to the Mapping Source to Mapping Target relationship. For example, a lookup table used by SDA3 to FHIR STU3 transformations looks like:

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

The next level of key-value pairs corresponds to the Mapping Source Value Set to the Mapping Target Value Set. Search for the correct lookup table by finding the corresponding key-value pair. For example:

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

Once you have located the lookup table, you can add, delete, or edit the key-value pairs that correspond to the code-to-code mappings.

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

Loading Custom Lookup.json File

Once you have customized `Lookup.json`, you need to load it using the Terminal before it can be used by the SDA-FHIR transformations. To load the JSON file:

1. Open the Terminal.
2. Change to your FHIR namespace. For example:

   ObjectScript

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

3. Run the following command:

   ObjectScript

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

10

FHIR Clients

InterSystems products come with standard FHIR client classes that your standalone ObjectScript application or interoperability production can use to send an HL7® FHIR® request to a FHIR REST endpoint over HTTP or to a local InterSystems FHIR server. The methods that your application uses to make the requests are the same regardless of which FHIR client class your application is using. In each case, after instantiating the client class that corresponds to your use case, the application calls the method that corresponds to a FHIR interaction or operation.

You have three client classes to choose from:

HS.FHIRServer.RestClient.HTTP

Sends a FHIR request over HTTP to a FHIR endpoint. When instantiating the class, the URL of the FHIR server's endpoint is identified by an entry in the Service Registry.

HS.FHIRServer.RestClient.FHIRService

Sends a FHIR request to the Service of an InterSystems FHIR server in the same namespace. When instantiating the class, the InterSystems FHIR server is identified by the server's endpoint (for example, `/fhirapp/fhir/r4`)

HS.FHIRServer.RestClient.Interop

Uses an interoperability production to send a FHIR request over HTTP to a FHIR endpoint. It has two variations:

- Send out a FHIR payload that has been formulated within a custom business host or retrieve FHIR data from within a business host.
- Route a FHIR request from a standalone ObjectScript application through an interoperability production before being sent over HTTP.

For details about this interoperability FHIR client, see Interoperability FHIR Client.

These classes all inherit from a single base class, HS.FHIRServer.RestClient.Base, that contains the logic for the methods that a FHIR client uses to perform a FHIR interaction or operation. Each type of FHIR client is instantiated with a `CreateInstance()` method.

10.1 Interactions and Operations

Within the RESTful architecture of the FHIR specification, a FHIR client works with resources on the server through interactions. A FHIR client developed with InterSystems technology provides methods that correspond to these interactions, allowing your ObjectScript code to perform an interaction with a single method call.

While the FHIR client provides at least one method for every interaction, it provides a single method regardless of which operation you are performing on the FHIR server. For details on invoking this method to perform an operation, see **Operation( )** in the Class Reference.

10.1.1 Calling an Interaction Method

If your FHIR client is writing to the server with interactions like `update`, it must use the `SetRequestFormat()` method to specify the format of the payload being written to the server. Possible formats are `JSON`, `XML`, `Form`, `XPatch`, and `JPatch`. Similarly, your FHIR client can specify the preferred format of the resources returned by the FHIR server using the `SetResponseFormat`. Possible formats are `JSON` and `XML`.

Unless the request and response formats change for individual interactions, your application can set them once and have them applied to all interaction methods. For example, a standalone FHIR client sending requests to a FHIR server over HTTP might set the request and response formats immediately after instantiating the client.

ObjectScript

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

Once the FHIR client class has been instantiated and the request and response formats set, the application can call methods that correspond to the FHIR interactions they want to perform on the server. To explore the FHIR interaction methods, including signatures, that are available to a FHIR client, refer to HS.FHIRServer.RestClient.Base in the Class Reference. Note that FHIR interactions that allow conditional actions have two different methods. For example, your application can call `Update()` or `ConditionalUpdate()` depending on whether the `update` interaction is conditional.

The data type of the payload that is passed as an argument is determined by the type of FHIR client that has been instantiated.

- For clients accessing a FHIR server over HTTP, the payload argument can be a string or stream.
- For clients accessing an InterSystems FHIR server in the local namespace, the payload argument can be a string, stream, or dynamic object.

The following is an example of instantiating a FHIR client and performing a `read` interaction on the external FHIR server:

ObjectScript

```
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 Including Custom Headers

If your FHIR request requires custom header information, for example to pass in an API key, use the **SetOtherRequestHeaders()** method. This method takes as an input a multidimensional array by reference, where each custom header has a subscript in the array. To populate an array with a custom header, provide a *header name* and a *header value*:

ObjectScript

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

To clear the “other headers” collection for the Rest **`clientObj`** instance, use the **ClearOtherRequestHeaders()** API method. This method takes no arguments:

ObjectScript

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

10.2 Customizing Requests and Responses

Internally, each interaction method calls three overridable methods that can be customized to modify how a request is sent or to manipulate the response received by the request. These three methods, `MakeRequest()`, `InvokeRequest()`, and `MakeClientResponseFromResponse()` are implemented by each type of FHIR client, not in the base class. Refer to the comments in the FHIR client class for more information (HS.FHIRServer.RestClient.HTTP, HS.FHIRServer.RestClient.FHIRService, or HS.FHIRServer.RestClient.Interop).

10.3 Requests without FHIR Client Class

Though using a FHIR client class is recommended when making requests to an internal FHIR server from an ObjectScript application, it is possible to write custom classes that perform CRUD operations on the server without these standard client methods. For example, you can write a custom class to interact with the FHIR server without going through the Service, thereby bypassing restrictions on the interactions that are allowed. You can also make direct calls to the Service with the `DispatchRequest()` method. For more information about these special cases, see ObjectScript Applications.

11

FHIR Requests and Responses

This topic discusses the requests and responses used by FHIR servers and the FHIR Interoperability Adapter.

For information about the requests and responses used by the InterSystems FHIR client, see FHIR Clients.

11.1 Non-production Requests/Responses

For a FHIR server that does not use an interoperability production:

- The message class that the server architecture uses to pass HL7® FHIR® requests is HS.FHIRServer.API.Data.Request.
- The message class that the server architecture uses to pass responses from the server to the FHIR client where the request originated is HS.FHIRServer.API.Data.Response.

11.1.1 Accessing FHIR Payloads

By default, when a FHIR request is received by the REST handler, it stores the FHIR payload in the Json property of a Request object (HS.FHIRServer.API.Data.Request), which automatically puts the JSON structure into a dynamic object. FHIR requests that contain XML are converted to JSON before being represented as a dynamic object in the Json property. Responses from the FHIR server (HS.FHIRServer.API.Data.Response) also contain a Json property for FHIR data.

Working with FHIR data begins by getting access to the Json property of the request or response. Once you have the FHIR payload, you can manipulate it as a dynamic object. For examples, see FHIR Data.

11.2 Interoperability Requests/Responses

For a FHIR server, FHIR Interoperability Adapter, or FHIR client that leverages an interoperability production:

- The message class used to pass FHIR requests through the production is HS.FHIRServer.Interop.Request.
- The message class used to pass a response through the production HS.FHIRServer.Interop.Response.

  Note: If you construct a HS.FHIRServer.Interop.Response object, you must explicitly set the ContentType property in order to create the HTTP response Content-Type header. Setting the ResponseFormatCode in the HS.FHIRServer.API.Data.Response is not sufficient.

These classes include a property QuickStreamId that points to the FHIR payload.

11.2.1 Accessing FHIR Payloads

When a FHIR implementation is using an interoperability production, you access the FHIR payload of the message object differently than implementations where a production is not used. In production-based implementations, the request and response messages (HS.FHIRServer.Interop.Request and HS.FHIRServer.Interop.Response) contain a QuickStreamId that is used to access a QuickStream object containing the FHIR payload. Though an interoperability request message also contains a Request property of type HS.FHIRServer.API.Data.Request, this Request property cannot be used to access the FHIR payload because its Json property is transient (the same is true for interoperability responses). As a result, a business host in the production that needs to access the FHIR payload must use the QuickStreamID to obtain the payload.

If the payload is in JSON format, the business host can access the payload and convert it to a dynamic object in order to modify it. For example, a BPL business process could use the following code to access and modify the FHIR payload of a request message that is in JSON format:

ObjectScript

```
//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()
}
```

For more examples of manipulating FHIR data as dynamic objects, see FHIR Data.

11.3 ObjectScript Applications

If an ObjectScript application needs to retrieve resources from the Resource Repository, it can build a non-production request object (HS.FHIRServer.API.Data.Request) before dispatching it to the endpoint’s Service. If the application is retrieving data, it is returned as the non-production response object (HS.FHIRServer.API.Data.Response). For more details, see Direct Calls to DispatchRequest.

11.3.1 Setting the Client-Visible URL

In some cases—notably, when requests are forwarded to a FHIR endpoint through a proxy—the URL at which the FHIR server received a request may differ from the URL which was originally requested by a REST client.

In such cases, the FHIR Server’s rest handler determines the client-visible base URL from the content of a request object’s FORWARDED or X-FORWARDED HTTP headers. This logic is implemented by the GetBaseURL() class method of the HS.FHIRServer.Util.BaseURL class.

If your FHIR server must construct the client-visible URL according to a different logic, simply define a custom **GetBaseURL()** in the HS.Local.FHIRServer.Util.BaseURL class. A method defined in this class will override the original.

12

Working with FHIR Data

Within the FHIR server architecture, HL7® FHIR® data is represented in dynamic objects, so working with the data is a combination of knowing how to manipulate dynamic objects and how FHIR resources are represented in JSON. Consult the FHIR specification for details about JSON representations of FHIR resources.

If a FHIR payload is in JSON, for example in an Interoperability request or response, you can convert it to a dynamic object for manipulation using the %FromJSON method.

12.1 FHIR Data and Dynamic Objects

Since FHIR data is often represented as dynamic objects within InterSystems products, knowing how to work with dynamic objects is essential. The following code fragments provide an introduction to manipulating with dynamic objects that contain FHIR data. As you'll see, you need to be familiar enough with the FHIR specification to know the structure of fields in the JSON representation of a FHIR resource. For complete details on handling dynamic objects, see Using JSON.

These code examples assume you have a variable `patient` that is a dynamic object containing a FHIR Patient resource.

Searching for a Value

The following code searches through identifiers of the Patient resource looking for a particular system using two different approaches. In order to write this code, you would need to be familiar enough with the FHIR specification to know that the JSON structure of a Patient resource contains an `identifier` that has a `system` name/value pair.

ObjectScript

```
// Put JSON representation of Patient resource into a dynamic object
set patient = ##class(%DynamicObject).%FromJSONFile("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,!
    }
  }
}
```

Extracting a Value

The following code fragment extracts the family name from the Patient resource.

ObjectScript

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

Modifying a Value

The following code fragment sets the Patient resource’s `active` field, which is a boolean, to `0`.

ObjectScript

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

Adding a New JSON Object

When you want to add a new JSON object to an existing dynamic object, you can choose whether to use an ObjectScript syntax or a JSON syntax. For example, the following code adds a new `identifier` to the patient, using two different approaches that have the same result.

ObjectScript

```
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 FHIR Object Classes

The FHIR standard defines a huge number of resource types, with numerous elements, structures, and data constraints. Remembering the exact syntactic details for all of the resource types is a burden, and something as simple as misspelling a field name can result in errors and failure. FHIR payloads typically reside in %DynamicAbstractObject (DAO) structures, which are invisible to the auto-completion tooling within the InterSystems IRIS for Health ecosystem.

InterSystems IRIS for Health provides a set of FHIR R4 object classes, included in `HSLIB`, that enable your IDE to provide auto-completion prompts for FHIR resources, shifting the cognitive burden from recall to reference. You don’t have to remember how to spell that element name; the IDE reminds you.

12.2.1 Features of the FHIR Object Classes

Each R4 resource has a corresponding ObjectScript class in the HS.FHIRModel.R4 package. For example, the HS.FHIRModel.R4.AllergyIntolerance class corresponds to the AllergyIntolerance resource. These classes streamline development by providing a shared, predictable framework of data structures and methods for resources and constituent elements, as defined by the base specification.

Within the FHIRModel framework:

- Elements unique to a resource are modeled by a class within a subpackage named HS.FHIRModel.R4.[ResourceName]X, For example, HS.FHIRModel.R4.AllergyIntoleranceX.Reaction models the data structure of an AllergyIntolerance resource’s `reaction` element.
- A collection of elements is modeled by a class named SeqOf[ElementClassName]. For a collection that is unique to a resource, this class is implemented within the [ResourceName]X subpackage. For example, the HS.FHIRModel.R4.AllergyIntoleranceX.SeqOfAllergyIntoleranceXReaction models the collection of `reaction` elements for an AllergyIntolerance resource.
- A resource class includes an **Include[ElementName]()** method for each complex element or collection of elements within it. This method adds the appropriate nested data structure for the element or collection to the resource object.
- A collection class includes a **MakeEntry()** method, which adds a new element to the collection object.
- All classes implement a common set of methods for fetching, navigating, and mutating their contents. These methods are inherited from the %Library.AbstractSet class.
- A dynamic abstract object that represents the JSON for a FHIR resource can be converted to an instance of its corresponding FHIRModel class using the **fromDao()** class method. Conversely, an instance of a FHIRModel class that represents a FHIR resource can be converted to a dynamic abstract object using the **toDao()** method. It can then be converted to a valid JSON payload using the dynamic object's **%ToJSON()** method. Alternately, you can use the FHIRModel class’s **toString()** method to directly generate the string-formatted JSON payload.
- You can extend the FHIR object classes as needed.

12.2.2 Methods for Use with FHIR Objects

This list includes the methods you will most likely need when converting between FHIR resources represented as Dynamic Abstract Object (DAO) structures and FHIR objects, and when working with FHIR objects. For more detail about these methods, or for additional methods, see the %Library.AbstractSet class or the relevant HS.FHIRModel.R4 subclasses in the class reference.

A likely workflow is something like this:

1. Recieve a FHIR resource as a JSON payload.
2. Convert the JSON payload to a %DynamicObject (a subclass of %DynamicAbstractObject), as described in Working with FHIR Data.
3. Convert the %DynamicObject to an object of the analogous FHIRModel class using **fromDao()**.
4. Work with the FHIR object as needed.
5. Convert the FHIR object back into JSON format using **toString()**.

12.2.2.1 Conversion Methods for FHIR Objects

fromDao(dao As %DynamicAbstractObject) As <HS.FHIRModel.R4 subclass>

Converts from DAO to the specified FHIR object.

toDao() As %DynamicAbstractObject

Converts from FHIR object to DAO.

toString()

Converts from FHIR object directly to string-formatted JSON payload.

12.2.2.2 Fetch Methods for FHIR Objects

get(key As %DataType) as %Any

Get the element identified by the given key, which may be either a label for key-value collections or a numeric position in a zero-based sequence.

iterator() as %Iterator

Return an interator over the members of this set. The object returned will have the following methods:

- **hasNext()** — returns true (1) if there is more data waiting to be processed.
- **next()** — returns an actual tuple with properties named key and value, drawn from the data in the queue.

12.2.2.3 Set and Clear Operations for FHIR Objects

add(value As %Any) as %AbstractSet

Sequences only. Append the new member `value` to the sequence.

addAll(values As %AbstractSet) as %AbstractSet

Sequences only. Append all members of the sequence `values` to the current sequence.

clear()

Remove all elements from the current set.

put(key As %DataType, value As %Any) as %AbstractSet

Labeled sets only. Put `value` into the set and associated it with the label `key`. If an element is already associated with the label `key`, replace it with the new `value`.

putAll(keys As %AbstractSet, values As %AbstractSet) as %AbstractSet)

Labeled sets only. Put all `{keys[n], values[n]}` elements into the set for all `n` in `values`.

remove(key As %DataType) as %Any

Remove the member identified by `key` from the set.

replace(key As %DataType, value As %Any) as %AbstractSet

Labeled sets only. Replace the value of the element identified by `key` with the new `value` provided.

12.2.2.4 Introspection Operations for FHIR Objects

apply(expression As %Any) as %AbstractSet

Return an array of members matching the provided SQL-JSON Path Language (JPL) `expression`.

contains(key As %DataType) as %Boolean

Returns `true` (1) if `key` is currently a non-null member of the set or sequence; otherwise returns `false` (0). If the set is labeled, `key` should be a string; if dealing with a sequence, `key` should be a numeric value greater than or equal to zero.

containsAll(array As %DynamicArray) as %Boolean

Returns `true` (1) if the set contains all keys listed in `array`.

size() as %Integer

Returns the number of non-null members in this set.

12.3 Data Load Utility

The Data Load utility sends resources and bundles that are stored in a local system directory directly to the FHIR server with or without going over HTTP. The local FHIR data fed into the Data Load utility can be individual resources, bundles, or both. The data can be provided in any combination of JSON, NDJSON, and XML files. A common use of this utility is feeding large amounts of synthetic data from open source patient generators into the FHIR server.

If getting data to the FHIR server as fast as possible is the objective, it is better to send it directly to the server without using HTTP. In this case, pass the `FHIRServer` argument to the Data Load utility along with the server’s endpoint. For example, suppose the server’s endpoint is `/fhirapp/fhir/r4` and the directory that contains FHIR bundles is `c:\localdata`. To run the Data Load utility, enter

ObjectScript

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

The utility should print `Completed Successfully` when it is done processing the files. If it does not, you can print any errors by entering `Do $SYSTEM.Status.DisplayError(status)`.

Alternatively, you can send all the bulk data over HTTP by passing `HTTP` along with the name of a Service Registry HTTP service. For more information about creating a HTTP service, see Managing the Service Registry. For example, you could run:

ObjectScript

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

The Data Load utility’s **SubmitResourceFiles()** utility takes optional arguments which control whether it displays progress, logs statistics, limits the number of files in the directory that it will process, or applies a translate table. In addition, you have the option to specify the number of workers to process the files as a multi-threaded operation. For details on these arguments, see **HS.FHIRServer.Tools.DataLoader.SubmitResourceFiles()**.

The utility also provides an API for loading FHIR data asynchronously. Using the methods in this API, you can initiate a new Data Load operation using **Job()**. The **Job()** method returns a job ID for this operation by reference, which you can then use to check its status (using **Status()**), cancel it (using **Cancel()**), and clean up associated globals after it is complete (using **CleanUp()**).

13

FHIRPath

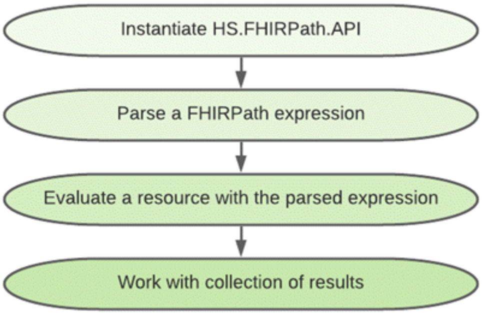
FHIRPath is a language, similar to XPath, that allows you to navigate an HL7® FHIR® resource to evaluate and extract data from its fields using a straightforward syntax that includes paths, functions, and operations. For example, you could evaluate whether the given name of a Patient contained a value: `Patient.name.given.empty()`. Or you could extract the value of the Patient resource's `telecom` field, but only if `offical` is the value of its `use` field: `Patient.telecom.where(use = 'official')`.

In FHIRPath, expressions are collection-based. Each function works on one input collection and each binary operator operates on two input collections, and the values returned by the expression are gathered into an output collection. Some functions and operations place constraints on the size of their input collections.

For complete details about FHIRPath including how to build an expression, see the HL7 FHIRPath specification. InterSystems supports a subset of the functions and operations that are defined in the specification.

13.1 Workflow

With InterSystems technology, the process of using FHIRPath to evaluate and extract data from a resource is straightforward:


Instantiate HS.FHIRPath.API
Parse a FHIRPath expression
Evaluate a resource with the parsed expression
Work with collection of results

The following sections provide details about each step in the workflow.

13.1.1 Instantiate HS.FHIRPath.API

The process of using FHIRPath to evaluate and extract data from a resource begins with calling **HS.FHIRPath.API.getInstance()**. When you call this method, you must specify the FHIR package that corresponds to a version of FHIR. For example, if the resources you are evaluating conform to FHIR R4, the corresponding package ID is currently `hl7.fhir.r4.core@4.0.1`. In this case, instantiating HS.FHIRPath.API would look like:

ObjectScript

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

You can obtain the IDs of the currently loaded packages using the Management Portal or ObjectScript:

- Management Portal — Navigate to **Home** > **Health** > *MyFHIRNamespace* > **FHIR Configuration**, and select the **Package Configuration** card. The package ID is obtained by appending the `@` symbol and version number to the name of the package. For example, the ID of the following package is `hl7.fhir.r4.core@4.0.1`:

hl7.fhir.r4.core 4.0.1

- ObjectScript — To list package IDs programmatically, see Listing Available Packages.

The HS.FHIRPath.API object includes the methods used to parse FHIRPath expressions and evaluate resources. This object is also included as a property on the HS.FHIRMeta.API object under the FHIRPathAPI property.

13.1.2 Parse the FHIRPath Expression

Once you have instantiated the HS.FHIRPath.API object, you are ready to parse the FHIRPath expression. The method that parses the expression, **HS.FHIRPath.API.parse()**, returns a tree structure that is used by the methods that evaluate a resource. For example, assuming you have an object named `fhirPathAPI` instantiated as shown in the previous section:

ObjectScript

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

13.1.3 Evaluate the Resource

Once you have parsed the FHIRPath expression, you can use its tree structure to evaluate or extract data from a resource. Two evaluation methods are available:

- **HS.FHIRPath.API.evaluate()** — The **evaluate()** method returns the results of the evaluation in a *multidimensional* array.
- **HS.FHIRPath.API.evaluateToJson()** — The **evaluateToJson()** method returns the collection in a *dynamic* array.

In both cases, the resource being evaluated is passed into the method as a dynamic object. The tree that was returned by the `parse()` method is also passed as an argument. For example:

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)
```

An additional method, **HS.FHIRPath.API.evaluateArray()**, can be used to parse the multidimensional array returned by the **evalaute()** method.

13.1.4 Work with the Results

While working with results in a *dynamic* array that is produced by **evaluateToJson()** has its benefits, the *multidimensional* array produced by **evaluate()** contains additional information that is not otherwise available. The following provides a guide to the data in the multidimensional array, assuming that your response to **evaluate()** was returned in a variable named `OUTPUT`.

Node	Description
`OUTPUT`	Number of nodes in the array that contain values.
`OUTPUT(n)`	Value of the *n*th element of the array.
`OUTPUT(n,"t")`	Data type of the *n*th element in the array, including identifying FHIR data types.

You can further parse the returned multidimensional array using the **evaluateArray()** method.

By contrast. when using the **evaluateToJson()** method to produce a dynamic array, you can determine whether the data type is a string, boolean, number, or object from looking at the values in the array, but you cannot determine the FHIR data type.

13.1.5 Workflow Example: evaluate() Method

This example includes the resource being evaluated, the ObjectScript needed to evaluate the resource, and a look at the multidimensional array produced by the evaluation.

Sample Resource

ObjectScript

```
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"
    }
  ]
}
```

Extracting Data from the Resource

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)
```

Viewing the Multidimensional Array

If you used the `zw OUTPUT` command in the InterSystems Terminal to view the multidimensional array returned by **evaluate()**, the result would be:

```
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"
```

Notice that the values are identified as a `ContactPoint` FHIR data type.

13.1.6 Workflow Example: evaluateArray() Method

This example takes the multidimensional array produced by the evaluation in the **evaluate()** example above as input and demonstrates the ObjectScript needed to evaluate the resulting array, and looks at the multidimensional array produced by the evaluation.

Extracting Data from the Output Array

ObjectScript

```
Merge INPUT = OUTPUT
Kill OUTPUT

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

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

Viewing the Multidimensional Array

If you used the `zw OUTPUT` command in the InterSystems Terminal to view the multidimensional array returned by **evaluateArray()**, the result would be:

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

Notice that the values are identified by their ObjectScript data type (string, boolean, number, or object).

13.1.7 Workflow Example: evaluateToJson() Method

This example includes the resource being evaluated, the ObjectScript needed to evaluate the resource, and a look at the dynamic array produced by the evaluation.

Sample Resource

ObjectScript

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

Evaluating the Resource

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)
```

Viewing the Dynamic Array

If you used the `zw dynArray` command in the InterSystems Terminal to view the dynamic array, the result would be:

```
dynArray=[false]
```

13.2 Functions

The FHIRPath specification defines a wide range of functions that can be used in an expression. InterSystems supports a subset of those functions.

Table 13–1: Supported FHIRPath Functions

Function	Example
[ ] (index)	`Practitioner.name[1]`
`aggregate`	`item.factor.aggregate($total+$this,0)`
`as`	`Condition.abatement.as(string)`
`children`	`Encounter.participant.children().ofType(Reference)`
`descendants`	`Bundle.descendants().ofType(Patient)`
`empty`	`Patient.contact.where(relationship = 'N').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`
`first`	`Patient.telecom.where(system = 'phone').first()`
`hasExtension`	Returns `true` if any of the input collection have an extension with the specified URL. (This function is not in the FHIRPath v2.0.0 specification.)
`iif`	`iif(1=1,2,3)`
`indexOf`	`'abcdefg'.indexOf('cd')`
`is`	`Condition.abatement.is(dateTime)`
`last`	`Patient.name.first().given.last()`
`not`	`Bundle.entry.resource.ofType(Patient).gender.not()`
`ofType`	`Bundle.entry.resource.ofType(Patient)`
`resolve`	`Organization.partOf.resolve()`
`skip`	`Bundle.entry.resource.ofType(Encounter).skip(5)`
`startsWith`	`'abcdefg'.startsWith('abc')`
`substring`	`'abcdefg'.substring(1, 2)`
`tail`	`Bundle.entry.resource.ofType(Observation).tail()`
`take`	`Patient.name.take(1)`

Function	Example
`union`	`Practitioner.name.family.union(Practitioner.id)`
`where`	`Patient.telecom.where(use = 'official')`

13.3 Operations

The FHIRPath specification defines a wide range of operations that can be used in an expression. InterSystems supports a subset of those operations.

Table 13–2: Supported FHIRPath Operations

Operation	Example

+ (addition)	`8 + 3`
	`5 seconds + 3 seconds`
	`'string1' + ' and ' + 'string2'`

& (string concatenation)	`'string1' & ' and ' & 'string2'`

= (equals)	`Practitioner.name[0].family = 'Cooper'`
	`Practitioner.meta.versionId = 10`

!= (not equals)	`Practitioner.name[1].family != 'Smith'`
| (union collections)	`Practitioner.name.family | Practitioner.id`
`and`	`true and false`
`as`	See implementation notes.
`implies`	`Patient.name.given.exists() implies Patient.name.family.exists()`
`is`	`Practitioner.name[0] is HumanName`
`or`	`true or false`
`xor`	`true xor false`

Implementation Notes for `as`

According to the FHIRPath specification, the left operand of the `as` operation must be a collection with a *single item*. However, the InterSystems implementation of FHIRPath allows this collection to have multiple values. For example, suppose you have an Observation with *multiple* extensions that reference a Patient. With the InterSystems implementation of FHIRPath, the following expression would still be valid: `extension.value as Reference`.

13.4 Improving Performance

InterSystems provides an in-memory cache that can store parsed FHIRPath expressions, improving performance when you have a set of expressions that are repeated frequently. Once the cache is enabled, tree structures produced by the `parse()` method are stored until the cache is cleared.

To enable the in-memory cache, call:

ObjectScript

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

To disable the cache, call:

ObjectScript

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

14

FHIR Server Security

You can control which clients can make requests to the FHIR server and the interactions they can perform using InterSystems security strategies and OAuth 2.0. If both forms of authentication are provided in the same request, both must be valid for the request to succeed.

During development and debugging, you can temporarily disable all security restrictions.

14.1 Basic Authentication

By default, the FHIR server enforces basic authentication in which any user with credentials to an InterSystems product can access the FHIR server by including those credentials in the header of the REST call. In this security strategy, the user's authorization within the InterSystems product is not a factor; any authenticated user can perform CRUD interactions on the FHIR server.

14.1.1 Adding Authorization Requirements

By adding authorization requirements to basic authentication, you can restrict server access to InterSystems users who are authorized to work with a specific security resource (which is unrelated to an HL7® FHIR® resource). In InterSystems security terms, only *users* who belong to *roles* that have *privileges* to the *resource* are authorized to perform interactions on the server. Users with a Write privilege to the required resource can perform create, delete, update, and conditional update interactions on the FHIR server. Users with a Read privilege to the resource can perform all interactions except the ones that require write access. Remember that FHIR transactions are recursive, so a user must hold Write privileges if the transaction request contains both read and write interactions.

The following is a basic overview of how to create a resource, assign privileges to the resource for a role, and assign users to the role. For a detailed description of InterSystems authorization, see the Authorization Guide; for an introduction to security, see About InterSystems Security.

1. To create the resource that controls whether users are authorized to perform interactions on the server, open the Management Portal and navigate to **System Administration** > **Security** > **Resources**. Setting the **Public Permission** to Read allows all authenticated users to perform Read interactions on the server. For more information, see Create or Edit a Resource.
2. To create a role that will have privileges to the resource, navigate to **System Administration** > **Security** > **Roles**. Most commonly, there will be two roles, one for users who should have Read access and another for users who should have Write access. For more information, see Create Roles.
3. To grant privileges to a role:

   a. Click **Add** in the **Privileges** section of the role’s **General** tab.
   b. Select the resource that will control server authorization, and click **OK**.
   c. Click **Edit** next to the new Privilege.
   d. Select the permissions you want the role to have for the resource.

   For more information, see Give New Privileges to a Role.

4. Now that you have a role that has permissions to the security resource, select the **Members** tab and add the users that you want to have those permissions. For more information, see Assign Users or Roles to the Current Role.

14.1.1.1 Configuring the Server

Once you have created or chosen the security resource that will control a user’s ability to perform FHIR interactions, use the following steps to configure the server to require this resource:

1. In the Management Portal, navigate to **Health** > **FHIR Configuration** > **Server Configuration**. Make sure you are in the FHIR server’s namespace.
2. Select the endpoint of the FHIR server.
3. Select **Edit**.
4. In the **Required Resource** field, enter the name of the security resource that controls access to the FHIR server.
5. Select **Update**.

14.2 OAuth 2.0 Authentication

By setting up the FHIR server as an OAuth 2.0 resource server, you can reject a client’s FHIR requests unless it has a valid access token that it obtained from an OAuth 2.0 authorization server. A FHIR request’s access token is checked twice, once by the REST handler and again when it reaches the FHIR server’s internal Service. Because the access token is enforced when the request hits the REST handler, the request has already been validated when it enters an interoperability production (if the FHIR server has been configured to use a production). The REST handler and the Service use the same class to validate the token, which is the class specified by the `OAuth2TokenHandlerClass` parameter of the server’s Interactions class. For a default FHIR server, this token handling class is HS.FHIRServer.Util.OAuth2Token.

The first step in identifying the FHIR server as a resource server is to create a client configuration using **System Administration** > **Security** > **OAuth 2.0** > **Client**. After creating a Server Description for the OAuth 2.0 authorization server, create a new client configuration for the FHIR server, specifying that it is of type Resource Server. For more information about setting up a resource server in InterSystems products, see Using an InterSystems IRIS Web Application as an OAuth 2.0 Resource Server.

Once you have defined the client configuration for the FHIR server:

1. In the Management Portal, navigate to **Health** > **FHIR Configuration** > **Server Configuration**. Make sure you are in the FHIR server’s namespace.
2. Select the endpoint of the FHIR server.
3. Select **Edit**.
4. In the **OAuth Client Name** field, enter the **Application Name** of the resource server as defined in the Management Portal.
5. Select **Update**.

14.2.1 Access Token Scopes

Note: Although `read`/`write` syntax is supported, permissions are best specified using SMART on FHIR v2–style syntax. See the HL7 specification for details.

This section explains how the FHIR server enforces the scopes of an OAuth 2.0 access token that is passed along with a request. If your FHIR server needs to interpret scopes differently, you need to customize the FHIR server and override the `OAuth2TokenHandlerClass` parameter to specify your custom token handling class.

Access tokens can have the following scopes:

- `Patient` scopes limit authorization to resources related to the patient specified in the patient context claim. They are likely to be used, for example, when a patient is looking at their data through a web portal.
- `User` scopes allow access to view or manipulate FHIR resource types that the particular user is authorized to access. This kind of authorization is subject to any implementation-specific authorization processing (for example, consent).
- `System` scopes represent external systems. They are used to facilitate system-to-system interactions such as bulk data extracts.

Note: When a FHIR interaction is authorized by `patient` or `user` scopes, it should be subject to any additional implementation-specific processing (such as consent) that may be in use. This type of additional processing is not expected for interactions authorized by `system` scopes.

Basic Processing

The access token that accompanies a request must include at least one server scope, patient resource scope, or user resource scope, or else the request is rejected with an HTTP 403 error. If multiple scopes are present, the union of the scopes is evaluated. For example, if both user and patient scopes are present, all scopes are potentially evaluated, until any of them authorizes the current FHIR interaction, or until none of them does.

Patient Resource Scope / Patient Context Value

If an access token includes a patient resource scope, it must also include a patient context value (also known as “launch context”) that is the Patient resource ID. This patient context value provides access to the specified Patient and its related resources. In most cases, the patient resource scope must provide explicit access to a related resource. For example, if the patient context value is `1234`, and the patient resource scope is `patient/Observation.cruds`, the FHIR server can grant access to an Observation that references the Patient with the id `1234`. In this case, `patient/Observation.cruds` (or another scope granting access to Observations) is required. As an exception to this requirement, a FHIR client can access a resource that is shared among multiple Patients without obtaining a patient resource scope that is specific to that resource. For example, if the scope is `patient/Patient.rs`, then a client can access an Organization referenced by the Patient without having a scope `patient/Organization.rs`.

To obtain the patient context value from the access token, the FHIR server examines the `patient` property of the access token.

Search

The FHIR server handles search requests accompanied by a valid access token in the following manner:

- `_include` and `_revinclude` parameters *are* allowed.
- If the FHIR server is enforcing a patient context value:
  - Chained and reverse chained (`_has`) parameters are allowed.
  - The search resource type must be allowed by the patient scopes.

  - If the search resource type is *not* Patient, reference search parameter values must indicate a Patient resource that is *in the patient context*.
  - If `_include` and `_revinclude` parameters are present they must indicate only pulling in resources that are allowed by the scopes.
  - For a Patient search, any `_id` value must match the patient context value.
  - In all other cases, perform the search and check that all of the resources in the result set are allowed by the scope and context.

`Create` Interaction

Requests to create a new Patient resource must include a user resource scope that gives write permissions (`user/Patient.cud` or `user/Patient.cruds`). You cannot perform a `.c` interaction for a Patient resource with a patient resource scope; patient resource scopes must include a patient context value, and the `.c` interaction cannot include a resource id.

`$everything`

Requests for the Patient or Encounter `$everything` operation must include an access token that has read access to all of the resources that might be returned by the request. If a resource is encountered in the compartment that is not covered by the scope, then the entire request is rejected with an HTTP 403 Forbidden error.

The practical application of this requirement is:

- If a `_type` operation query parameter is specified, then the scope must include read access to all of the resource types requested.
- If no types are specified and the access token is using a patient resource scope, it should have a `patient/*.rs` or `patient/*.cruds` scope in order to return any resource encountered in the compartment.
- If no types are specified and the access token is using a user resource scope, it should have a `user/*.rs` or `user/*.cruds` scope in order to return any resource encountered in the compartment.

14.3 No Authentication

While authentication is essential on a live FHIR server, being forced to provide credentials to the FHIR server during development and testing can be cumbersome. You can allow all FHIR requests to reach the server, temporarily ignoring authentication and authorization strategies. To allow unauthenticated access:

1. In the Management Portal, navigate to **Health** > **FHIR Configuration** > **Server Configuration**. Make sure you are in the FHIR server's namespace.
2. Select the endpoint of the FHIR server.
3. Select **Edit**.
4. Select the **Allow Unauthenticated Access** check box in the **Debugging** section.
5. Select **Update**.

15

FHIR Server Debugging

InterSystems provides a debug mode and logging to help debug a FHIR server during development

15.1 Debugging the FHIR Server

Putting the FHIR server in debug mode helps solve problems during development and can temporarily eliminate the need to authenticate HL7® FHIR® requests. To set debug options:

1. In the Management Portal, navigate to **Health** > **FHIR Configuration** > **Server Configuration**. Make sure you are in the FHIR server's namespace.
2. Select the endpoint of the FHIR server.
3. Select **Edit**.
4. In the **Debugging** section, select the check boxes of the debugging options you want to enable.
    - **Allow Unauthenticated Access** — Allows all FHIR requests to reach the server, ignoring authentication and authorization strategies.
    - **New Service Instance** — Instantiates a new Service object for every FHIR request. Set this option when making changes to your custom architecture classes, such as your Interactions and InteractionsStrategy subclasses.
    - **Include Tracebacks** — The FHIR server responds to a FHIR request by sending a stack trace in an OperationOutcome resource.
5. Select **Update**.

15.2 Logging

The FHIR server provides two types of logging:

- Internal FHIR Server Logging — Provides information about how the FHIR server architecture is processing FHIR requests, including which class methods are being called.
- HTTP Request Logging — Provides information about the HTTP requests coming from REST clients to the FHIR server.

15.2.1 Internal FHIR Server Logging

The FHIR server provides basic logging information about how the architecture is processing the FHIR requests being received by the server, including which class methods are being called, SQL-related messages, and how `_include` searches are being handled. To enable this type of logging:

1. Open the InterSystems Terminal.
2. Navigate to the FHIR server's namespace. For example, enter:

   ObjectScript

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

3. Create a global, `^FSLogChannel`, that specifies what type of logging information should be stored. The syntax for creating the global is:

   ObjectScript

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

   Where *channelType* is one of the following:

   - `Msg` — Logs status messages.
   - `SQL` — Logs SQL-related information.
   - `_include` — Logs information related to searches that use the `_include` and `_revinclude` parameters.
   - `all` — Logs all three types of information.

   For example, to enable logging for all types of information, enter:

   ObjectScript

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

Note: To switch to a new type of logging information (for example, from `Msg` to `SQL`), kill the existing `^FSLogChannel` global before setting it again with the new *channelType*.

15.2.1.1 Viewing the Log

Once logging for the FHIR server architecture is enabled, the log entries are stored in the `^FSLOG` global. To use the Management Portal to view the log, navigate to **System Explorer** > **Globals** and view the `FSLOG` global (not `FSLogChannel`). Make sure you are in the FHIR server's namespace.

Each node of the global is structured like:

`CurrentMethod^CurrentClass|LogType|LogMessage`

For example, a log entry in a node of the `^FSLOG` global might be:

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

15.2.1.2 Disabling Logging

To disable logging for the FHIR server architecture, simply kill the `^FSLogChannel` global or set it to `0`. For example, you can enter the following in the Terminal:

ObjectScript

```
kill ^FSLogChannel
```

15.2.2 HTTP Request Logging

When HTTP request logging is enabled, the REST handler that is receiving requests from FHIR clients writes information about each HTTP request to the `ISCLOG` global. To enable this type of logging:

1. Open the InterSystems Terminal.
2. From any namespace, enter the following commands to configure the global `^%ISCLOG` to start logging HTTP requests:

   ObjectScript

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

   Note that the global you use to configure logging (`^%ISCLOG`) has a different name than the global to which the logging information is written (`^ISCLOG`).

15.2.2.1 Viewing the Log

Once logging for HTTP requests is enabled, the log entries are stored in the `^ISCLOG` global, which is located in the `%SYS` namespace.

To use the Management Portal to view the log, navigate to **System Explorer** > **Globals** and view the `ISCLOG` global (not `%ISCLOG)`. Make sure you are in the `%SYS` namespace.

15.2.2.2 Disabling Logging

To disable HTTP request logging, open the Terminal and enter the following command:

ObjectScript

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

15.2.3 FHIR Test Utility

The FHIR Test Utility that appears in the Management Portal (**Health** > **FHIR Test Utility**) does not work with the current FHIR architecture. It still works with the legacy FHIR technology.

16

FHIR Server Maintenance

While maintaining a FHIR server that is in production, it might be necessary to stop processing HL7® FHIR® requests to the endpoint, then re-enable the endpoint when the maintenance is complete.

Note: Sometimes when you seek to modify the FHIR server, you may find the repository locked. This is normal: All InterSystems IRIS API methods lock the repository, and the repository can also be locked explicitly using the **##class(HS.FHIRServer.Repo).Lock()** method. If the repository is locked, you will have to wait until the other agent releases it.

To stop and re-start an endpoint:

1. In the Management Portal, navigate to **Health** > **FHIR Configuration** > **Server Configuration**. Make sure you are in the FHIR server's namespace.
2. Select the endpoint of the FHIR server.
3. Select **Edit**.
4. To make the FHIR server's endpoint available to requests, select the **Enabled** check box in the **Configuration** section. To stop an endpoint and reject requests, clear the check box.

17

Customizing a FHIR Server

When using a FHIR server, there are two strategies for customizing the behavior of the FHIR server. Like legacy HL7® FHIR® technology, you can use logic in interoperability productions to modify the server's behavior. However, you also have the option of customizing the architecture of the FHIR server to implement custom functionality. This option is important because a FHIR server that does not use an interoperability production can be significantly faster than one that does.

When customizing the server architecture, you are most commonly extending the Resource Repository, only customizing those parts of the server that are unique to your environment. In more rare cases, you may need to write an entirely custom backend for the FHIR server; the FHIR server's architecture gives you the flexibility to do this. Regardless of whether you are extending the Resource Repository or writing a custom backend, the process of customizing the FHIR server starts with pre-installation subclassing.

Some behavior of the FHIR server is controlled through configuration options that do not require customization of the architecture. For details about these options, see Configuring a FHIR Server.

As you customize your FHIR server, you can update the server's Capability Statement. For details, see Modifying the Capability Statement.

17.1 Pre-Installation Subclassing

Customizing a FHIR server begins with using an IDE to subclass the architecture and define a few parameters. Because the InteractionsStrategy is specified during installation, this step must occur *before* the server's endpoint is created by the installation process.

Most commonly, your FHIR server is extending the architecture of the Resource Repository. In these cases, open an IDE and subclass:

- HS.FHIRServer.Storage.JsonAdvSQL.Interactions
- HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy
- HS.FHIRServer.Storage.JsonAdvSQL.RepoManager

If you are writing an entirely custom backend for your FHIR server instead of using the Resource Repository, subclass the architecture superclasses: HS.FHIRServer.API.Interactions, HS.FHIRServer.API.InteractionsStrategy, and HS.FHIRServer.API.RepoManager.

17.1.1 Subclass Parameters

After using an IDE to create your Interactions, InteractionsStrategy and RepoManager subclasses, you must modify the following parameters of the InteractionsStrategy and RepoManager.

Superclass	Subclass Parameters
HS.FHIRServer.API.InteractionsStrategy	• `StrategyKey` — Specifies a unique identifier for the InteractionsStrategy. • `InteractionsClass` — Specifies the name of your Interactions subclass.
HS.FHIRServer.API.RepoManager	• `StrategyClass` — Specifies the name of your InteractionsStrategy subclass. • `StrategyKey` — Specifies a unique identifier for the InteractionsStrategy. Must match the `StrategyKey` parameter in the InteractionsStrategy subclass.

Once you have compiled your subclasses, you are ready to install the FHIR server. Simply specify the name of your InteractionsStrategy subclass during installation.

17.2 Activating Custom Code

When making changes to your custom Interactions or InteractionsStrategy code during development, use the **New Server Instance** debugging option to activate your new code when the next FHIR request is made. For details, see Debugging the FHIR Server .

17.3 Customizing the Resource Repository

Once you have subclassed the FHIR server architecture of the Resource Repository, you are ready to customize the server. Most commonly, your customizations involve overriding methods and parameters in the subclass of HS.FHIRServer.Storage.JsonAdvSQL.Interactions. The following is an introduction to the most common customizations that you can make to a FHIR server that uses the Resource Repository.

Table 17–1: Customization Quick Start

Goal	Action in subclass of HS.FHIRServer.Storage.JsonAdvSQL.Interactions
Customize a specific FHIR interaction	Override the method that corresponds to the interaction
Preprocess all requests	Override `OnBeforeRequest` to implement logic that is transparent to the user. This overridden method should include a call to the super class, for example: `Do ##super(pFHIRService, pFHIRRequest, pTimeout)`. If you want FHIR clients to be aware that a request is being handled differently, create a custom FHIR operation.
Post-process all requests	Override `OnAfterRequest` to implement logic that is transparent to the user. This overridden method should include a call to the super class, for example: `Do ##super(pFHIRService, pFHIRRequest, .pFHIRResponse)`. If you want FHIR clients to be aware that a request is being handled differently, create a custom FHIR operation.
Post-process results of a Read interaction	Override `PostProcessRead`. (Example)
Post-process results of a Search interaction	Override `PostProcessSearch` (Example)
Add custom FHIR operation	Override the `OperationHandlerClass` parameter to specify the name of your subclass of `HS.FHIRServer.Storage.BuiltInOperations`. See Custom FHIR Operations.
Customize how bundles are processed	Override the `BatchHandlerClass` parameter to specify the name of your custom class. The default handler class is HS.FHIRServer.DefaultBundleProcessor.
Customize how OAuth tokens are processed	Override the `OAuth2TokenHandlerClass` parameter to specify the name of your custom class. The default handler class is HS.FHIRServer.Util.OAuth2Token.

The following code samples demonstrate a few customizations that you could make to a FHIR server that uses the Resource Repository.

17.3.1 Post-Processing Results

It is common to want to manipulate the results of a Read interaction or Search interaction. For example, you might want to modify data in a Patient that is returned by a Read interaction or remove certain resources from the results of a search. In the following example, results are modified based on Consent rules; the sample code assumes you have written a separate class to handle the Consent processing. The roles extracted from the request are InterSystems security roles.

Class Definition

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.JsonAdvSQL.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: When customizing a FHIR server, it can be useful to determine if a resource is a shared resource. Shared resources do not contain Patient information; in FHIR terms, these resource types are not in the Patient compartment. You can use the `IsSharedResourceType()` method to determine if a resource is shared. For example, your custom Interactions class could include the following conditional statement:

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.JsonAdvSQL.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 Assigning Custom IDs to Resources

It is possible to customize a Resource Repository server to assign each resource a custom id when performing Create interactions. The following example assigns a random UUID to the resource when it is stored in the Resource Repository.

Class Definition

```
Class MyCustom.FHIR.Interactions Extends HS.FHIRServer.Storage.JsonAdvSQL.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 Modifying the Capability Statement

The FHIR server’s Capability Statement is client-facing metadata that documents how the server behaves; FHIR clients can retrieve the Capability Statement to determine what the server expects and how it will process FHIR requests. As you customize your FHIR server, you may want to update the Capability Statement so FHIR clients have an accurate description of what the server supports. You have two options for updating the Capability Statement:

- Retrieve the existing Capability Statement, edit its JSON, and then post it back to the server. Though straightforward, there is a limitation to this approach: the Capability Statement is automatically regenerated by certain actions, for example adding a new search parameter, so you might have to restore your customized Capability Statement after taking one of these actions. For details, see Manually Updating Capability Statement.
- Modify the InteractionsStrategy subclass by overriding the methods that generate the Capability Statement. This gives you greater control over the Capability Statement and will not cause problems when it is regenerated. For details, see Overriding Capability Statement Methods.

17.4.1 Manually Updating Capability Statement

You can retrieve the FHIR server’s Capability Statement with a REST client or programmatically, edit it with a text editor or third-party tool, and then update the server with the new version. Be aware that you may need to repeat this procedure after certain actions, for example, adding a new search parameter. Therefore, you may want to store a copy of the revised Capability Statement rather than recreating it when needed.

In the following examples, assume the IP address of the InterSystems server is `172.16.144.98`, the superserver port is `52782`, and the base URL of the endpoint is `/fhirapp/r4`.

- To retrieve the Capability Statement with a REST client, send a `GET` request to `*base-url*/metadata`. For example:

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

- To retrieve the Capability Statement programmatically and save it as a JSON file, enter:

ObjectScript

```
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")
```

Once you have modified the Capability Statement, submit the revised version to the server programmatically from the InterSystems Terminal. In the following example, `/fhirapp/r4` is the endpoint's base URL and `MyCapabilityStatment.json` is the revised version. The `{}.%FromJSONFile` method takes a JSON file and puts it into a dynamic object.

ObjectScript

```
set strategy = ##class(HS.FHIRServer.API.InteractionsStrategy).GetStrategyForEndpoint("/fhirapp/r4")

set interactions = strategy.NewInteractionsInstance()
set newCapabilityStatement = {}.%FromJSONFile("c:\localdata\MyCapabilityStatement.json")
do interactions.SetMetadata(newCapabilityStatement)
```

17.4.2 Overriding Capability Statement Methods

Because the Capability Statement is regenerated automatically when changing certain FHIR server behavior, you might want to override the methods used to generate the server's Capability Statement rather than manually updating it. This requires development tasks in an IDE, but gives you more control of the generation process. These tasks assume you have extended the Resource Repository by subclassing HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy. The method you need to override in this subclass depends on whether you want to edit basic metadata like the server's publisher or modify the descriptions of the server's functionality.

If you just want to change the server's basic metadata in the Capability Statement, for example, the server's name, you can modify the JSON template from which the Capability Statement is generated. This JSON template is located in the `GetCapabilityTemplate()` method of the endpoint's InteractionsStrategy class. To change the server's metadata strings:

1. Create a `GetCapabilityTemplate()` method in your subclass of HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy to override the method.
2. Copy the contents of **HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy.GetCapabilityTemplate()** into your subclass' `GetCapabilityTemplate()` method.
3. Edit the metadata strings and compile your subclass.
4. Use the Console Setup utility to update the Capability Statement. For details, see Command Line Options.

If you want to change the substance of the Capability Statement, for example, what interactions are supported for a resource, you need to override the InteractionsStrategy's `GetMetadataResource()` method. It is strongly recommend that your overriding method call `##super` to invoke **`HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy.GetMetadataResource()`**, and then post-process the Capability Statement that is returned by the method. You modify the returned Capability Statement as a dynamic object. For example, your subclass might look like:

Class Definition

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

Once you have overridden the method that generates the Capability Statement, be sure to update the Capability Statement using the Console Setup. For details, see Command Line Options.

18

Custom FHIR Operations

The FHIR server supports HL7® FHIR® operations that perform special functions based on requests from the FHIR client using an RPC-like approach rather than a RESTful one. These can be standard FHIR operations like `$everything` or custom ones. FHIR servers using or extending the Resource Repository already support certain standard FHIR operations (see Supported Interactions and Operations for a complete list).

The following is an overview of the process of adding FHIR operations to your FHIR server.

1. Subclass the FHIR server's architecture. For details, see Pre-Installation Subclassing.
2. Create a subclass of HS.FHIRServer.API.OperationHandler. If you are using the Resource Repository, subclass HS.FHIRServer.Storage.BuiltInOperations instead of HS.FHIRServer.API.OperationHandler so you do not lose the default operations like `$everything`. As a best practice, you might want to create a separate subclass for each operation, and then create a master class that inherits from all of them.
3. In your Interactions subclass, override the value of the `OperationHandlerClass` parameter to be the classname of the operation subclass that you just created.
4. Write a method for each operation in your operation handler subclass.
5. Add the operations to the CapabilityStatement resource.

The following sections provide more details on the last two steps of the process.

18.1 Writing Methods for Custom Operations

Operations supported by the FHIR server correspond directly to methods in the operation handler subclass. The names of these methods must conform to the following syntax:

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

Within this syntax, the variables are:

- *Scope* identifies the type of endpoint to which the FHIR client is appending the operation. Possible values are:
  - `System` — Identifies operations that are appended to a "base" FHIR endpoint (for example, http://fhirserver.org/fhir). These operations apply to the entire server.
  - `Type` — Identifies operations that are appended to a FHIR endpoint with a resource type (for example, http://fhirserver.org/fhir/Patient). These operations work with all instances of the specified resource type.

  - `Instance` — Identifies operations that are appended to a FHIR endpoint that points to a specific instance of a resource (for example, http://fhirserver.org/fhir/Patient/1). These operations work solely with a specific instance of a resource.

- *OperationName* is the `$` operation that the FHIR client appends to its call to the server.

The following table of examples shows the correlation between method names and the operations called by a FHIR client.

Method name	**REST client call to the operation**
`FHIRSystemOpMyoperation`	`http://fhirserver.org/fhir/$myoperation`
`FHIRTypeOpValidate`	`http://fhirserver.org/fhir/Observation/$validate`
`FHIRInstanceOpEverything`	`http://fhirserver.org/fhir/Patient/1/$everything`

If your operation contains a hyphen (`-`), just remove the hyphen from the method name. For example, if the system-wide operation is `$my-operation`, name the method `FHIRSystemOpMyoperation`.

The following is an example of the method signature for `$everything`:

Class Member

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

18.2 Adding the Operation to Capability Statement

The Capability Statement of the FHIR server should include all of the operations that the server supports. You have two choices for updating the Capability Statement with new operations:

- Manually add the operations to the Capability Statement. This approach has one drawback: the Capability Statement is sometimes regenerated, for example, when adding a new search parameter, and manual modifications are lost upon regeneration. For details on this process, see Manually Updating Capability Statement.
- Modify the `AddSupportedOperations()` method in your operation handler subclass to automatically add the new operation to the Capability Statement when it is regenerated. See the following section for details on this approach.

You can use the following two-step procedure to automatically add a new operation to the Capability Statement.

1. Add the operation to the `AddSupportedOperations()` method of the operation handler subclass. When the command-line utility generates the server's Capability Statement, it takes the supported operations from this method. As an example, the operation handling class for a server that supports the `$everything` operations would include a method that looked like:

   ### Class Member

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

   If the superclass of your operation handling class already includes some operations, be sure to call the `AddSupportedOperations()` method of that superclass within the `AddSupportedOperations()` of the subclass. For example, the method of the operation handling subclass might look like:

Class Member

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

If you created a subclass for each operation and a master class that inherits from all of them, make sure the master class calls the `AddSupportedOperations()` method of each operation's subclass.

2. Use the command-line utility to regenerate the Capability Statement:
    a. From the InterSystems Terminal, change to the FHIR server's namespace. For example:

       ObjectScript

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

    b. Run the installation and configuration utility:

       ObjectScript

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

    c. Choose the option `Update the CapabilityStatement Resource`.
    d. Select the endpoint you are configuring.
    e. Confirm your selection.

19

Bypassing the InterSystems FHIR Client

When using a server-side application to make HL7® FHIR® requests to the internal FHIR server, your application should usually use the standard FHIR client. For details about using these built-in classes, see FHIR Client.

However, you may want to use a custom ObjectScript class so you can interact with the repository without making a request through the Service. For example, you might want to perform a write operation even though the server restricts requests to read-only interactions. In this case, you can bypass the Service.

In other cases, you may want to use the same method that the FHIR client and REST handler use, but from a custom class. For details, see Direct Calls to `DispatchRequest`

Your ObjectScript application can also validate a resource.

19.1 Bypassing the Service

A server-side application can call the methods of an Interactions subclass directly instead of submitting programmatic requests via the Service. For example, an application could call the Interactions subclass' `Add()` method directly rather than sending a POST request to the Service. This is especially useful if the server-side application needs to perform actions that are prohibited by the Service. For example, if the server's metadata configures the endpoint as read-only, programmatic requests to the Service cannot create resources. However, using method calls to the Interactions subclass, a server-side application could update the storage strategy with resources, effectively bypassing the restrictions enforced by the Service.

Programmatic calls to methods of the Interactions class pass FHIR data as dynamic objects.

19.2 Direct Calls to `DispatchRequest`

An ObjectScript application can also act as a FHIR client by calling `DispatchRequest()` directly, which is the method used by the standard FHIR client and the internal FHIR server's REST handler.

19.2.1 GET Resources

Your ObjectScript application can use the server's Service to retrieve resources. For example, assuming `178.16.235.12` is the IP address of InterSystems server and `52783` is the superserver port, a REST call might be:

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

Using ObjectScript to access the same endpoint looks like:

ObjectScript

```
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)
```

In this example, the response is a data object (HS.FHIRServer.API.Data.Response) with the JSON response represented in a dynamic object.

Note: The first request to the server must instantiate the FHIR service by calling the `EnsureInstance()` method. It does not cause problems to make this call before every request, but it takes a miniscule amount of time to check whether the service has been modified.

19.2.2 POST Resources

You can also post data to the FHIR server programmatically. In the following example, suppose the application is creating a Patient resource that is described in a JSON object in the file MyPatient.json. The ObjectScript code might look like:

ObjectScript

```
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 = {}.%FromJSONFile("c:\resources\MyPatient.json")
do fhirService.DispatchRequest(request, .response)
```

In this example, the source of the JSON stored in the request could have come from a dynamic object in your application rather than an external file.

19.3 Handling FHIR Data as XML

When you use a REST client to perform CRUD operations on the FHIR server, the FHIR server automatically accepts or returns FHIR data as XML based on the incoming request. However, when you are performing CRUD operations programmatically from a custom ObjectScript class, all data going into the FHIR service must be in JSON format. Likewise, all data returned by the service is in JSON format. The FHIR server provides helper methods to convert XML to JSON and JSON to XML.

To send XML data into the FHIR service, put the XML into a stream object and send it to the **HS.FHIRServer.Service.StreamToJSON()** method, specifying that the format is XML. For example, the following code turns the XML payload into a JSON request that can be passed to the FHIR service:

ObjectScript

```
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")
```

To convert a JSON response from the FHIR service into XML, use the **HS.FHIRServer.Util.JSONToXML.JSONToXML()** method.

19.4 Handling FHIR Data as a Stream

The **HS.FHIRServer.Service.StreamToJSON()** method converts an XML or JSON stream into a JSON object so it can be passed to the FHIR service as part of a request. The FHIR service cannot handle a stream directly. The method accepts two arguments: the stream and the format of the data in the stream. For example, the following lines of code turn a JSON stream into a JSON object so it can be sent to the FHIR service:

ObjectScript

```
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")
```

For XML streams, simply pass `XML` as the second argument.

19.5 Validating FHIR Resources

Your ObjectScript application can programmatically validate a resource against the FHIR server’s metadata without using the FHIR `$validate` operation as long as the resource is represented as a dynamic object. For example, the following code validates a Patient resource against the server’s FHIR Release 4 metadata, which includes the schema for the Patient resource. When calling the `LoadSchema()` method, you can specify the common name of the FHIR version (for example, `R4` or `STU3`) or the name of the server’s base metadata (for example, `HL7v40` or `HL7v30`).

ObjectScript

```
// Put JSON representation of Patient resource into a dynamic object
set patient = ##class(%DynamicObject).%FromJSONFile("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)
```

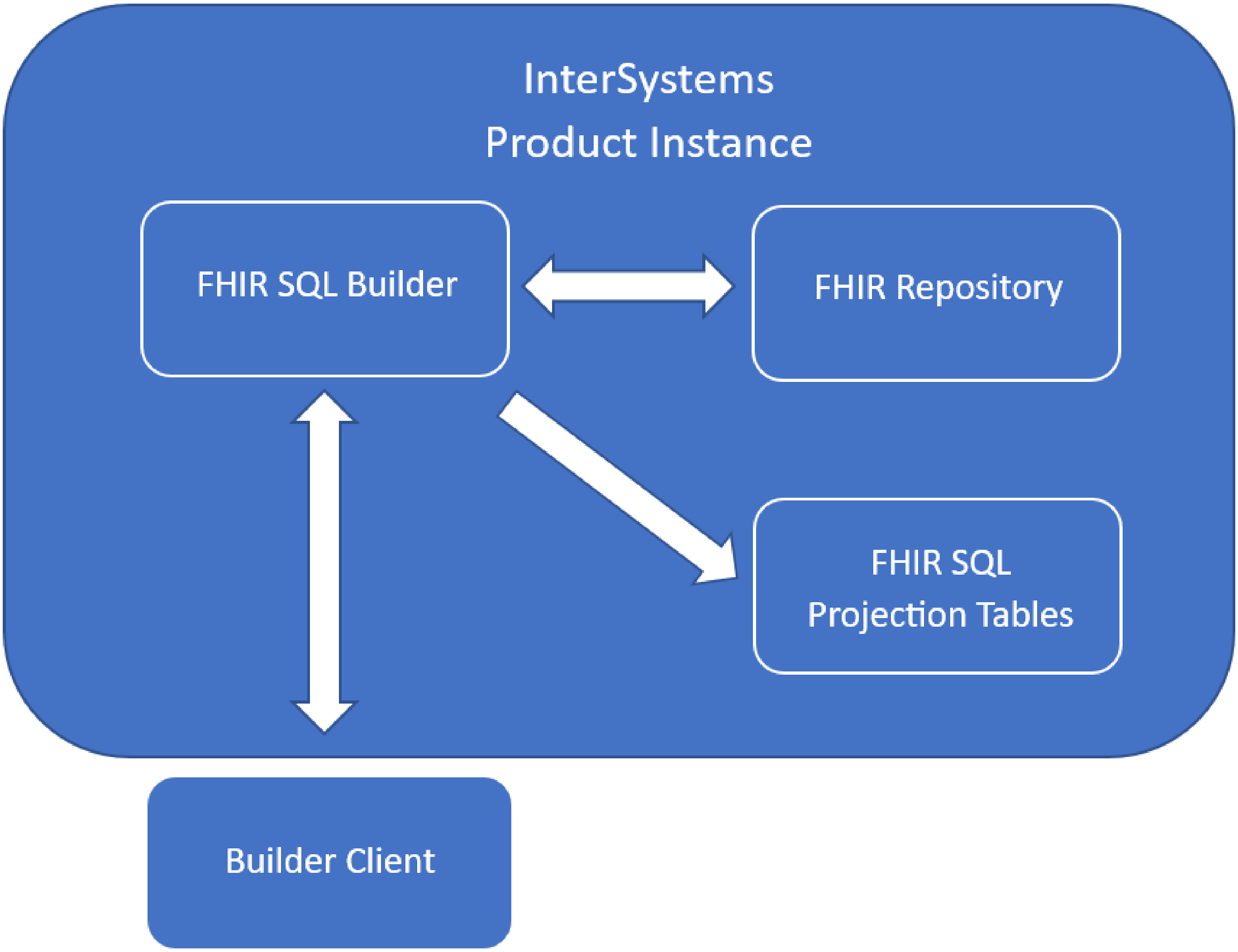
20

The FHIR SQL Builder

The FHIR SQL Builder, or *Builder*, is a sophisticated projection tool used to create custom SQL schemas using data in an HL7® FHIR® repository without moving the data to a separate SQL repository. The Builder is designed specifically to work with FHIR repositories and multi-model databases in InterSystems products.

The objective of the Builder is to enable data analysts and business intelligence developers to work with FHIR using familiar analytic tools, without having to learn a new query syntax. FHIR data is encoded in a complex directed graph and cannot be queried using standard SQL syntax. A graph-based query language, FHIRPath, is designed to query FHIR data, but it is non-relational. Enabling a data steward to create a customized SQL projection of their FHIR repository, using tables, columns, and indexes, the Builder makes it possible for data analysts to query FHIR data without the complexity of learning FHIRPath or the FHIR search syntax.

The following diagram shows the relationships between the Builder and other components in InterSystems products.



The Builder analyzes a FHIR repository to generate summary information, including the types of resources, elements, and values it contains, as well as the number of each type of resource. You decide which FHIR resources and elements to include in your custom SQL projection and how to map them.

20.1 Schema Generation Overview

To generate a schema the main actions are:

1. Analyze the FHIR repository.
2. Present the analysis to the user.
3. Use decisions made by the user to create the required tables.

The analysis process needs to examine enough of the repository to provide useful information, while limiting how much it examines to conserve time and resources. You can make decisions that influence this balance when configuring the analysis task. You can also configure the task to run at a time when there is less demand for computing resources.

20.2 Configuration

If your license key is in place before the installation of your InterSystems product finishes, the Web Applications that implement the Builder will already be enabled. Otherwise, you will need to manually enable them. On the instance that is running the Builder:

1. In the Management Portal, go to **System Administration** > **Security** > **Applications** > **Web Applications**.
2. On the **Web Applications** page, enable the applications /csp/fhirsql and /csp/fhirsql/api/ui by selecting the name of the application, which opens the **Edit Web Application** page.
3. On the **General** tab of the **Web Application** page, click the **Enable Application** check box and then click **Save**.

The FHIR SQL Builder is designed to work with FHIR repositories across different instances of InterSystems products. On each instance that runs a FHIR server, follow the steps above to enable /csp/fhirsql/api/repository. Only one instance should have the /csp/fhirsql/api/ui API enabled, but any number of instances can have the /csp/fhirsql/api/repository API enabled.

Once the applications have been properly enabled, you will be able to open the Builder at http://hostname:portnumber/csp/fhirsql/index.html#/, where hostname is the name of the host of your instance (this can be localhost) and portnumber is the port number of your instance.

Users that access the Builder must be assigned to one of the preconfigured FHIR SQL Builder roles:

- **`FSB_Analyst`** allows a user to access the FHIR SQL Builder application. It also allows a user to query a FHIR SQL projection table if the user has been added as a package user for the projection table.
- **`FSB_Data_Steward`** provides the privileges of **`FSB_Analyst`** and also allows a user to launch an analysis of a FHIR Repository, manage transform specifications for a FHIR SQL projection table, and create and manage FHIR SQL projection tables.
- **`FSB_Admin`** provides the privileges of **`FSB_Data_Steward`** and also allows a user to create a new FHIR Repository configuration for analysis.

Users who are not assigned one of these roles encounter a 403 error when they attempt to access the FHIR SQL Builder. For more information about assigning roles to users, see “Manage Roles” in Roles.

Starting the FHIR SQL Builder application brings you to the home page. The work area is divided into sections which let you configure the **Analyses**, **Transformation Specifications**, and **Projections** of your repository.

20.3 Analyze the FHIR Repository

The goal of analyzing your FHIR Repository is to summarize the available structural relationships, constituent elements, and embedded collections of the repository. It forms the basis for defining a transformation specification by outlining how various elements are related to one another and providing a range of values you can expect for a given element. The analysis task does not need to look at every record in the repository; is up to you to decide the size of the sample that you will analyze.

To configure an analysis, click **New** on the right side of the **Analyses** section to open the **New FHIR Analysis** dialog. Fill out this dialog as follows:

- **FHIR repository** – You can select a repository from the drop-down list or create a new repository by clicking **New** to the right of the field to open the **New FHIR Repository Configuration** dialog. Fill out this dialog as follows:
  - **Name** – Enter a name for the repository.

  - **Host** – Enter the DNS name or IP address of the host where the repository you wish to analyze is hosted. If the Builder UI and the repository are on the same instance, you can use `localhost`.
  - **Port** – Enter the port used to access the repository.
  - **SSL Configuration** (optional) – If you are using SSL, select the SSL configuration you want to use. For information about creating a new SSL configuration, refer to “Create or Edit a TLS Configuration” in About Configurations.
  - **Credentials** – Select credentials from the drop-down list or create new credentials by clicking the **New** button to the right of the field to open the **New Credentials** dialog. Fill out this dialog as follows:
    - **Name** – Enter a name for this credentials object.
    - **Username** – Enter the username. This must be the name of a User on the current instance. To see a full list of the available names, open the Management Portal and navigate to **System Administration** > **Security Management** > **Users**.
    - **Password** – Enter the password.
    - Click **Save** to return to the **New FHIR Repository Configuration** dialog.

    Important: The user account you specify with a FHIR Repository must also be assigned to the **`FSB_Admin`** role.

  - **FHIR Repository URL** – Select a repository from the drop-down list. When the values entered for **Name**, **Host**, **Port**, and **Credentials** establish a valid connection, this field provides a list of available FHIR repositories.
  - Click **Save** to return to the **New FHIR Analysis** dialog.

- **Selectivity Percentage** – Limits analysis to a percentage of the FHIR repository. If used, **Maximum Records** cannot be used.
- **Maximum Records** – Limits analysis to a maximum number of records in the FHIR repository. If used, **Selectivity Percentage** cannot be used.
- **Defer Start of Task** – Select this check box to run the analysis task at a later point in time. If selected, enter a **Start Date** and a **Start Time**.
- Click **Launch Analysis Task** to start analyzing the FHIR repository.

You will see the newly started analysis on the list of analyses on the Builder home page. Columns provide the following information:

- **FHIR Repository** – The name you provided when configuring the analysis.
- **Start Time** – The date and time the analysis started.
- **Last Modified** – The date and time you last modified the analysis.
- **Status** – The possible values are: **Running**, **Stopping**, **Stopped**, **Completed**, and **Errored**.
- **Total Resources** – The number of FHIR records analyzed.
- **Percent Complete** – The amount of the analysis completed as a percentage of the anticipated total.
- **Actions** – Provides buttons you can use to control the progress of the analysis.
  - ▷ **Resume** – Resumes running a paused analysis.
  - ⏸ **Pause** – Pauses a running analysis.

  - 🗑 **Delete** – Deletes the analysis of the repository. An analysis cannot be deleted if a Transformation Specification depends on it.

20.4 Creating a New Transformation Specification

Once the FHIR Repository analysis is complete, you can use the resulting information to create a transformation specification. A transformation specification structures the table schemas it generates and determines which resources and elements should be included in the projection. Transformation specifications can also be exported, imported, and copied; see Exporting, Importing, and Copying a Transformation Specification for more information.

To create a new Transformation Specification, click the **New** button on the right side of the **Transformation Specifications** section to open the **New Transformation Specification** dialog. Fill out this dialog as follows:

- **Name** – Provide a unique name for this specification.
- **Analysis** – Select a FHIR Repository analysis from the drop-down list. Analyses are identified by the repository name and the date the analysis was created.
- **Description (Optional)** — Provide a brief description of the transformation specification.
- Click **Create Transformation Specification** to open the **Edit Transformation Specification** page.

The page opens listing resources and elements in the FHIR repository and their counts. Selecting a resource, such as **Patient**, shows the elements in that resource.

The structure presented by the analysis reflects the often deeply nested structure of the FHIR repository. Elements are:

- primitives, such as string, boolean, date, or number
- objects, which contain properties that are primitives, objects, or collections
- collections, which contain primitives, objects, or collections

Resources are structured like trees, and selecting a node (which represents either an object or a collection in the repository) opens the properties it contains below it. Leafs (which represent primitive elements) across the repository may have identical names, like “code.” As such, it is best to refer to a primitive element in the tree by the full path through the nodes from the parent resource to the child leaf (for example, “AllergyIntolerance.code.coding.code” as opposed to “Patient.code.coding.code”).

20.4.1 Adding a Primitive Element to the Schema

Select a primitive element to open a panel that allows you to add the element to the schema. For primitives, such as strings or numbers, you can click **Show Histogram** to view a histogram of the unique values in the repository. You can edit the name of the column that will appear in the table by editing the **Column name** field. Select the check box label **Index** if you would like to add an index on this column in the table. For information regarding which elements you may want to add an index to, see “What to Index” in Optimizing Query Performance.

Finally, click **Add To Projection** to add the element to the Projection.

The **Currently Selected Items** table on the edit page shows the current state of the schema. It has columns the following columns:

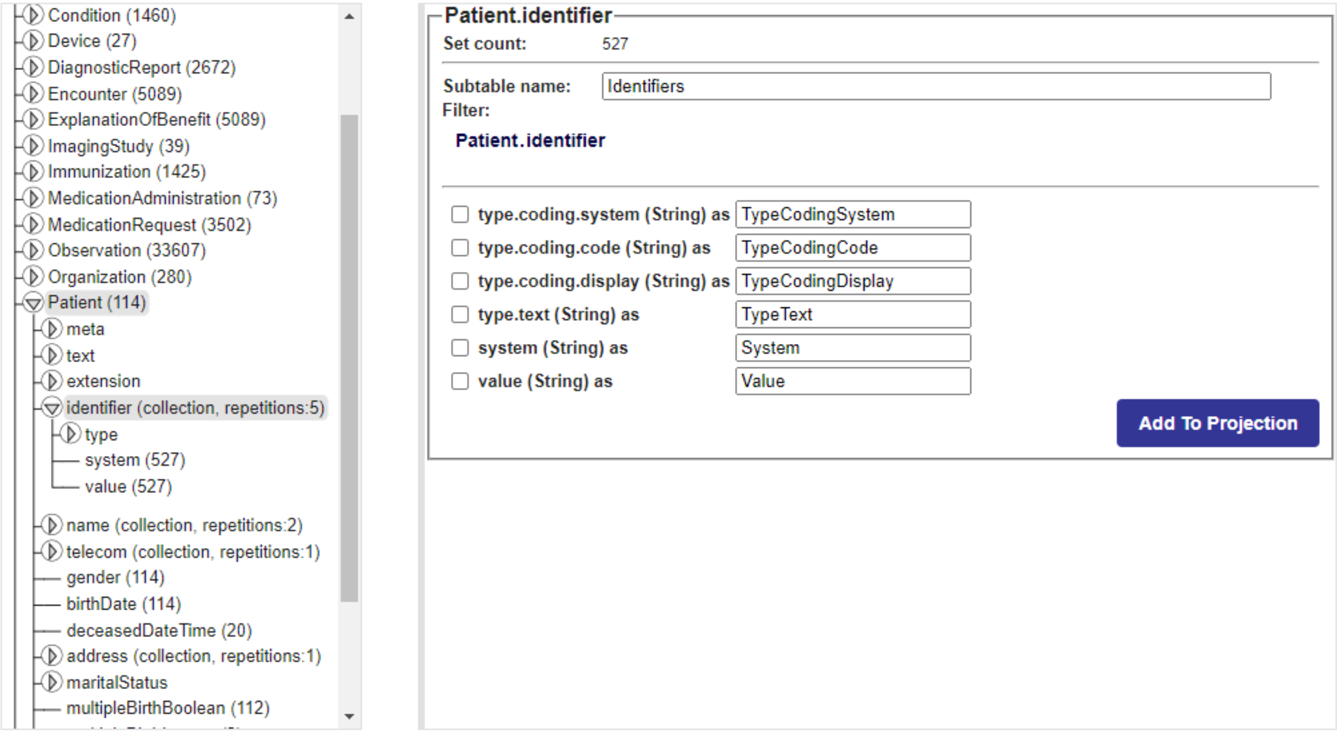
- **Table** – The name of the table the added column is for. You can add elements from multiple different resources to a schema in a Transformation Specification; theses elements will appear in the tables as specified by the values in this column.

- **Column** – The name of the column that will appear in the table.
- **Type** – The datatype of the contents of the column.
- **Index** – Indicated whether an index will be created for the column.
- **Actions** – Provides buttons you can use to edit or delete the specific element.
  - ✎ **Edit** – Opens the panel that you used to add the element to the schema, enabling you to edit your preferences for this column, including setting filters, in the schema.
  - 🗑 **Delete** – Deletes the column specified by the from the schema.

You can add as many elements to the schema as needed. Click **Done** to return to the home page, where you will find the new specification listed.

20.4.2 Adding Data from Collections to the Schema

Some elements have collections associated with them. Selecting a collection will bring up a selection panel where all the elements at the bottom node of the collection are listed in rows. An example, using the **Identifier** collection of a **Patient**, is below. Note that this particular collection has both a object and primitive elements within it.



Patient.identifier

Set count: 527

Subtable name: Identifiers

Filter:

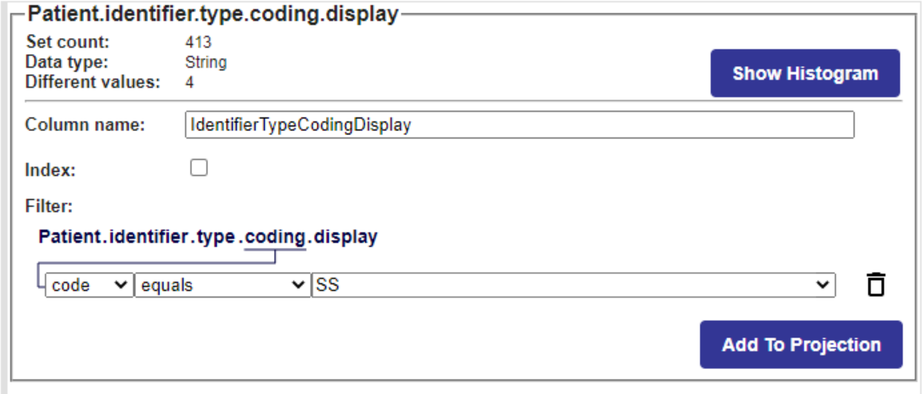
Patient.identifier

☐ **type.coding.system (String) as** TypeCodingSystem
☐ **type.coding.code (String) as** TypeCodingCode
☐ **type.coding.display (String) as** TypeCodingDisplay
☐ **type.text (String) as** TypeText
☐ **system (String) as** System
☐ **value (String) as** Value

Click the check boxes next to the primitive elements you would like to add to the projection. When adding data from a collection in this manner, the Builder will automatically create a subtable for this data and it will not be included in the table of the parent resource. The name of the table is identified by the **Subtable name** field, which you can edit. The rows of this table correspond to individual entries of the collection and the columns are the elements of the collection you have specified, plus some special columns that help identify what resource they came from and where in the collection they appear.

However, it is sometimes desirable to have data from a collection stored within a table representing the resource it came from and not in a subtable. To do so, you should navigate to a primitive element you would like to add by clicking through the collection and selecting **Add to Projection**. However, this action will automatically populate the column with the value of the first element of the collection. As this element is likely arbitrary, it is not recommended to leave the column as such. Instead, you should use the **Filter** option.

The **Filter** option allows you to choose specific data from a collection to include in the projection. From any leaf in the resource's tree, you are able to click on a preceding node in the full path, except for the initial resource you are drawing from (for example, the **Patient** resource name) or the leaf itself (for example, the terminal **display** node). In the example below, the filter has been created by clicking on **coding**; it is usually best practice to set a filter on the deepest node of the resource tree. You can set conditions to determine which fields to show in the projection. The filter is set to include the **display** element of the item in the collection where the **code** element equals "SS."



Patient.identifier.type.coding.display

Set count: 413
Data type: String
Different values: 4

Column name: IdentifierTypeCodingDisplay

Index: ☐

Filter:
Patient.identifier.type.coding.display
code equals SS

The table projected by this transformation will show the **display** element of the **coding** objects that had a **code** of "SS." However, if the filter was set on **code** elements equal to "MR," the table will show the **display** element of the **coding** objects that had a **code** of "SS."

You can use the filter option with different operations; it is not limited to use with equals. Additional operations include **greater than**, **less than or equal to**, **greater than or equal to**, **exists**, **like regex**, **resolves to type**, and more. You should configure a filter that fits your needs.

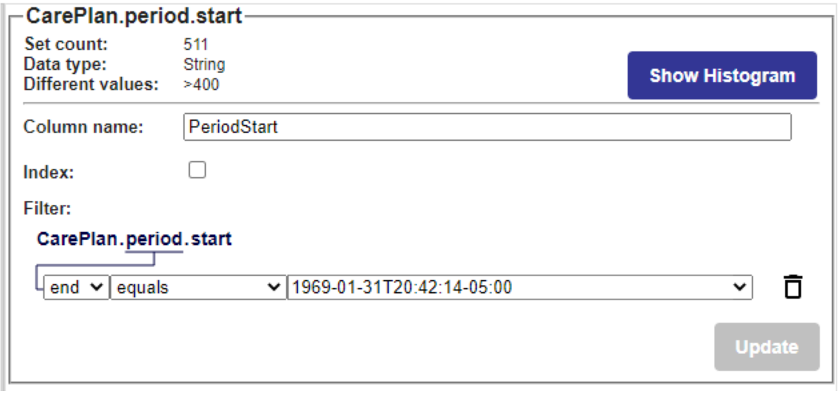
You can also add multiple filters for a particular element to finely tune which data appears in your table; however, multiple filters cannot be set on the same tier of a path. Additionally, you cannot set a filter on elements of a subtable; instead, you must set a filter on the subtable itself, which will then include only the elements that meet the filtered condition.

Note that in some cases, a collection may not include an object that matches the filter you have specified. In these cases, the rows that do not contain such elements will not have any data in that column.

20.4.3 Adding an Object to the Schema

To add an element of an object associated with the resource, click through the object's elements until you get to a primitive element as described in "Adding a Primitive Element."

You are allowed to set a filter on objects, as you can with collections. In this case, the filter is most useful as a method to remove any values from the table that do not match a certain criteria. In the example below, the filter is used to show the **start** element of a CarePlan when the **end** element is January 31, 1969. In the resultant **CarePlan** table, rows that did not end on that date will not have any data in the **start** column.



CarePlan.period.start

Set count: 511

Data type: String

Different values: >400

Column name: PeriodStart

Index: ☐

Filter:

CarePlan.period.start

end | equals | 1969-01-31T20:42:14-05:00

20.4.4 Viewing Transformation Specifications

On the main FHIR SQL Builder page, you will see your Transformation Specifications listed under the header. Columns in the listing provide the following information:

- **Name** – The name of this specification.
- **Analysis** – The analysis used to create this specification.
- **Description** – The description of the transformation specification as defined when it was initially created.
- **Last Modified** – The date and time the specification was last modified.
- **Actions** – Provides buttons you can use to manage the specification.
  - ✎ **Edit** – Opens the **Edit Transformation Specification** page.
  - ⧉ **Copy** – Opens the **Copy Transformation Specification** dialog box to create a new specification identical to this one. For more information, see “Copying a Transformation Specification.”
  - ⤓ **Export** – Downloads a JSON file of the transformation specification that can be shared and imported. For more information, see “Exporting a Transformation Specification.”
  - 🗑 **Delete** – Deletes the transformation specification. A transformation specification cannot be deleted if a projection depends on it.

20.5 Exporting, Importing, and Copying a Transformation Specification

Transformation specifications can be exported, imported, and copied. Imported and copy transformation specifications can be further edited to further customize them.

20.5.1 Exporting a Transformation Specification

Transformation specifications can be exported to allow you to share these specifications with other users or instances, where they can be imported. To export a transformation specification, click the **Export** icon on the row of the **Transformation Specifications** table on the home page. A JSON file will be downloaded to you local file system. This file can be shared and imported into other systems.

20.5.2 Importing a Transformation Specification

A previously exported transformation specification can be re-imported on the same system or imported on another system.

To import a transformation specification, click on the **Import** button on the **Transformation Specifications** table on the home page. You will then be prompted to select the JSON transformation specification file that you want to import from your file system. When a valid file is selected, the **Import Transformation Specification** dialog will open with the following fields:

- **Name** – The name of the imported transformation specification. This will be pre-populated from the JSON file, but can be edited here.
- **Analysis** – The completed FHIR repository analysis from which to build the transformation specification. The analysis should have resources that correspond with the resources in the transformation specification.
- **Description** – A brief description to help distinguish between specifications. This will be pre-populated from the JSON file, but can be edited here.

Click the **Import** button to complete the import process. The new specification will appear in the **Transformation Specifications** table.

20.5.3 Copying a Transformation Specification

Any existing transformation specification can be copied to create a new specification with a new name and description. This new specification can be further modified and used, independent of its original specification.

To copy a transformation specification, click the **Export** icon on the row of the **Transformation Specifications** table on the home page. Fill out the dialog box as follows:

- **Name** – The name of this new specification. This will come pre-populated as “Copy of `transformation`,” where `transformation` is the name of the transformation specification that is being copied.
- **Description** – A brief description to help distinguish between specifications. Like the **Name** field, this will come pre-populated as “Copy of `transformation`,” where `transformation` is the name of the transformation specification that is being copied.

Click the **Copy** button to complete the import process. The new specification will appear in the **Transformation Specifications** table.

20.6 Create the Projection

The final step is to create a data **Projection**. Click **New** on the right side of the **Projections** section to open the **New Projection** dialog. Fill out this dialog as follows:

- **FHIR Repository** – The repository to use for this projection. This drop-down menu includes the names of the repositories added when creating a new **Analysis**.
- **Transformation Specification** – The specification used to create this projection.

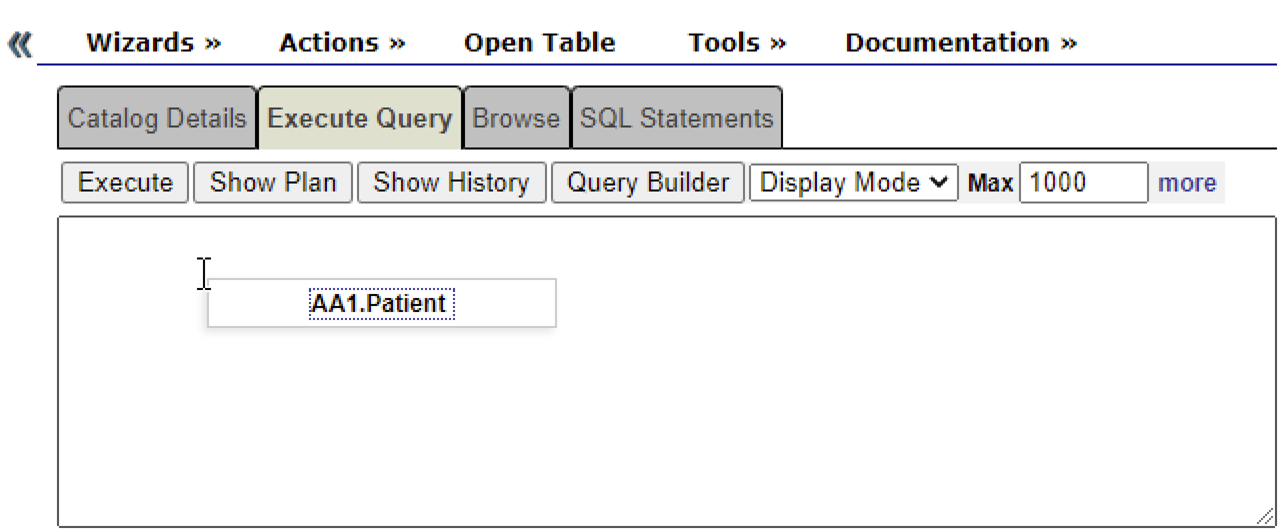
- **Package Name** – A name for the package the SQL tables will be put in.
- Click **Launch Projection** to create the projection.

Once you have created the projections, columns provide the following information:

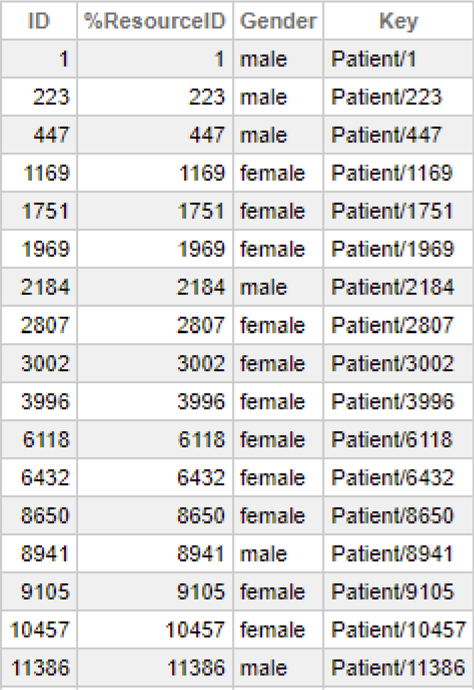
- **FHIR Repository** – The name of the FHIR repository on which the projection is based.
- **Transformation Specification** – The Transformation Specification that the Projection is based on.
- **Namespace** – The namespace that the Projection will be stored in.
- **Package Name** – The package that the Projection will be stored in.
- **Actions** – Provides buttons you can use to manage the specification.
  - 🔗 **Link** – Opens the **System Explorer** > **SQL** page of the **Management Portal**.
  - ⟳ **Update** – Updates the **Projection** to reflect changes in the underlying **Transformation Specification**.
  - 🗑 **Delete** – Deletes the **Projection**.

Clicking the **Link** button opens a page in the **Management Portal**. Enter the package name you specified when creating the **Projection** into the **Filter** box on the upper left side of the screen. The projections you have created will appear under **Tables** on the left.

Drag the table to the **Execute Query** tab, as shown:



Then execute the query. The table will appear below.



ID	%ResourceID	Gender	Key
1	1	male	Patient/1
223	223	male	Patient/223
447	447	male	Patient/447
1169	1169	female	Patient/1169
1751	1751	female	Patient/1751
1969	1969	female	Patient/1969
2184	2184	male	Patient/2184
2807	2807	female	Patient/2807
3002	3002	female	Patient/3002
3996	3996	female	Patient/3996
6118	6118	female	Patient/6118
6432	6432	female	Patient/6432
8650	8650	female	Patient/8650
8941	8941	male	Patient/8941
9105	9105	female	Patient/9105
10457	10457	female	Patient/10457
11386	11386	male	Patient/11386

Note: A system task called **IndicesTask** will be created on the FHIR Repository if the system were to shut down while indexes were being built. This task will resume building indexes when the system starts up again.

21

Bulk FHIR Coordinator

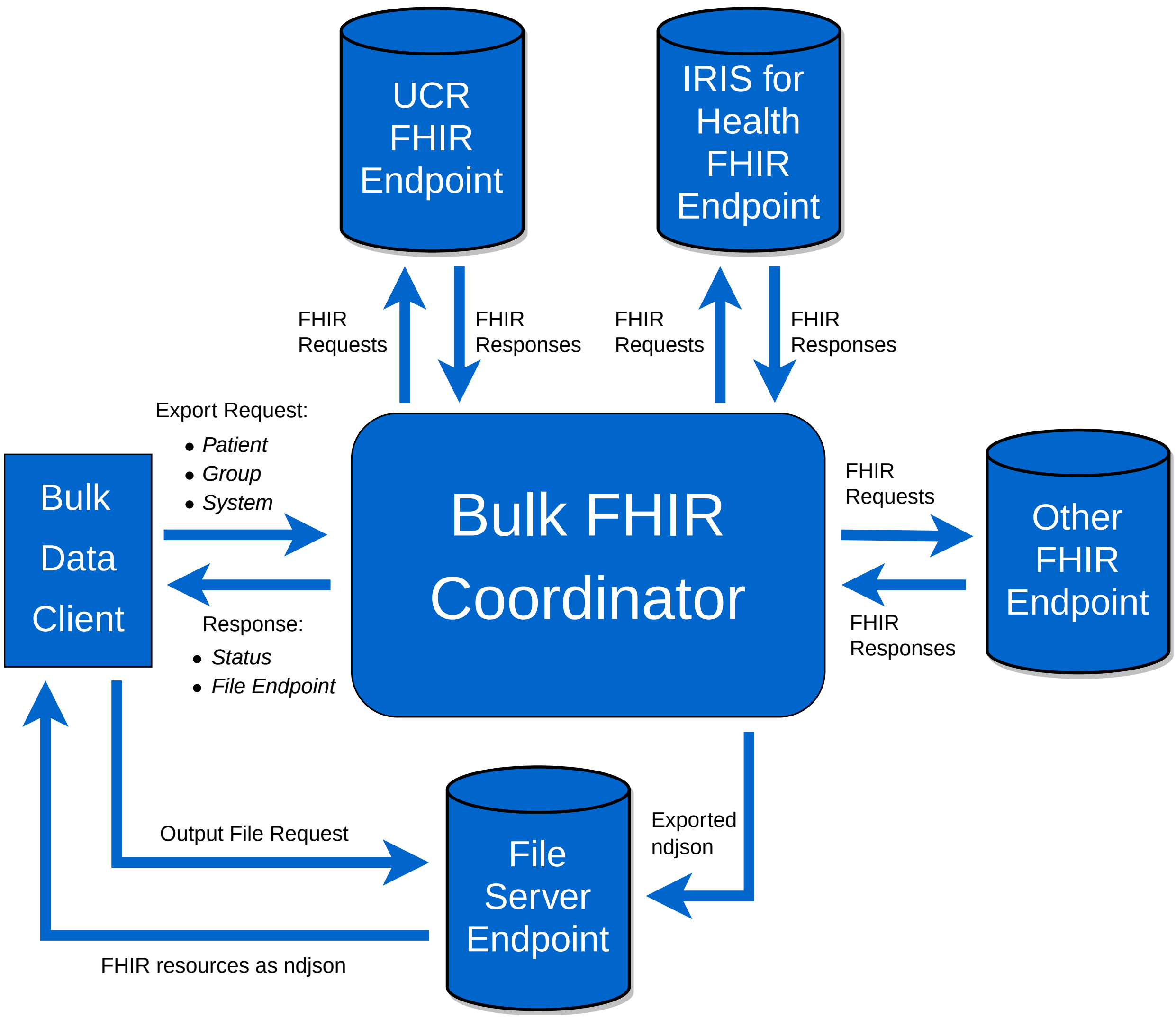
While a typical HL7® FHIR® interaction seeks specific information about a particular patient, a FHIR bulk data interaction extracts large quantities of data across patients from a FHIR resource server. Typical uses for bulk FHIR include identifying study cohorts, population health, or transferring data from one EHR to another.

21.1 Introduction to the Bulk FHIR Coordinator

In order to simplify the FHIR bulk data interaction for clients and to not overwhelm a FHIR server with bulk data requests, the InterSystems *Bulk FHIR Coordinator*(BFC) mediates the interaction between a *bulk data client* and a *FHIR resource server endpoint* for bulk data requests. The Bulk FHIR Coordinator can facilitate bulk FHIR export for FHIR resource servers that *do not natively support the bulk data interaction*.

When the Bulk FHIR Coordinator requests and receives FHIR resources from a FHIR endpoint on behalf of a client, it is called an *export*.

The diagram below illustrates how the Bulk FHIR Coordinator mediates the interaction between a client and FHIR endpoints.



You can interact with the Bulk FHIR Coordinator either by using the BFC *home page* or through a bulk data *REST client*:

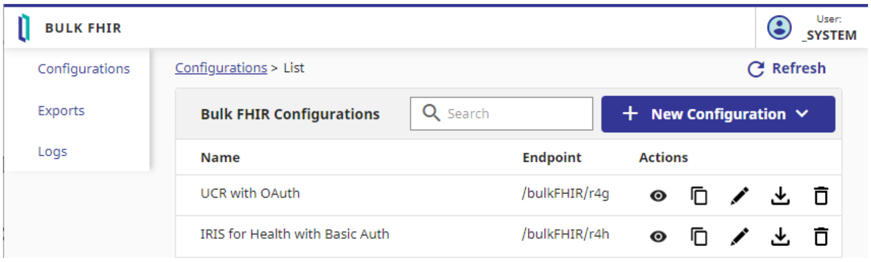
- On the BFC home page, you can enter a set of configurations. Each bulk FHIR configuration identifies a FHIR resource server endpoint, and defines the authorization type, file location, and other parameters to be used in the bulk data interaction. The FHIR endpoint may be InterSystems IRIS for Health, HealthShare Health Connect, HealthShare Unified Care Record, or *any other system* that supports returning all resources and, for patient and group exports, supports the Patient/$everything operation.

  From the home page you can also initiate exports, view export status, download exported files, and view the export logs.

- Using a REST client you can perform each of the requests described in the FHIR Bulk Data Export specification. Each bulk FHIR configuration provides two endpoints:
  - *bulk FHIR endpoint* — supports export, status, and delete requests
  - *file storage endpoint* — supports file download requests

21.2 The Bulk FHIR Coordinator Home Page

To work with bulk FHIR configurations, navigate to **Home** > **Health** > *foundationNamespace* > **Bulk FHIR Coordinator**:



Name	Endpoint	Actions
UCR with OAuth	/bulkFHIR/r4g	👁 ⧉ ✎ ⤓ 🗑
IRIS for Health with Basic Auth	/bulkFHIR/r4h	👁 ⧉ ✎ ⤓ 🗑

Each configuration has a name and a unique endpoint. Use the icons on the **Configurations** > **List** page to perform the following actions:

Icon	Action	Notes
	New Configuration	Allows you to create a new configuration either by opening the Edit page or by importing a JSON file.
👁	View Exports	Opens the Export view page for the configuration where you can view in-progress and completed bulk FHIR exports or request a new export.
⧉	Copy	Opens the Edit page with an identical configuration, except that the Name and Endpoint have `_copy` appended. Step through the configuration pages using the **Next** button and make any necessary adjustments. Click **Configure** on the **Review** page to save the new configuration.
✎	Edit	Opens the Edit page for the configuration. Step through the configuration pages using the **Next** button and make any necessary adjustments. Click **Configure** on the **Review** page to save the changes.
⤓	Download	Creates a JSON-formatted record of your configuration. The name of the file is the same as the name of your configuration, with any special characters replaced by an underscore. To import a JSON-formatted configuration, click the **New Configuration** button, select **Import JSON**, and then locate the file using the browse control.
🗑	Delete	Enter the endpoint URL as indicated in the text box to confirm and then click the **Delete** button.

Note: Which actions appear for each configuration depend on the user's roles.

21.3 Creating or Editing a Bulk FHIR Configuration

To create or edit a bulk FHIR configuration:

1. Log in to InterSystems IRIS for Health as a user with administrative privileges.
2. Navigate to **Home** > **Health** > *foundationNamespace* > **Bulk FHIR Coordinator**.

3. On the **Bulk FHIR Configurations** page, invoke the **Create/Edit** workflow by clicking either ✎ or ⧉ for an existing configuration or by clicking **New Configuration** followed by **Create New**.

The bulk FHIR configuration **Create/Edit** workflow consists of five separate pages:

1. Configuration Settings
2. Authorization Types
3. Fetch
4. Storage Location
5. Review

Enter values on each page in the **Create/Edit** workflow and then click **Next** to move to the next page. On the final review page, click **Configure** to save the bulk FHIR configuration. The sections that follow explain the various settings.

21.3.1 Bulk FHIR Create/Edit Workflow: Configuring Settings

The settings on the **Configuration Settings** page of the **Create/Edit** workflow for bulk FHIR configurations are:

Name

Enter a unique name for your configuration. If you include some of the key parameters in the name, it will help you distinguish among several configurations. A name is required.

Auto-start Exports

Select this option if export jobs should start as soon as a request is received. Deselect this option if initiating an export requires manual approval before starting. Selected by default.

Authorized Users

Optionally enter a comma-delimited list of non-administrative users who are permitted to perform exports using this configuration. A user who holds *only* the %HS_BFC_Exporter role must be listed as an authorized user in order to perform exports. This includes the dummy users associated with OAuth clients for the purpose of mapping roles.

Note: A user who holds only the %HS_BFC_Exporter role will not be able to access Bulk FHIR Coordinator home page UI unless they are either provided a direct link to the page, or their startup namespace is configured to be the BFC foundation namespace.

BFC Endpoint

Enter the URL for this bulk FHIR endpoint. This value is required and must be unique among configurations. It should be a relative value, for example `/bulkFHIR/r4a`. When you save this configuration, a web app will be created using this URL that will serve as the REST endpoint.

Core FHIR Package

This is a read-only field that is derived from the **Fetch Endpoint URL** for the configuration when the configuration is saved. An example value is `hl7.fhir.r4.core@4.0.1`

Permitted Exports

Select which type of exports are allowed:

- *Patient* — the set of FHIR resources pertaining to all patients.

- *Group* — the set of FHIR resources pertaining to all members of the specified Group resource. With the HS.BulkFHIR.Fetch.ODS.Adapter fetch adapter, a Group resource is understood to be a Unified Care Record cohort.
- *System* — all FHIR resources, whether or not they are associated with a patient. This supports use cases like backing up a server, or exporting terminology data by restricting the resources returned using the `_type` parameter.

  Note: System export is not supported with the HS.BulkFHIR.Fetch.ODS.Adapter fetch adapter.

Expire After

The time in minutes after which stored ndjson files expire. Defaults to 1440 minutes (one day). Any ndjson files that have expired are deleted by the Bulk FHIR expiration task that runs hourly by default.

Max file size

The maximum size of each ndjson file in bytes. When this limit is reached, the open ndjson file in the export is saved and a new file is started. Defaults to one million bytes.

Flush Interval

The flush interval in minutes. When the interval is reached, the open ndjson file in the export is saved and a new ndjson file is started. Defaults to 60 minutes.

Working Directory

Directory where temporary files are stored before they are passed to the storage file adapter.

21.3.2 Bulk FHIR Create/Edit Workflow: Configuring Authorization

Which settings appear on the **Authorization Types** page of the **Create/Edit** workflow depend upon the Auth Adapter that you select:

Auth Adapter

This field is required. This defines the type of auth used between the *bulk data client* and the *Bulk FHR Coordinator*. Select either HS.BulkFHIR.Auth.BasicAuth.Adapter or HS.BulkFHIR.Auth.OAuth.Adapter. InterSystems IRIS for Health includes a utility that will configure an OAuth 2.0 server for you if you do not already have one.

For the `BasicAuth` adapter, no additional settings are required. For the `OAuth` adapter, also enter the settings below:

Issuer URL

The URL of an existing OAuth 2.0 server. When using the `OAuth` adapter this field is required. The OAuth server may be on the IRIS for Health instance or elsewhere. IRIS for Health includes a utility that will create an OAuth 2.0 server for you if you do not already have one.

Example value: `https://example.org/oauth2`.

BFC Client Name

When using the `OAuth` adapter this field is required. Enter the **Application Name** of the *OAuth client configuration* for the Bulk FHIR Coordinator *as an OAuth resource server.*

When a REST client presents a token to the Bulk FHIR Coordinator endpoint, the **BFC Client** validates the token with the OAuth server.

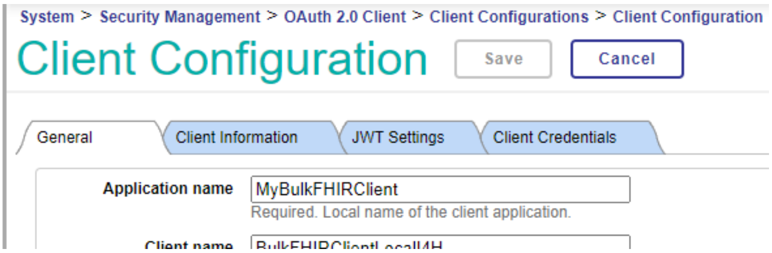
- If the **BFC Client** configuration is not already defined then it will be created automatically when you save this bulk FHIR configuration, as long as your OAuth server supports dynamic client registration.
- If your OAuth server does not support dynamic client registration, then you must:
  1. Request that OAuth server administrator provision a client account on the OAuth server for the BFC resource server.
  2. Manually add an OAuth client configuration on the Bulk FHIR Coordinator instance using the value in **BFC Client Name** by navigating to **Home** > **System Administration** > **Security** > **OAuth 2.0** > **Client** > *issuerEndpoint* > **Client Configurations**.

Clients

A list of OAuth clients approved for performing bulk FHIR exports from this BFC endpoint.

Your OAuth server should have a set of *OAuth client descriptions* defined that match the *OAuth client configurations* in **Home** > **System Administration** > **Security** > **OAuth 2.0** > **Client** > *issuerEndpoint* > **Client Configurations**.

Each OAuth client configuration has an **Application Name** (indicated on the **General** tab).


System > Security Management > OAuth 2.0 Client > Client Configurations > Client Configuration
Client Configuration
Save
Cancel
General
Client Information
JWT Settings
Client Credentials

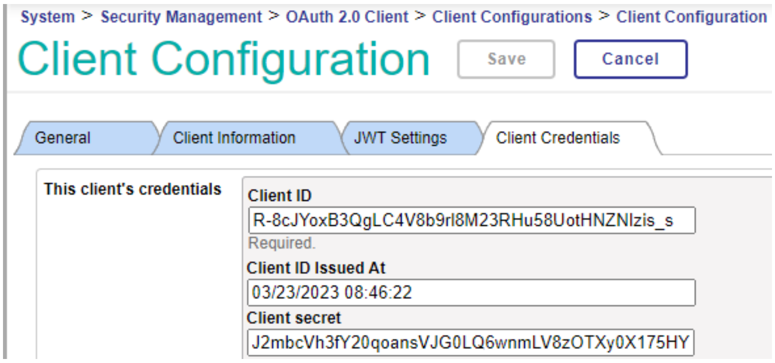
Application name: MyBulkFHIRClient
Required. Local name of the client application.

To indicate which OAuth clients may use this BFC configuration, enter in the **Clients** field a comma-separated list of the form *name*:*authentication_method* where:

- *name* is the **Application Name** in the OAuth client configuration.
- *authentication_method* identifies which Open ID Connect workflow this client will use to authenticate with the OAuth server. The value for *authentication_method* must be either `client_secret_post` or `private_key_jwt`.

Note: If the client configurations you enter do not already exist, they will be created when you save this bulk FHIR configuration, as long as your OAuth server supports dynamic client registration. Alternatively, you can create these clients manually.

Each OAuth client configuration also has a **Client ID** and **Client Secret** (indicated on the **Client Credentials** tab).


System > Security Management > OAuth 2.0 Client > Client Configurations > Client Configuration
Client Configuration
Save
Cancel
General
Client Information
JWT Settings
Client Credentials

This client's credentials

Client ID: R-8cJYoxB3QgLC4V8b9rl8M23RHu58UotHNZNIzis_s
Required.

Client ID Issued At: 03/23/2023 08:46:22

Client secret: J2mbcVh3fY20qoansVJG0LQ6wnmLV8zOTXy0X175HY

When a bulk data REST client sends a request to the BFC endpoint, the access token that it presents includes a client_id and client_secret. The access token's client_id is validated against the **Client ID**s of the OAuth client configurations listed in the **Clients** field of the BFC configuration, and the client_secret in the access token is validated against the **Client Secret** in the OAuth client configuration.

Important: Each OAuth export client must have *both* an OAuth client configuration *and* a dummy InterSystems IRIS user of the same name. The dummy user serves to map the appropriate roles to the OAuth client. See Setting up Users for detailed instructions.

The dummy user is used solely as a means to map user roles to an OAuth client, which enables a REST export client to engage in bulk FHIR interactions with this BFC endpoint. This user is typically created as "Not Enabled", which prevents an actual user from logging in with those credentials.

21.3.3 Bulk FHIR Create/Edit Worklow: Configuring Fetch

On the **Fetch** page of the **Create/Edit** workflow:

1. Select a fetch adapter
2. Configure the fetch adapter
3. Configure the authorization settings

21.3.3.1 Configuring Fetch: Selecting an Adapter

Fetch Adapter

This field is required. Select either HS.BulkFHIR.Fetch.PureFHIR.Adapter or HS.BulkFHIR.Fetch.ODS.Adapter. The ODS adapter is specific to the Unified Care Record ODS.

Note: *System* export is not supported with HS.BulkFHIR.Fetch.ODS.Adapter.

21.3.3.2 Configuring Fetch: Configuring the Adapter

All of the settings under the **Adapter Configuration** heading appear for both the PureFHIR and ODS fetch adapters with the exception of the Registry Webservice settings that appear only for the ODS fetch adapter.

Endpoint URL

The full URL of the FHIR endpoint, for example, `https://example.org/fhir/r4`. This field is required.

SSL Configuration

The InterSystems IRIS SSL/TLS client configuration that describes how to communicate with the FHIR endpoint when using HTTPS.

Resource Types

The comma-delimited default list of FHIR resource types that should be included in an export operation for this configuration. This list can be overridden by a client using the `_type` query parameter in the bulk data request. If this field is left blank, then all resource types are included by default.

Max Requests Per Second

Maximum number of HTTP(S) requests to make to the FHIR endpoint each second. This number will be shared across all active export operations for the configuration, and may be used to limit the load imposed by the Bulk FHIR Coordinator on the FHIR endpoint. The default value is 10 requests per second.

HTTP Timeout

Timeout value in seconds for each HTTP(S) request to the FHIR endpoint when fetching data. If your export is for a very large population and the FHIR endpoint has a large page size, you may wish to extend this timeout if you get timeout errors while fetching. The default value is 180 seconds.

Worker Jobs

Number of background worker jobs assigned to do the fetch processing. The default value is 4 jobs.

Registry Webservice Credential Id

Required when using HS.BulkFHIR.Fetch.ODS.Adapter. The interoperability credential to use when calling the Hub web service at the Unified Care Record Registry.

This credential should match the *Username Token Profile* setting in the UCR service registry entry for the Hub web service. You can identify this service registry entry as it will refer to *baseURL*`/services/HS.Hub.HSWS.WebServices.cls`. The service registry entry is typically named HSREGISTRY. A typical value of the *Username Token Profile* setting in the HSREGISTRY service registry entry is `HS_Services`.

The username and password in the credential will be used when invoking the Hub web service at runtime.

Registry Webservice Endpoint URL

Required when using HS.BulkFHIR.Fetch.ODS.Adapter. The Hub web service URL at the Unified Care Record Registry. Typically:

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

21.3.3.3 Configuring Fetch: Configuring Authorization

Once the fetch adapter is configured, specify the authorization settings for the BFC fetch interactions with the FHIR endpoint.

Authorization Type

This defines the type of auth used between the Bulk FHIR Coordinator and the FHIR endpoint when performing an export. Select either `HTTP` for basic auth, `X-API` for X-API-Key header auth, or `OAuth` for OAuth 2.0.

Note:

- The HTTP Credential Id setting appears only if you select basic auth.
- The X-API Key Credential setting appears only if you select X-API-Key header auth.
- The remaining settings appear only if you select OAuth 2.0.

HTTP Credential Id

For basic auth only, the interoperability credential to use when communicating with the FHIR endpoint.

X-API Key Credential

For X-API-Key header auth only, the interoperability credential to use when communicating with the FHIR endpoint. The X-API-Key header will contain the password from the credential when the BFC sends an HTTP request to the FHIR endpoint.

OAuth Issuer URL

Issuer URL of the FHIR endpoint's OAuth server.

If this OAuth server supports discovery, a server description will be created for it when you save this BFC configuration.

Client Name

The **Application Name** of the OAuth client configuration that the Bulk FHIR Coordinator will use to authenticate with the FHIR endpoint's OAuth server when performing an export.

This client configuration will be created automatically when you save this BFC configuration if the FHIR endpoint’s OAuth server supports discovery and dynamic client registration. Alternatively, you may create this client configuration manually at **Home** > **System Administration** > **Security** > **OAuth 2.0** > **Client** > *FHIRServerIssuerEndpoint* > **Client Configurations**.

Grant Type

OAuth grant type to use when obtaining an access token from the FHIR endpoint's OAuth server.

Depending on the client configuration’s **Required Grant Types**, the possible values for this field are:

- `password` — Resource Owner Password Credentials
- `client_credentials` — Client Credentials

Fetch Token Scopes

Comma-delimited list of OAuth scopes to specify when obtaining an access token from the FHIR endpoint's OAuth server. This applies *only* when the original request to the Bulk FHIR Coordinator did not use an access token. For example, `system/*.read` allows everything.

Important: If the Authorization Type is `OAuth`, any patient or group export requires a minimum of the `system/Patient.read` scope in order to support the Patient/$everything operation that is used in the fetch in order to return both the patient compartment and related resources outside of the patient compartment such as Practitioner. Even if the `_type` parameter filters out Patient resources, the operation still requires the `system/Patient.read` scope, along with scopes for all other resources being retrieved.

Fetch Token Credential ID

The interoperability credential to use to authenticate with the FHIR endpoint's OAuth server if a grant type requires basic authentication credentials.

21.3.4 Bulk FHIR Create/Edit Workflow: Configuring Storage

The settings on the **Storage Location** page of the **Create/Edit** workflow for bulk FHIR configurations are:

Storage Adapter

This field is required. Select HS.BulkFHIR.Storage.File.Adapter.

File URL

Relative URL path for the web application file which serves bulk export files (for example, /file or /bulkfhir/file). Different configurations in the same namespace may use the same URL.

This URL varies depending on your web server configuration: if you employ a single web server for multiple InterSystems IRIS for Health instances, include the instance prefix.

Directory

Storage location for ndjson files that contain the exported FHIR resources. If not specified, defaults to *installDir*`/mgr/Temp/BulkFHIR/`*namespace*`/`. This directory will contain numbered subdirectories for each session. Each session subdirectory will contain resource group directories and files. Distinct directories must be used between namespaces due to the potential for collisions in session identifiers.

21.3.5 Bulk FHIR Create/Edit Workflow: Reviewing and Validating Your Configuration

The **Review** page of the **Create/Edit** workflow for bulk FHIR configurations lists the value of each setting from the other pages. Review the settings and click **Configure** save your bulk FHIR configuration. When you click **Configure**, the BFC will validate each setting and present any issues that need to be addressed. Once each setting has passed validation, the BFC performs any automatic configuration of OAuth client configurations and server descriptions that is necessary.

21.4 Importing a Bulk FHIR Configuration using JSON

To import a configuration, click the **New Configuration** button, select **Import JSON**, and then locate the file using the browse control.

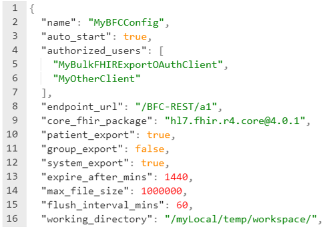
Specifics of the JSON specification for bulk FHIR configuration are shown pretty-printed below by section:

- A simple property that you wish to leave blank may simply be excluded.
- Square brackets in the JSON indicate that a comma-separated list of values may be entered.
- Text colors in the JSON fragments indicate:
  - Green text indicates that a string value in quotes is expected.
  - Red text indicates that a numeric value is expected.
  - Orange text indicates that a boolean value of `true` or `false` is expected.

Sample JSON is shown below for the following pages:

- Configuration Settings page
- Authorization Types Page
- Fetch page
- Storage Location page

21.4.1 Sample JSON for the Configuration Settings Page

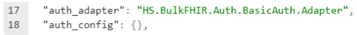


```
{
  "name": "MyBFCConfig",
  "auto_start": true,
  "authorized_users": [
    "MyBulkFHIRExportOAuthClient",
    "MyOtherClient"
  ],
  "endpoint_url": "/BFC-REST/a1",
  "core_fhir_package": "hl7.fhir.r4.core@4.0.1",
  "patient_export": true,
  "group_export": false,
  "system_export": true,
  "expire_after_mins":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

21.4.2 Sample JSON for the Authorization Types Page

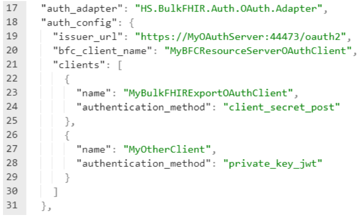
- Basic auth adapter sample JSON
- OAuth adapter sample JSON

21.4.2.1 Basic Auth Adapter



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

21.4.2.2 OAuth Adapter

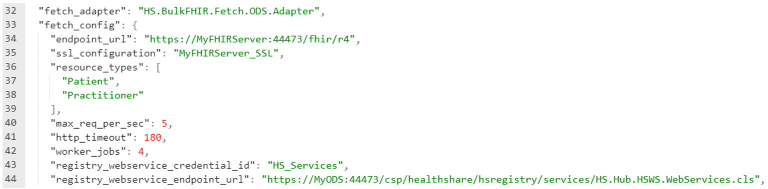


```
  "auth_adapter": "HS.BulkFHIR.Auth.OAuth.Adapter",
  "auth_config": {
    "issuer_url": "https://MyOAuthServer:44473/oauth2",
    "bfc_client_name": "MyBFCResourceServerOAuthClient",
    "clients": [
      {
        "name": "MyBulkFHIRExportOAuthClient",
        "authentication_method": "client_secret_post"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

21.4.3 Sample JSON for the Fetch Page

- ODS fetch adapter sample JSON
- Pure FHIR Fetch adapter sample JSON
- Fetch Authorization Settings

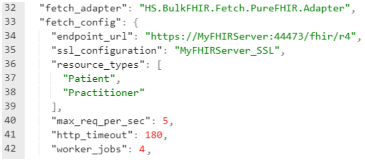
21.4.3.1 ODS Fetch Adapter



```
  "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",
```

See Fetch Authorization Settings for the closing portion of the property `"fetch_config": {`

21.4.3.2 Pure FHIR Fetch Adapter



```
  "fetch_adapter": "HS.BulkFHIR.Fetch.PureFHIR.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,
```

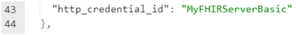
See Fetch Authorization Settings for the closing portion of the property `"fetch_config": {`

21.4.3.3 Fetch Authorization Settings

- HTTP fetch authorization sample JSON
- X-API fetch authorization sample JSON
- OAuth fetch authorization sample JSON

HTTP Fetch Authorization

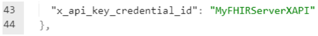
The JSON fragment below is the closing portion of the property `"fetch_config": {`



```
    "http_credential_id": "MyFHIRServerBasic"
  },
```

X-API Fetch Authorization

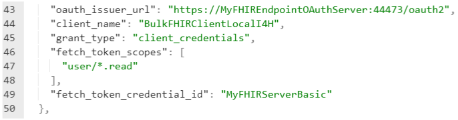
The JSON fragment below is the closing portion of the property `"fetch_config": {`



```
    "x_api_key_credential_id": "MyFHIRServerXAPI"
  },
```

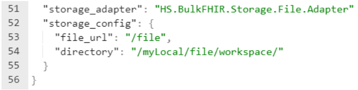
OAuth Fetch Authorization

The JSON fragment below is the closing portion of the property `"fetch_config": {`



```
    "oauth_issuer_url": "https://MyFHIREndpointOAuthServer:44473/oauth2",
    "client_name": "BulkFHIRClientLocalI4H",
    "grant_type": "client_credentials",
    "fetch_token_scopes": [
      "user/*.read"
    ],
    "fetch_token_credential_id": "MyFHIRServerBasic"
  },
```

21.4.4 Sample JSON for the Storage Location Page



```
  "storage_adapter": "HS.BulkFHIR.Storage.File.Adapter"
  "storage_config": {
    "file_url": "/file",
    "directory": "/myLocal/file/workspace/"
  }
}
```

21.5 Performing an Export from the Bulk FHIR Home Page

When the Bulk FHIR Coordinator requests a set of FHIR resources from a FHIR endpoint on behalf of a client it is called an *export*.

This section describes how to:

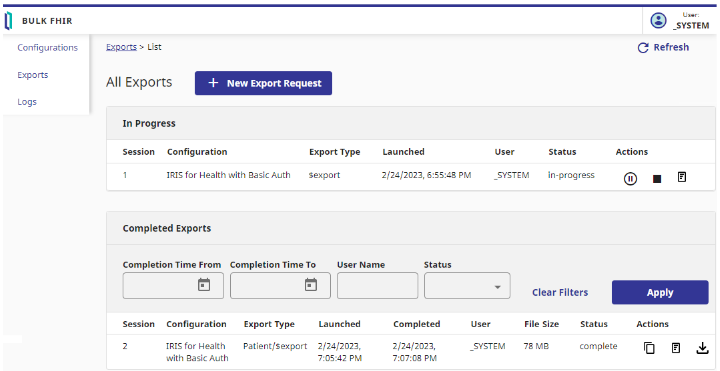
- Access the Export page
- Initiate an export and check on its status
- Download the ndjson for a completed export

- View the export logs

21.5.1 Accessing the Export Page

To initiate or inspect a bulk FHIR export:

1. Log in to InterSystems IRIS for Health as a user with appropriate privileges.
2. Navigate to **Home** > **Health** > *foundationNamespace* > **Bulk FHIR Coordinator**.
3. Navigate to the **Exports** page either by:
    - Clicking the **Exports** link to open the **Exports** > **List** page to view all exports.
    - Clicking 👁 to open the **Exports** > **View** page to view the exports for a particular configuration.



In Progress

Session	Configuration	Export Type	Launched	User	Status	Actions
1	IRIS for Health with Basic Auth	$export	2/24/2023, 6:55:48 PM	_SYSTEM	in-progress	⏸ ■ 🗎

Completed Exports

Completion Time From | Completion Time To | User Name | Status

Session	Configuration	Export Type	Launched	Completed	User	File Size	Status	Actions
2	IRIS for Health with Basic Auth	Patient/$export	2/24/2023, 7:05:42 PM	2/24/2023, 7:07:08 PM	_SYSTEM	78 MB	complete	⧉ 🗎 ⤓

On the **Exports** page, you can view the status of exports that are in progress and completed. You can also take the following actions:

Icon	Action	Notes
	New Export Request	Initiate a new export.
⏸	Pause	Pause an export that is in progress.
▶	Resume	Resume an export that was previously paused.
■	Cancel	Cancel an export that is in progress.
🗎	View Logs	View the logs for an export that is in progress or that is completed. Opens the **Logs** > **List** page where you can filter and view the export logs.
⧉	Copy	Create a new export using the information from a completed export.
⤓	Download	Opens the **Exports** > **Exported Files** page where you can search for and download the ndjson files for specific resources by type, and also download export errors.

Note: Which actions appear depend on the user's roles.

21.5.2 Initiating an Export Request

1. From the **Exports** page, click **New Export Request** to initiate an export.
2. Select a BFC **Configuration** from the drop down list.

3. Click **Next**.
4. Select the type of **Export** from among the available choices which may include **System**, **Group**, and **Patient**.
5. If you selected a group export, enter the **Group ID**. For Unified Care Record ODS, the group ID would be a cohort name.
6. Optionally enter a date in the **Since** field in the format YYYY-MM-DD, or select a date using the date chooser. You may also enter a time by clicking **Add Time** and then entering a time.
7. Click **Export Now** to initiate your export.
8. Your export will appear as a row in the **In Progress** table on the **Exports** page. You may Pause/Resume or Cancel the export when it is in progress. You may also view the logs to determine how the export is progressing.

21.5.3 Downloading the ndjson for a Completed Export

1. From the **Exports** page, in the **Completed Exports** pane, optionally enter a filter value and click **Apply** to filter the list of completed exports.
2. In the desired row of the **Completed Exports** table, click ⤓ to view a list of available files for that export. The exported files are segregated by resource type.
3. To pare down the list of files, optionally enter a **Search** term.
4. To download an ndjson file click ⤓.

21.5.4 Viewing the Export Logs

The export logs provide detailed information for various event types in several different components of the bulk FHIR workflow:

Component	Event Type	Details
BFC	session_action	actions: created, start, pause, resume, complete, failure (reason, stack)
BFC	flush	reasons: size, interval, finalize_session
fetch	rest_request	path, rate_limit_time, http_response_time, http_status (reason, stack)
storage	flush	reasons: size, interval, finalize_session
storage	file_access	client, file

To view the exports logs for a session

1. From the **Exports** page, click 🗎 in the row for an **In Progress** or **Completed** session. Alternatively, click **Logs** to view the logs for all sessions.
2. Optionally enter filter values and click **Apply** to pare down the list of logs.
3. Click 🗎 to view a particular log file.

21.6 Performing a Bulk FHIR Export from a REST Client

When the Bulk FHIR Coordinator requests a set of FHIR resources from a FHIR endpoint on behalf of a client it is called an *export*.

This section describes how to:

- Initiate a REST export
- Check the status of an export that is in progress
- Download the ndjson for a completed export
- Cancelling an export

Note: In some cases—for example, if a proxy server is employed—the public-facing URL to which a REST client directs its Bulk FHIR requests may differ from the URL at which the BFC is hosted.

In such cases, the BFC's rest handler determines the client-visible base URL from the content of a request object's FORWARDED or X-FORWARDED HTTP headers. This logic is implemented by the GetBaseURL() class method of the HS.FHIRServer.Util.BaseURL class. The BFC constructs download links for exports using the GetURLforLink() class method of the HS.BulkFHIR.Util.BaseURL class. This method assumes that download links can use the same client-visible base URL as the URL at which status and `$export` requests are received.

If you must construct client-visible URLs according to different logic in either of these contexts, define a custom **GetBaseURL()** or **GetURLforLink()** class method in the HS.Local.BulkFHIR.Util.BaseURL class. Methods defined in this class will override the originals.

21.6.1 Initiating an Export Request from a REST Client

To initiate a bulk FHIR export from a REST client, send a GET request to your BFC endpoint indicating the desired operation, for example:

- System — `GET https://`*bfcEndpoint*`/$export`
- Patient — `GET https://`*bfcEndpoint*`/Patient/$export`
- Group — `GET https://`*bfcEndpoint*`/Group/`*groupID*`/$export`

If this BFC configuration uses the OAuth **Auth Adapter**, obtain an access token by specifying:

- The OAuth server's access token endpoint:

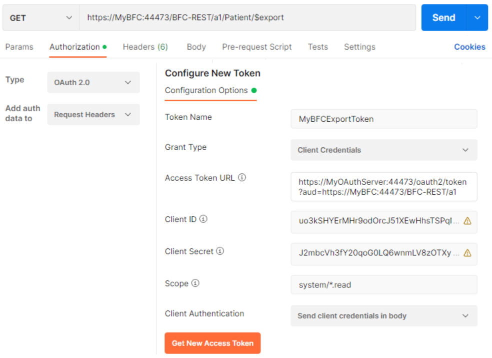
  `issuerEndpoint``/token`

  and audience if required:

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

- The client id and client secret for one of the OAuth **Clients** listed on the **Authorization Types** tab of your BFC configuration.
- A grant type that is supported by the OAuth client. (In InterSystems IRIS, this would be one of the **Required Grant Types** selected on the **General** tab of your OAuth client configuration.)
- A scope, where the minimum required scope is `system/Patient.read`. A scope of `system/*.read` allows everything.

An example patient export using OAuth is shown below:



GET https://MyBFC:44473/BFC-REST/a1/Patient/$export

Type: OAuth 2.0
Add auth data to: Request Headers

Token Name: MyBFCExportToken
Grant Type: Client Credentials
Access Token URL: https://MyOAuthServer:44473/oauth2/token?aud=https://MyBFC:44473/BFC-REST/a1
Client ID: uo3kSHYErMHr9odOrcJ51XEwHhsTSPql...
Client Secret: J2mbcVh3fY20qoG0LQ6wnmLV8zOTXy...
Scope: system/*.read
Client Authentication: Send client credentials in body

The following optional parameters are supported with `$export`:

`_outputFormat`

The BFC exports Newline Delimited JSON (ndjson) files.

The value `application/fhir+ndjson` and the abbreviated values `application/ndjson` and `ndjson` are accepted.

`_since`

Resources will be included in the response if their state has changed after the indicated time in “FHIR Instant” format:

YYYY-MM-DDThh:mm:ss.sss+zz:zz

`_type`

Comma-delimited list of FHIR resource types to include in the export. Defaults to all resource types supported by the fetch adapter as configured.

21.6.2 Checking the Status of an Export from a REST Client

The response header to your initial GET request will include a CONTENT-LOCATION key that indicates a URL of the form:

`bfcEndpoint/status/sessionNumber`

Periodically send GET requests to the CONTENT-LOCATION URL to obtain the status of your bulk FHIR export session.

The following status responses may be returned:

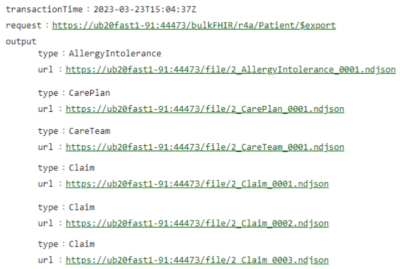
202 Accepted

- The BFC is processing the export.
- The response header will include an X-PROGESS key with the value `in-progress`.

200 OK

- The export is complete, and files are ready for download.
- The response header will include an EXPIRES key indicating how long the ndjson files will be kept on the BFC file server.

- The response body will contain file URLs for ndjson files stored on the BFC file server. For each resource type returned, there will be one or more files:



```
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

- An error occurred on the BFC.

21.6.3 Downloading the ndjson for a Completed Export

Once you receive a `Status: 200 OK` in the response header, your files are ready to download from the BFC file server. To retrieve the files, send a GET request to each file URL.

If this BFC configuration uses the OAuth **Auth Adapter**, obtain a new access token by specifying:

- The **Grant Type** identified on the **Fetch** tab of your BFC configuration.
- The OAuth server’s access token endpoint:

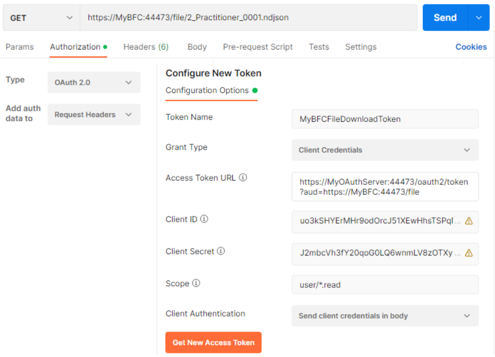
  `issuerEndpoint/token`

  and audience if required:

  `?aud=https://bfcFileEndpoint`

- The client id and client secret for one of the OAuth **Clients** listed on the **Authorization Types** tab of your BFC configuration.
- A scope, typically `user/*.read` for file download.

An example is shown below:



GET https://MyBFC:44473/file/2_Practitioner_0001.ndjson

Type: OAuth 2.0

Add auth data to: Request Headers

Configure New Token

Token Name: MyBFCFileDownloadToken

Grant Type: Client Credentials

Access Token URL: https://MyOAuthServer:44473/oauth2/token?aud=https://MyBFC:44473/file

Client ID: uo3kSHYErMHr9odOrcJ51XEwHhsTSPql ...

Client Secret: J2mbcVh3fY20qoG0LQ6wnmLV8zOTXy ...

Scope: user/*.read

Client Authentication: Send client credentials in body

21.6.4 Cancelling an Export

To cancel an export that is in progress, send a DELETE request to the CONTENT-LOCATION URL. The BFC will return an HTTP status code of “202 Accepted” if the delete is successful. Other status codes indicate an error.

21.7 Bulk FHIR Roles and Resources

The Bulk FHIR Coordinator employs role-based access:

- Bulk FHIR roles
- Bulk FHIR resources and privileges

21.7.1 Bulk FHIR Roles

The Bulk FHIR Coordinator offers the following user roles:

%HS_BFC_Exporter

The %HS_BFC_Exporter role has the following permissions:

- View and perform Patient or Group exports on configurations where the user is listed as an authorized user.
- Pause, stop, resume, or cancel those exports, and view and download logs for those exports.

Assign this role to each dummy InterSystems IRIS user that you create to match your OAuth client configurations (in addition to making the dummy user an authorized user for the configuration).

Because %HS_BFC_Exporter is intended primarily for the dummy user that maps roles to an OAuth client , a user with *only* this role cannot access the Bulk FHIR Coordinator home page through the Management Portal menu. If you assign this role to a real user, either make the BFC foundation namespace the user’s startup namespace, or provide the user a direct link to the BFC home page in the portal.

When combined with the %HS_BFC_Export_Manage role, this user can initiate Patient or Group exports on *all* configurations, and view and download logs for exports that the user initiates. They can also access the home page in the portal.

%HS_BFC_Export_Manage

The %HS_BFC_Export_Manage role has the following permissions:

- View all configurations and all exports.
- Cannot create configurations, initiate exports, or view export logs.

Can be combined with %HS_BFC_Exporter to expand privileges.

%HS_BFC_Administrator

The %HS_BFC_Administrator role has the following permissions:

- View, create, edit, copy, and delete all configurations.
- Perform *system* exports on any configuration that supports them.
- View, pause, stop, resume, cancel, and view or download logs for *all* exports.
- Download exports that the user initiates.

When combined with %HS_BFC_Download_Manage, can download files from all exports.

%HS_BFC_Download_Manage

The %HS_BFC_Download_Manage role has the following permissions:

- Download files from all exports.

Use to expand the privileges of %HS_BFC_Administrator.

Navigate to **Home** > **Security** > **Roles** to view the resources and privileges associated with these roles.

21.7.2 Bulk FHIR Resources and Privileges

Actions within the Bulk FHIR Coordinator are associated with the following privileges:

Resource	Privileges
`%HS_BFC_Configuration`	• W — create, edit or delete configurations • R — view configurations
`%HS_BFC_Download_Manage`	• U — download files created by exports which were started by any user
`%HS_BFC_Export_Download`	• U — download files created by exports which were started by the current user only
`%HS_BFC_Export_Group`	• U — start a group export
`%HS_BFC_Export_Log`	• R — view logs for exports which were started by the current user
`%HS_BFC_Export_Manage`	• U — view, pause, stop, and resume exports in progress which were started by any user
`%HS_BFC_Export_Patient`	• U — start a patient export
`%HS_BFC_Export_Status`	• R — view exports started by the current user • W — pause or cancel exports started by the current user
`%HS_BFC_Export_System`	• U — start a system export
`%HS_BFC_Log_Manage`	• U — view logs for exports which were started by any user

You can also find a description of these resources and privileges by navigating to **Home** > **Security** > **Resources**.

21.8 Creating an OAuth 2.0 Server for the Bulk FHIR Coordinator

If you wish to use OAuth 2.0 for authentication between bulk FHIR REST clients and the Bulk FHIR Coordinator and you do not already have an OAuth 2.0 server, InterSystems IRIS for Health includes a utility that will create an OAuth 2.0

server on your local instance specifically to support SMART Backend Services Authorization for Bulk FHIR Coordinator endpoints. This OAuth server is configured to support dynamic client registration.

Several prerequisites must be met before you can successfully run this utility:

1. Your web server is configured for SSL/TLS.
2. You have created an SSL/TLS configuration for your instance.
3. In the **Configure Secure Communication** dialog in the Installer Wizard, you have created and activated a secure communication configuration.
4. After configuring secure communications in the Installer Wizard, you have configured and activated a Foundation namespace where you will create your bulk FHIR configurations.

The OAuth 2.0 server utility consists of two methods in the class HS.BulkFHIR.OAuth2Installer. Call these methods from your Foundation namespace.

SetupOAuthServer()

Configures an IRIS OAuth 2.0 authorization server in the local IRIS instance for bulk FHIR *and* creates a service registry entry that points to the OAuth server issuer endpoint. This method depends on class parameters `OAuthSSLConfigName` and `OAuthIssuerServiceName` for the values of those two items.

Arguments:

- `pForceDelete`

  0 = abort and return fail if an existing OAuth server is found (default)

  1 = delete existing OAuth server and its clients before re-creating

- `pVerbose`

  0 = do not display method outcome text

  1 = display method outcome text (default)

SetupServiceEntry()

Creates a service registry entry in the current namespace that points to the issuer endpoint for the OAuth server in the current IRIS instance. This method depends on class parameters `OAuthSSLConfigName` and `OAuthIssuerServiceName` for the values of those two items.

This method is only necessary if your OAuth server is already set up as desired and the you want to create a bulk FHIR configuration in a second Foundation namespace.

Arguments:

- `pVerbose`:

  0 = do not display method outcome text

  1 = display method outcome text (default)

Note: Setup of the OAuth 2.0 client configuration can be done automatically, when you create and save your bulk FHIR configuration.

21.9 Bulk FHIR Setup Checklist

Configuring bulk FHIR interactions requires a lot of moving parts in different locations. The checklist below serves as away to insure that all of the required configuration has occurred so that your bulk FHIR interactions succeed:

- FHIR Resource Server Setup Checklist
- Bulk FHIR Coordinator Setup Checklist
- REST Client Setup Checklist

21.9.1 FHIR Resource Server Setup Checklist

- For each FHIR resource server, obtain the endpoint URL.
- Obtain the SSL/TLS configuration information.
- If using the OAuth fetch adapter, obtain the FHIR endpoint's OAuth server endpoint URL. Determine the accepted grant types.
- If using the ODS fetch adapter, obtain the Unified Care Record Registry web service endpoint and SSL/TLS configuration information.
- If the FHIR endpoint imposes a limit on the number of resources that can be returned in a given search, consider increasing this limit in order to prevent search errors. For InterSystems IRIS for Health, when a FHIR server is created, the Max Search Results setting defaults to 1000. To increase this number, go to **Home** > **Health** > **FHIR Configuration** > **Server Configuration** > *endpoint* > **Configuration** > **Max Search Results**. The recommended value depends on the contents of the FHIR server, but a value of 3000 should suffice.

21.9.2 Bulk FHIR Coordinator Setup Checklist

Before creating your BFC configurations, make sure that the prerequisites are in place:

- Create SSL/TLS Configurations
- Create Interoperability Credentials
- Set Up OAuth
- Set Up Users
- Set Up Storage Locations

21.9.2.1 Create SSL/TLS Configurations

- Create an SSL/TLS configuration for communicating with each FHIR Resource Server and OAuth server.
- If using the ODS fetch adapter, create an SSL/TLS configuration for communicating with Unified Care Record Registry web service.

21.9.2.2 Create Interoperability Credentials

- If using the HTTP fetch adapter, create a credential to authenticate with the FHIR endpoint.
- If using the X-API Key fetch adapter, create a credential to authenticate with the FHIR endpoint where the password in the credential is the API key.

- If using the OAuth fetch adapter, and the FHIR endpoint's grant type requires basic authentication credentials, create a credential for the fetch token.

21.9.2.3 Set Up OAuth

If you use OAuth 2.0 as your BFC auth adapter or your FHIR endpoint requires OAuth 2.0 for fetch, you will have to properly set up OAuth, which may include creating an OAuth server for the BFC, server descriptions for FHIR endpoints that require OAuth, and various client configurations.

Create or Identify an OAuth Server

If you use OAuth as your BFC **Auth Adapter**, you will need to provide the URL of the OAuth server for the Bulk FHIR Coordinator that supports SMART Backend Services Authorization. If you do not already have an OAuth server, you can use an InterSystems IRIS for Health utility to create one.

Create an OAuth Client for the BFC as an OAuth Resource Server

If you use OAuth as your BFC **Auth Adapter**, you will need an OAuth client configuration for the BFC as an *OAuth resource server* against your OAuth server issuer endpoint. Note the **Application Name**.

This OAuth client configuration will be created automatically when you save your BFC configuration if your OAuth server supports dynamic client registration.

Create OAuth Clients for Exports

If you use OAuth as your BFC **Auth Adapter**, you will need OAuth client configurations against your OAuth server issuer endpoint for use by bulk FHIR REST clients. Note the **Application Name** and **Client ID** of each client.

These OAuth client configurations will be created automatically when you save your BFC configuration if they are listed in the **Clients** field and your OAuth server supports dynamic client registration. Alternatively, they may be created manually.

Create Server Descriptions and OAuth Clients for FHIR Endpoints

For each FHIR endpoint with an **Authorization Type** of OAuth, create a server description on the BFC instance by using discovery against the FHIR endpoint's OAuth server. Create an OAuth client configuration for the BFC against each FHIR endpoint's OAuth server issuer endpoint using dynamic client registration or by manually entering the client ID and client secret.

Both the server description and the BFC client configuration for the FHIR endpoint's OAuth server will be created automatically when you save your BFC configuration if the FHIR endpoint's OAuth server supports discovery and dynamic client registration.

21.9.2.4 Set Up Users

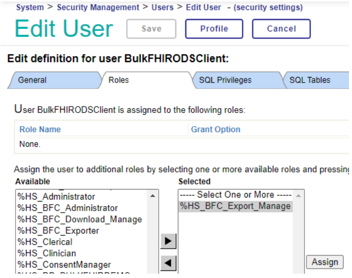
- Create an administrative user with the %HS_BFC_Administrator role.
- Create a dummy user for each OAuth export client The dummy user should hold at least the %HS_BFC_Exporter role and be listed as an authorized user:

  Each OAuth export client must have *both* an OAuth client configuration *and* a dummy InterSystems IRIS user of the same name. The dummy user serves to map the appropriate roles to the OAuth client.

  To create a dummy user for an OAuth client:

  1. On the Bulk FHIR Coordinator instance, navigate to **Home** > **System Administration** > **Security** > **Users** > **Create New User**.

2. In the **Name** field, enter the *same name* that you entered in the **Clients** *name* string when you configured the auth adapter, namely the **Application Name** specified in the OAuth client configuration.
3. In the **Password** and **Password (confirm)** fields, enter a random string of characters, using the same string for both fields. Even though this account will not be used for login purposes, a password is required in order to create an InterSystems IRIS user.
4. *Deselect* **User Enabled** as this user account will not be used for login purposes. This will prevent anyone from attempting to login as the user.
5. Click **Save**.
6. On the **Roles** tab, add the appropriate user roles, typically %HS_BFC_Exporter. To add a role, select it in the **Available** pane and click ▶ to move it to the **Selected** pane. Then click **Assign** as shown below.


System > Security Management > Users > Edit User - (security settings)
Edit User
Save
Profile
Cancel
General
Roles
SQL Privileges
SQL Tables
Assign

Edit definition for user BulkFHIRODSClient:

User BulkFHIRODSClient is assigned to the following roles:

Role Name	Grant Option
None.	

Assign the user to additional roles by selecting one or more available roles and pressin

Available:
%HS_Administrator
%HS_BFC_Administrator
%HS_BFC_Download_Manage
%HS_BFC_Exporter
%HS_Clerical
%HS_Clinician
%HS_ConsentManager

Selected:
----- Select One or More -----
%HS_BFC_Export_Manage

The dummy user is used solely as a means to map user roles to an OAuth client, which enables a REST export client to engage in bulk FHIR interactions with this BFC endpoint.

21.9.2.5 Set Up Storage Locations

- Identify a temporary working directory for your exports.
- Identify a storage directory with sufficient space for the ndjson files that will be produced by the exports.
- When you save your BFC configuration, a CSP app will be created using the file URL you provide.

21.9.3 REST Client Setup Checklist

- As described above, use dynamic client registration against the BFC endpoint URL to create an OAuth client configuration for the REST client to use.
- When you initiate or check the status of a bulk FHIR export from a REST client using OAuth, present an access token with:
  - The **Grant Type** identified on the **Fetch** tab of your BFC configuration.
  - The OAuth server’s access token endpoint (*issuerEndpoint*`/token`) and audience if required (`?aud=https://`*bfcEndpoint*).
  - The client id and client secret for one of the OAuth **Clients** listed on the **Authorization Types** tab of your BFC configuration.
  - A scope, where the minimum required scope is `system/Patient.read`. A scope of `system/*.read` allows everything.
- When you download ndjson files from the BFC file server with a REST client using OAuth, present an access token with:
  - The **Grant Type** identified on the **Fetch** tab of your BFC configuration.

- The OAuth server’s access token endpoint (*issuerEndpoint*`/token`) and audience if required (`?aud=https://`*bfcFileEndpoint*).
- The client id and client secret for one of the OAuth **Clients** listed on the **Authorization Types** tab of your BFC configuration.
- A scope, typically `user/*.read` for file download.

22

Pre-2020.2 FHIR Technology

For details about using pre-2020.2 HL7® FHIR® technology, see the legacy FHIR books that are available at InterSystems Legacy Documentation.

22.1 Upgrade pre-2020.2 Transformations

The strategy for customizing bi-directional SDA-FHIR transformations in InterSystems products was different in the legacy FHIR technology (pre-2020.2). This section discusses how to convert code developed to customize transformation in legacy FHIR implementations to the new FHIR architecture.

The APIs called by an application to perform transformations have changed. In the legacy implementation, applications called methods of the HS.FHIR.DTL.Util.API.HC.Transform class to invoke the transformation. This class is obsolete and direct calls to its methods will not work with the new FHIR architecture. Now, transformations are invoked with methods of the `HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR` and `HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3` classes.

The pre-2020.2 FHIR technology used callback objects to implement custom logic controlling how transformations were executed. In the new architecture, customization is accomplished by subclassing the transformation API class and overriding its methods. For information about customizing these transformation methods, see Customizing Transformation API Classes.

When upgrading from your pre-2020.2 callback classes, you need to migrate the logic in your callback methods to the overridable methods in the new transformation classes. The following table summarizes the relationship between callback methods in the pre-2020.2 HS.FHIR.DTL.Util.API.HC.Callback.Default.SDA3ToSTU3 class and new overridable methods in HS.FHIR.DTL.Util.API.Transform.SDA3ToFHIR.

Legacy Callback Method	New Overridable Method
`IsDuplicate`	`IsDuplicate`
`AssignResourceId`	`GetId`
`GetIdByIdentifier`	`GetId`
`GetPatientId`	`GetId`
`GetURLPrefix`	`GetBaseURL`

The following table summarizes the relationship between callback methods in the legacy HS.FHIR.DTL.Util.API.HC.Callback.Default.STU3ToSDA3 class and new overridable methods in HS.FHIR.DTL.Util.API.Transform.FHIRToSDA3.

Legacy Callback Method	New Overridable Method
`AssignEncounterNumber`	`GetIdentifier`
`AssignExternalId`	`GetIdentifier`
`GetSendingFacility`	`GetSendingFacility`
`GetSendingFacilityFromReference`	`GetSendingFacility`

One of the methods in the new transformation classes, `GetDTL()`, can be overridden to select a custom DTL class that was written for the pre-2020.2 FHIR technology. In this case, the `GetDTL()` method should call the old method `GetDTLPackageAndClass()`. For example:

Class Member

```
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
}
```

22.1.1 Upgrading Transformation Productions

The business processes used to perform transformations in a FHIR interoperability production, HS.FHIR.DTL.Util.HC.SDA3.FHIR.Process and HS.FHIR.DTL.Util.HC.FHIR.SDA3.Process, have been updated to use the new transformation API. If your legacy implementation used the standard business processes, you must complete the following tasks before starting the production after the upgrade:

- Specify a value for the FHIRMetadataSet setting of the business process.
- If the TransmissionMode setting was set to **Batch**, you must change the setting to specify **transaction** or **individual**.

A

Upgrading a Legacy Repository to JSON Advanced SQL

A.1 Mappings

Although it is recommended to use the JSON Advanced SQL strategy when possible, the legacy strategy is still supported. This appendix provides details comparing the legacy strategy to the advanced strategy, and includes upgrade instructions for those currently using the legacy strategy who wish to take advantage of the additional features available with the advanced strategy.

A.1.1 Resource Repository Classes

The Resource Repository consists of the following architectural classes:

Interactions

- Legacy Repository Class: `HS.FHIRServer.Storage.Json.Interactions`
- Advanced Repository Class: `HS.FHIRServer.Storage.JsonAdvSQL.Interactions`

InteractionsStrategy

- Legacy Repository Class: `HS.FHIRServer.Storage.Json.InteractionsStrategy`
- Advanced Repository Class: `HS.FHIRServer.Storage.JsonAdvSQL.InteractionsStrategy`

RepoManager

- Legacy Repository Class: `HS.FHIRServer.Storage.Json.RepoManager`
- Advanced Repository Class: `HS.FHIRServer.Storage.JsonAdvSQL.RepoManager`

A.1.2 General Search Limitations

The JSON Advanced SQL strategy supports searching across multiple resource types within a compartment, as described in General Limitations.

The legacy strategy does *not* support searching across multiple resource types within a compartment.

A.1.3 Search Parameter Types

A.1.3.1 Date, Number, and Quantity

For searches using the date, number, or quantity search parameters, the JSON Advanced SQL strategy interprets the parameter value as an implicit range. In other words, a search for a quantity parameter with a value of 100 matches any values in the range [99.5, 100.5). This is in full conformance with the HL7® FHIR® specification for these search types (for example, http://hl7.org/fhir/search.html#date).

The legacy strategy does *not* interpret values for these search parameter types as implicit ranges.

A.1.3.2 Reference

For searches that use a reference parameter to search for canonical references, the JSON Advanced SQL strategy supports the use of versions within the search, as described by the FHIR specification (https://hl7.org/fhir/search.html#versions).

The legacy strategy does *not* support versioned searches for canonical references.

A.1.4 Search Modifiers

Modifier	Legacy Strategy Support	JSON Advanced SQL Support
`:above`	Supported for uri	Supported for URI
`:below`	Supported for uri	Supported for URI
`:code-text`	Not supported	Not supported
`:contains`	Supported for strings	Full support (string and URI)
`:exact`	Supported for strings except for accented characters. For example, `?given:exact=Nino` returns Patient with given name `Niño`	Full support
`:identifier`	Not supported	Full support
`:in`	Not supported	Not supported
`:iterate`	Limited to `_include`	Full support
`:missing`	Not supported	Not supported
`:not`	Not supported	Not supported
`:not-in`	Not supported	Not supported
`:of-type`	Not supported	Full support
`:text`	Not supported	Supported for reference and token, not supported for string
`:text-advanced`	Not supported	Not supported
`:[type]`	Full support	Full support

A.1.5 Prefixes

Prefix	Legacy Strategy Support	JSON Advanced SQL Support
`eq`	Full support	Full Support
`ne`	Full support	Full support
`gt`	Full support	Full support
`lt`	Full support	Full support
`ge`	Full support	Full support
`le`	Full support	Full support
`sa`	Not supported	Full support
`eb`	Not supported	Full support
`ap`	Not supported	Full support

A.1.6 Search Result Parameters

Parameter	Legacy Strategy Support	JSON Advanced SQL Support
`_contained`	Not supported	Not supported
`_count`	Full support as described in the official specification	Full support as described in the official specification
`_elements`	Full support as described in the official specification	Full support as described in the official specification
`_graph`	Not supported	Not supported
`_include`	Full support as described in the official specification	Full support as described in the official specification
`_maxresults`	Not supported	Not supported
`_revinclude`	Full support as described in the official specification	Full support as described in the official specification
`_score`	Not supported	Not supported
`_sort`	Full support as described in the official specification	Full support as described in the official specification
`_summary`	Support for `_summary=count`, `_summary=data`, and `_summary=text`. For details, see the official specification.	Support for `_summary=count`, `_summary=data`, and `_summary=text`. For details, see the official specification.
`_total`	Not supported	Not supported

A.2 Migrating to JSON Advanced SQL

Follow these instructions to migrate from the legacy strategy to the JSON Advanced SQL strategy with no down time:

1. Before you begin migrating from the legacy strategy to the JSON Advanced SQL strategy, perform these pre-migration steps:
   a. Evaluate your customizations of the existing repository that extends the legacy JSON strategy.
   b. Create a new strategy by extending these classes:
      - `HS.FHIRServer.Storage.JSONAdvSQL.InteractionsStrategy`
      - `HS.FHIRServer.Storage.JSONAdvSQL.RepoManager`
      - `HS.FHIRServer.Storage.JSONAdvSQL.Interactions`
   c. Migrate the relevant customizations from your legacy strategy to the new extended classes in your JSON Advanced SQL strategy.
   d. Validate that the new customizations work, and make needed fixes.
2. Begin migration by creating a new repository. To do so, execute the following command in the terminal:

   ```
   set status = ##class(HS.FHIRServer.Storage.JsonAdvSQL.ConvertJson).Start(<endpoint_to_be_converted>, <new_strategy_key>)
   ```

   `<new_strategy_key>` is the strategy key associated with your extended JSON Advanced SQL classes. For more information about strategy keys, see Subclass Parameters. For information about additional optional parameters such as `pNumWorkers` and `pWait`, see HS.FHIRServer.Storage.JsonAdvSQL.ConvertJson in the class reference.

   This command creates a new repository that uses existing `Resource` and `Version` tables as well as most other settings from the old repository. It creates new search tables and maps them to the `Resource` database if needed. It runs indexing of the new search tables in a background job.
3. Wait for the background job indexing the new search tables to finish. Progress and completion of this job are logged in the interoperability event log.
4. When the background job is complete, cut over from the old `Json`-based repository to the new one based on `JsonAdvSQL`, by executing the following command:

   ```
   set status = ##class(HS.FHIRServer.Storage.JsonAdvSQL.ConvertJson).Cutover(<endpoint_to_be_converted>,<service_id_of_new_endpoint>)
   ```

   The exact arguments to use in this command are included in the background job completion message in the interoperability event log.

   This command assocates the `cspUrl` with the new repository and disables the old one. Once this is complete, all new requests to that URL will be handled by the new `JsonAdvSQL`-based code and data structures. The search tables associated with the old repository will no longer be maintained.
5. Optional. You can clean up unnecessary parts of the old `Json`-based endpoint by executing the following command:

   ```
   set status = ##class(HS.FHIRServer.Storage.JsonAdvSQL.ConvertJson).Cleanup(<old_repo_service_id>)
   ```

   The old repository service ID is included in the terminal output from the cut-over process.

   The following entities are cleaned up:

   - All `HSFHIR` search tables associated with the old endpoint
   - All `HS_FHIRServer_Storage_JSON.SearchColumn` rows associated with the old service key

- The cached `CapabilityStatement` for the old endpoint
- The `ComparmentsIdx` index from the resource table
- The `HS_FHIRServer.Repo` where ID = `<old_repo_service_id>`