# Table of Contents

**About This Book** ........................................................................................................................................... 1

**1 Introduction to the FHIR Server** ................................................................................................................. 3
  1.1 Architecture .................................................................................................................................................. 3
  1.1.1 More About the Service ....................................................................................................................... 4
  1.1.2 More About the InteractionsStrategy ................................................................................................. 5
  1.1.3 More about the Interactions Class .......................................................................................................... 5
  1.1.4 Messaging ................................................................................................................................................ 5
  1.2 Writing a FHIR Server Application ........................................................................................................... 5
  1.2.1 Pre-Built Functionality ......................................................................................................................... 6
  1.2.2 Developing Custom Functionality ......................................................................................................... 6
  1.3 Controlling Server Behavior .................................................................................................................... 6
  1.3.1 ConfigData Object ................................................................................................................................... 6
  1.4 Making REST Calls ........................................................................................................................................ 7

**2 Installing and Configuring a FHIR Server** ................................................................................................... 9
  2.1 Configuring a FHIR Server ....................................................................................................................... 10
  2.2 Installing and Configuring Programmatically ........................................................................................ 13
    2.2.1 Configuring the FHIR Server Programmatically ........................................................................... 15

**3 FHIR Productions** ....................................................................................................................................... 17
  3.1 Using a FHIR Production .......................................................................................................................... 17
    3.1.1 Creating the Interoperability Production ......................................................................................... 17
    3.1.2 Configuring the FHIR Server Endpoint .............................................................................................. 18
  3.2 Production-Based Servers ........................................................................................................................ 18
  3.3 Forwarding FHIR Requests .................................................................................................................... 18
  3.4 Messaging .................................................................................................................................................. 19

**4 FHIR Data** ..................................................................................................................................................... 21
  4.1 Accessing FHIR Payloads .......................................................................................................................... 21
    4.1.1 Implementations Without a Production .............................................................................................. 21
    4.1.2 Production-Based Implementations ..................................................................................................... 21
    4.1.3 Direct Calls to Interactions Class ......................................................................................................... 22
  4.2 FHIR Data and Dynamic Objects ............................................................................................................ 22
  4.3 Data Load Utility .......................................................................................................................................... 23

**5 ObjectScript Applications** .......................................................................................................................... 25
  5.1 GET Resources .............................................................................................................................................. 25
  5.2 POST Resources .......................................................................................................................................... 48
  5.3 Bypassing the Service ............................................................................................................................... 26
  5.4 Handling FHIR Data as XML .................................................................................................................. 26
  5.5 Handling FHIR Data as a Stream .............................................................................................................. 27
  5.6 Validating FHIR Resources ...................................................................................................................... 27

**6 Profiling FHIR** .............................................................................................................................................. 29
  6.1 Modifying the Capability Statement ......................................................................................................... 29
    6.1.1 Retrieving the Capability Statement ................................................................................................... 29
    6.1.2 Updating the Capability Statement .................................................................................................... 30
  6.2 Extensions .................................................................................................................................................... 30
  6.3 Custom Metadata Sets ............................................................................................................................... 30
About This Book

This book explains how to use the FHIR® components of InterSystems products for FHIR STU3 and later. Currently, transformations between FHIR and SDA are not supported. To develop a FHIR server for FHIR DSTU2 and STU3 using legacy technology that includes transformations, see the legacy FHIR books that are available at InterSystems Legacy Documentation.

Chapters include:

• Introduction to the FHIR Server
• Installing and Configuring a FHIR Server
• FHIR Productions
• FHIR Data
• ObjectScript Applications
• Server Security
• Profiling FHIR
• FHIR Operations
• Debugging and Maintaining the FHIR Server
• InterSystems Resource Repository
• FHIR Clients

For more information about the FHIR standard, see http://www.hl7.org/fhir/; for FHIR license and legal terms see http://www.hl7.org/fhir/license.html.
Introduction to the FHIR Server

The following sections describe the basics of FHIR® server technology in InterSystems products. For an overview of what a default FHIR server supports in InterSystems IRIS for Health, see What is Supported?.

In most cases, implementing a FHIR server in Health Connect refers to the ability to accept requests from a FHIR client into an interoperability production. The default FHIR storage architecture provided with the FHIR server in InterSystems IRIS for Health is not available in Health Connect. Though it is possible to write a custom architecture for a FHIR server in Health Connect, most use cases do not include writing this custom code. For more information about using the endpoint of a FHIR server to receive FHIR requests in Health Connect, see FHIR Productions.

1.1 Architecture

Tracing a FHIR request through the FHIR server provides a good overview of the major architectural features of the server. 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 a server-side ObjectScript application. 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. InterSystems IRIS for Health applications using the default 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.
1.1.1 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.

### 1.1.2 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. When the installer creates a new endpoint, it calls the `Create` method of the InteractionsStrategy to set up the FHIR server for a specific purpose. 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 in InterSystems IRIS for Health 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.

The InteractionsStrategy is also responsible for managing the endpoints that have been registered in the namespace.

### 1.1.3 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.

### 1.1.4 Messaging

The message class that the server architecture uses to pass 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`.

For information about accessing the FHIR payload in a message, see Accessing FHIR Payloads.

### 1.2 Writing a FHIR Server Application

When developing an application that leverages the FHIR server technology, it is helpful to differentiate between classes that are intended to be subclassed and those that do not need to be modified.
1.2.1 Pre-Built Functionality

For most applications, the default Service (HS.FHIRServer.Service) effectively enforces the FHIR standard and routes requests for execution, and does not need to be subclassed. Likewise, the classes that process bundles (HS.FHIRServer.DefaultBundleProcessor) and validate resources (HS.FHIRServer.API.ResourceValidator) are fully-functional and do not need to be subclassed, though they can be if custom processing is required. In addition, the ConfigData object (HS.FHIRServer.API.ConfigData) that the Service uses to configure server behavior should not be subclassed.

InterSystems IRIS for Health comes with pre-built Interactions and InteractionsStrategy classes that store and retrieve resources from SQL tables, allowing you to implement a fully-operational FHIR server with minimal custom coding. This storage strategy is known as the Resource Repository.

1.2.2 Developing Custom Functionality

Implementing custom functionality begins with subclassing the Interactions and InteractionsStrategy classes. While it's possible to write a completely custom backend for your FHIR server by directly subclassing HS.FHIRServer.API.Interactions and HS.FHIRServer.API.InteractionsStrategy, if your application needs to store and retrieve FHIR resources, it is probably faster and more reliable to subclass the Resource Repository’s HS.FHIRServer.Storage.Json.InteractionsStrategy and HS.FHIRServer.Storage.Json.Interactions classes to implement the custom logic. These classes inherit from the superclasses in the HS.FHIRServer.API package.

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

- Modify the StrategyKey parameter to specify a unique identifier.
- Modify the InteractionsClass parameter to specify the name of your Interactions subclass.

Once you have successfully subclassed the Interactions and InteractionsStrategy, you can customize other aspects of the server’s functionality like operations, bundle processing, and validation by subclassing the appropriate class (HS.FHIRServer.API.OperationHandler, HS.FHIRServer.DefaultBundleProcessor, and HS.FHIRServer.API.ResourceValidator, respectively). For details about FHIR operations, including adding custom ones without overwriting default operations included with the Resource Repository, see FHIR Operations.

1.3 Controlling Server Behavior

The FHIR server installer takes a metadata set and InteractionsStrategy when creating an endpoint. While the InteractionsStrategy provides the backend logic of the FHIR server, the FHIR metadata set controls the capabilities of the server, for example what resources are allowed, what interactions on those resources are allowed, and the valid search parameters. InterSystems provides metadata sets for the base FHIR specifications of the supported FHIR versions; these default metadata sets should NEVER be modified directly. Rather, these standards can be extended or restricted by creating a custom metadata set or, in the special case of the CapabilityStatement, by passing in a modified CapabilityStatement to the SetMetadata method of the Interactions class.

1.3.1 ConfigData Object

When the installer creates an endpoint, it also creates a ConfigData object (HS.FHIRServer.API.ConfigData) that keeps track of the InteractionsStrategy and metadata of the endpoint and specifies server configuration settings that affect things like security, session timeout, and search results. These configuration settings can be specified with the command-line configuration utility or by programmatically setting properties of the ConfigData object.
The ConfigData object is managed by the InteractionsStrategy and used by the Service to configure itself.

### 1.4 Making REST Calls

When using a REST client to access the InterSystems FHIR server, keep the following in mind:

- The base path of an endpoint is: `ServerIPAddress:SuperServerPort/baseURL`, where:
  - `ServerIPAddress` is the IP address of the InterSystems server where the FHIR server is installed.
  - `SuperServerPort` is the InterSystems server’s superserver port. You can find this superserver port in the Management Portal by going to System Administration > Configuration > System Configuration > Memory and Startup.
  - `baseURL` is the endpoint created during installation. For example, `/fhirapp/namespace/fhir/R4`.

For example, a REST call to post a resource might look like:

```
```
Installing and Configuring a FHIR Server

InterSystems products use a command-line utility to create a FHIR® server and its endpoint. Once created, the same utility can be used to configure the server for specific purposes. Alternatively, you can install and configure a server programmatically. Multiple FHIR servers can be installed in the same namespace.

Important: Before installing a FHIR server, you must consider whether you want to customize it now or in the future. If you install a FHIR server without creating new Interactions and InteractionsStrategy subclasses, you will not be able to perform certain customizations such as adding FHIR operations or modifying how bundles are processed. For information about preparing for these customizations before installing the FHIR server, see Developing Custom Functionality. To view the options that can be configured without creating Interactions and InteractionsStrategy subclasses, see Configuring a FHIR Server.

To run the command-line utility that is used to install a FHIR server:

1. Open the InterSystems Terminal.
2. Change to the HSLIB namespace:
   
   ```
   set $namespace = "HSLIB"
   ```
3. Enter the following to create a new namespace for the FHIR server. If you have already created an interoperability namespace using the Management Portal, you can skip this step.

   ```
   set status = ##class(HS.HC.Util.Installer).InstallFoundation("FHIRNamespace")
   ```

   Where FHIRNamespace is the name of your new namespace.

   If the process of creating the namespace does not begin, you can check for errors by entering do $SYSTEM.Status.DisplayError(status).

4. Enter the following to change to the new namespace:

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

5. To run the installation and configuration utility, enter:

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

6. Choose option 1) Create a FHIRServer Endpoint.
7. Follow the prompts for your new endpoint.
2.1 Configuring a FHIR Server

The command-line utility used to install a FHIR server is also used to configure it. For example, you can route FHIR requests through an interoperability production, control server behavior, or put the server in debug mode during development. These configuration settings can also be modified programmatically by setting the properties of the server’s ConfigData object.

To configure the FHIR server:

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```

2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 3) Configure a FHIRServer Endpoint.

4. Choose the endpoint of the FHIR server that you are configuring.

5. Follow the remaining configuration prompts.
<table>
<thead>
<tr>
<th>Prompt</th>
<th>Description</th>
</tr>
</thead>
</table>

Configuring a FHIR Server
<table>
<thead>
<tr>
<th>Prompt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint enabled</td>
<td>Specify whether the endpoint is enabled. A disabled endpoint rejects requests from FHIR clients.</td>
</tr>
<tr>
<td>OAuthClientName:</td>
<td>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 <a href="#">OAuth 2.0 Authorization</a>.</td>
</tr>
<tr>
<td>ServiceConfigName:</td>
<td>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 <code>HS.FHIRServer.Interop.Service</code>. For more details, see <a href="#">FHIR Productions</a>.</td>
</tr>
<tr>
<td>RequiredResource:</td>
<td>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 <a href="#">Adding Authorization Requirements</a>.</td>
</tr>
<tr>
<td>FHIRSessionTimeout:</td>
<td>Maximum number of seconds between requests to the service before any session data is considered stale.</td>
</tr>
<tr>
<td>DefaultSearchPageSize:</td>
<td>Search result page size to use when a search does not contain a <code>_count</code> parameter.</td>
</tr>
<tr>
<td>MaxSearchPageSize:</td>
<td>Maximum search result page size to prevent an excessive user-specified page size.</td>
</tr>
<tr>
<td>MaxSearchResults:</td>
<td>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 <code>_include</code> 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.</td>
</tr>
<tr>
<td>MaxConditionalDeleteResults:</td>
<td>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.</td>
</tr>
<tr>
<td>Prompt</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DefaultPreferHandling:</td>
<td>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.</td>
</tr>
<tr>
<td>DebugMode:</td>
<td>For information about setting a debug mode, see Debugging the FHIR Server.</td>
</tr>
</tbody>
</table>

### 2.2 Installing and Configuring Programmatically

For applications that need to install a FHIR server programmatically rather than using the command-line utility, 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. This is passed as the first argument. Be sure the URL begins with a slash (/).
- Classname of the FHIR Server's InteractionsStrategy. This is passed as the second argument.
- The identifier of the FHIR metadata set used for the FHIR server. This is the third argument. To use base FHIR metadata, pass HL7v30 for STU3 or HL7v40 for R4. If you want the FHIR server to use a custom metadata set, call InstallMetadataSet() first, then pass its key into InstallInstance(). For more information about creating a custom metadata set, see Custom Metadata Sets.

Two optional arguments can also be passed to InstallInstance():

- If the FHIR server is using OAuth to control authentication and authorization, enter the OAuth client name as the fourth argument. For more information about using OAuth security, see OAuth 2.0 Authorization.

For example, the following ObjectScript code installs a FHIR server using the default JSON storage strategy for IRIS for Health (Resource Repository).

```
Set appKey = "/MyFHIRApp/fhir/r4"
Set strategyClass = "HS.FHIRServer.Storage.Json.InteractionsStrategy"
Set metadataConfigKey = "HL7v40"

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

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

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

The InstallInstance() method creates a HS.FHIRServer.API.ConfigData object when it creates the FHIR server endpoint. You can control the behavior of the FHIR server by modifying the properties of this object. Refer to the class reference or the configuration utility’s settings for a description of these properties.

For example, the code to change the debug mode of the FHIR server would retrieve the FHIR server’s configData object, modify the DebugMode property, and then save the ConfigData object. The ObjectScript code might look like:

```ObjectScript
Set appKey = "/MyFHIRApp/fhir/r4"
Set strategy = ##class(HS.FHIRServer.API.InteractionsStrategy).GetStrategyForEndpoint(appKey)
Set configData = strategy.GetServiceConfigData()
Set configData.DebugMode = 4
Do strategy.SaveServiceConfigData(configData)
```
Unlike legacy FHIR® implementations of InterSystems products, which always used interoperability productions for data processing, a default FHIR server now processes incoming requests without a production, passing the requests directly to the server’s Service. However with a small modification, you can route the request through a business service in a production, and then configure the rest of the production to control what happens to the request. For example, if you still want the request to reach the FHIR server’s Service, you can use a pre-built business operation that forwards the request to the Service. However, if your goal is to forward the request to another system, you can route the request to a different business operation that communicates with the external system.

Regardless of what the interoperability production does with the incoming FHIR request, the process of routing the request to the business service begins with creating the endpoint for a FHIR server. This is true even if the FHIR request never reaches the FHIR server’s Service, as is the case when the production’s business operation forwards the request to an external system.

3.1 Using a FHIR Production

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

• Create an interoperability production and add a special business service.
• After installing a FHIR server endpoint, configure the FHIR server’s ServiceConfigName option 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 server’s configuration matches the name in the interoperability production.

3.1.1 Creating the Interoperability Production

When the Foundation namespace for the FHIR server endpoint was created, the installation process created an interoperability production that can be used as a FHIR production, but you can also create a new production for this purpose. What is important is that you 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 a 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 ServiceConfigName option. For basic information on creating a production and adding a business service, see First Look: Connecting Systems Using Interoperability Productions.
3.1.2 Configuring the FHIR Server 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. Simply run the installation and configuration utility, and use the ServiceConfigName option to specify the full name of the production’s business service. Specifying the name of the business service does not automatically create the production or business service.

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

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```
   Where FHIRNamespace is the name of the endpoint’s namespace.

2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 3) Configure a FHIRServer Endpoint.

4. Choose the endpoint of the FHIR server that you are configuring.

5. When you get to the ServiceConfigName prompt, 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. Accept the remaining prompts.

Note: The ServiceConfigName option can also be specified when creating a new endpoint.

3.2 Production-Based Servers

Without using an interoperability production, a FHIR request goes directly from the REST handler to the FHIR server’s Service. However if you configure the endpoint to forward that request to a business service in an interoperability production, you need to add a built-in business operation that routes the request from the production to the Service if you want it processed by the FHIR server.

The HS.FHIRServer.Interop.Operation business operation takes the FHIR request from the business service or a business process within the FHIR production and forwards it to the FHIR server’s Service for processing. Once you have added this business operation to the production, set the Target Config Name setting of the business service or business process to specify the name of the business operation.

Responses from the Service are sent back to the HS.FHIRServer.Interop.Operation business operation on its way back to the business service, which returns it to the REST handler.

3.3 Forwarding FHIR Requests

Once the FHIR server’s endpoint is configured to forward a request to an interoperability production, the production controls what happens to that request. For example, you may not need the InterSystems product to act as a FHIR repository, but rather as a proxy server that forwards requests to external systems. In these cases, you would not use...
HS.FHIRServer.Interop.Operation as the production’s business operation, but rather forward the message to a pre-built or custom business operation that acted as a FHIR client by sending the FHIR data to an external FHIR endpoint. Rather than develop a custom business operation, you can use a pre-built legacy business operation that takes FHIR data and sends it out to an external FHIR server. However, this business operation, HS.FHIR.REST.Operation, uses the legacy message classes, so your implementation would have to convert the new message classes to the legacy message classes before using HS.FHIR.REST.Operation. For more information about these legacy message classes, see the legacy FHIR books that are available at InterSystems Legacy Documentation.

### 3.4 Messaging

The message class used to pass FHIR requests within the production is HS.FHIRServer.Interop.Request. The message class used to pass a response from the FHIR server back through the production to the FHIR client is HS.FHIRServer.Interop.Response.

For information about accessing the FHIR payload of a request or response, see Accessing FHIR Payloads in Production-Based Implementations.
4

FHIR Data

The following sections explore how to work with FHIR® data in InterSystems products.

4.1 Accessing FHIR Payloads

The process of accessing FHIR payloads varies depending on the message class carrying the payload. The messages classes for a FHIR production are different than the message classes of a default implementation that does not use a production, therefore accessing the FHIR payload of requests and responses varies depending on the implementation. In both case, manipulating the FHIR data consists of working with dynamic objects.

4.1.1 Implementations Without a Production

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. For example, the following code demonstrates how an ObjectScript application can retrieve a Patient resource from the FHIR server and store it in a patient variable so it can be manipulated as a dynamic object.

```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.RequestMethod = "GET"
set request.RequestPath = "/Patient/1"
do fhirService.DispatchRequest(request, .response)
set myPatient = response.Json
```

For more information about storing requests and retrieving responses programmatically, see ObjectScript Applications.

4.1.2 Production-Based Implementations

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 process in the production that needs to access the FHIR payload must use the `QuickStreamID` to obtain a JSON string, then convert that into a dynamic object. For example, a BPL business process could use the following code to access and modify the FHIR payload of a request message that was in JSON format:

```java
//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()
}
```

Of course, the preceding code sample does not apply to request and response message that have a FHIR payload in XML format.

### 4.1.3 Direct Calls to Interactions Class

FHIR data can be retrieved from the server’s storage strategy programmatically by calling methods of the Interactions class (HS.FHIRServer.API.Interactions). This data is retrieved as a dynamic object. For more information about these method calls, see Bypassing the Service.

### 4.2 FHIR Data and Dynamic Objects

In InterSystems products, 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.

The following code fragments provide an introduction to working 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 manipulating 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.

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

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

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

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

Extracting a Value

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

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.

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.

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)

4.3 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, and can be expressed in JSON, XML, or both. 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

```plaintext
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:

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

The Data Load utility takes three optional arguments that control whether it displays progress, logs statistics, or limits the number of files in the directory that it will process. For details on these arguments, see `HS.FHIRServer.Tools.DataLoader.SubmitResourceFiles()`
5

ObjectScript Applications

The FHIR® server’s \texttt{Service}\texttt{(HS.FHIRServer.Service)} allows an ObjectScript application to perform CRUD operations on the server directly without going over HTTP. The same method, \texttt{DispatchRequest()}, that the REST handler calls to send a FHIR request to the server can be used by an ObjectScript application to send a request to the server. If an application wants to manipulate the server’s FHIR data, it would send a request through the Service and then manipulate the \texttt{Json} property of the response. For more information about manipulating a server’s FHIR data, see Working with FHIR Data.

Your ObjectScript application can also validate a resource.

\textbf{Note:} Because the FHIR server’s Service is not, in most cases, used by Health Connect, the following methods of performing CRUD operations do not apply.

5.1 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:

\texttt{GET http://178.16.235.12:52783/fhirapp/namespace/fhir/r4/patient/1}

Using ObjectScript to access the same endpoint looks like:

\begin{verbatim}
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)
\end{verbatim}

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

\textbf{Note:} The first request to the server must instantiate the FHIR service by calling the \texttt{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.
5.2 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 = {}.%FromJSON("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.

5.3 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. For more information about working with this data, see FHIR Data and Dynamic Objects.

5.4 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, 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.
5.5 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.

5.6 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).%FromJSON("c:\localdata\myPatient.json")

// Validate the patient resource
set schema = #class(HS.FHIRServer.Schema).LoadSchema("R4")
set resourceValidator = #class(HS.FHIRServer.Util.ResourceValidator).%New(schema)
do resourceValidator.ValidateResource(patient)
```
6

Profiling FHIR

According to the FHIR® specification, FHIR is a “platform specification” that requires modification to be suitable for a healthcare implementation and purpose. Adapting FHIR resources, frameworks, and API for a specific healthcare purpose is known as Profiling. The InterSystems FHIR server supports profiling FHIR in the following ways:

- Capability Statement
- Extensions
- Search Parameters

**Important:** When profiling, never modify the base metadata files installed with your InterSystems product; create a custom metadata set that extends this base metadata. As an exception, modifying the capability statement does not require creating a custom metadata set.

### 6.1 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. This capability statement can be edited to make small changes like changing the server’s name and description, or to describe functional changes like which resources the server accepts or what FHIR operations are available. In InterSystems products, updating the capability statement consists of retrieving it from the server, editing it, posting it back to the server.

**Note:** Do not update the capability statement when adding custom search parameters. The capability statement is updated automatically when you update the custom metadata set that contains the new search parameters.

#### 6.1.1 Retrieving the Capability Statement

InterSystems strongly recommends retrieving the current capability statement from the server and modifying it rather than writing a new one. You can retrieve the capability statement resource with a REST client or programmatically. 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/namespace/fhir/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/namespace/fhir/r4/metadata
  ```

- To retrieve the capability statement programmatically, enter:
Once retrieved, the capability statement can be edited with an external editor or third-party tool. Capability statements retrieved programmatically must be converted from a dynamic object to a JSON file before modifying it.

### 6.1.2 Updating the Capability Statement

Once you have modified the capability statement, submit the revised version to the server programmatically from the InterSystems Terminal. In the following example, `/fhirapp/namespace/fhir/r4` is the endpoint's base url and `MyCapabilityStatement.json` is the revised version. The `{ }.%FromJson` method takes a JSON file and puts it into a dynamic object.

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

### 6.2 Extensions

The FHIR server accepts a resource with extensions as long as it is well-formed according to the FHIR syntax for extensions. For information about adding custom search parameters for an extension, see [Custom Search Paramters](#).

### 6.3 Custom Metadata Sets

By default, the installation process creates an endpoint based on the base metadata for a particular FHIR version. If you are planning to customize the FHIR server’s metadata, for example by creating custom search parameters, create a custom metadata set before beginning the installation process, even if you plan to wait to implement the customizations.

**Note:** Though the server’s capability statement is considered metadata, it is customized in a different manner. For more information, see [Modifying the Capability Statement](#).

To create a custom metadata set:

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```plaintext
   set $namespace = "FHIRNamespace"
   ```

2. Run the installation and configuration utility:
   ```plaintext
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 8) Create a custom metadata set.

4. Choose the base FHIR metadata for the FHIR version of your endpoint. The custom metadata set extends this base metadata.

5. Enter a name of the metadata set. This name appears when you are installing a new endpoint.

6. Enter a description of the metadata set.
7. Enter the directory that contains or will contain the custom metadata. For example, this is the directory that contains or will contain JSON files with custom search parameters. If the directory you specified for the custom metadata is empty, you must update the metadata set once you add files to the directory.

Now that you have created a custom metadata set, it appears as an option when installing an endpoint.

6.3.1 Updating a Custom Metadata Set

Whenever files are added to the custom metadata directory or the content of those files changes, you must update the metadata set so the server picks up the changes. The directory was specified when the custom metadata set was created. To update a custom metadata set:

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```
2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```
3. Choose option 9) Update a custom metadata set.
4. Choose the custom metadata set you are updating.
5. If desired, enter a new description of the custom metadata set.
6. If desired, enter a new directory that contains the custom metadata.
7. When you see the prompt Do you want to update the metadata cache?, enter one of the following:
   - If you have added new metadata to the directory, choose Y to rebuild the search tables and add the metadata to the capability statement. Be aware that this can be a lengthy process.
   - If you only changed the description of the metadata, choose N.

6.4 Custom Search Parameters

The search parameters that a client can use to retrieve resources from the FHIR server are defined by SearchParameter resources. In many cases, custom search parameters need to be added for custom extensions on resources.

To define custom search parameters, start by using an external editor or third-party tool like Forge to create JSON files that contain the new SearchParameter resources. Each JSON file must include a single bundle that contains one or more SearchParameter resources. You can define multiple files in which each bundle contains a single SearchParameter resource, or define a single file in which the bundle contains multiple SearchParameter resources.

The next step in the process depends on how the endpoint was installed:

- If the endpoint was installed using the base metadata rather than a custom metadata set, you will have to install a new endpoint. Create a custom metadata set that specifies the directory with the JSON files, and then install the new endpoint.
- If the endpoint was installed using a custom metadata set, place the JSON files in its custom metadata directory and update the metadata set. You are given the option of specifying a new directory for the JSON files when you update the metadata set.
- If the endpoint has not been installed yet, create a new metadata set and specify the directory with the JSON files.
Note: You do not need to manually edit the server’s capability statement when you add custom search parameters; the capability statement is automatically updated when you update the custom metadata set.
FHIR Operations

The FHIR® server supports 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. InterSystems IRIS for Health applications using the Resource Repository already support certain standard FHIR operations (see What is Supported? for a complete list). A FHIR server in Health Connect does not use the Resource Repository, so there are no default operations.

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

1. Subclass the FHIR server’s Interactions and InteractionsStrategy classes. If you are using the Resource Repository with InterSystems IRIS for Health, you want to subclass HS.FHIRServer.Storage.JSON.Interactions and HS.FHIRServer.Storage.JSON.InteractionsStrategy.

2. Create a subclass of HS.FHIRServer.API.OperationHandler. If you are using the Resource Repository that comes with InterSystems IRIS for Health, subclass HSFHIRServer.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.

7.1 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:

FHIRScopeOpOperationName

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.
FHIR Operations

- **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.

<table>
<thead>
<tr>
<th>Method name</th>
<th>REST client call to the operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FHIRSystemOpMyoperation</td>
<td><a href="http://fhirserver.org/fhir/$myoperation">http://fhirserver.org/fhir/$myoperation</a></td>
</tr>
<tr>
<td>FHIRTypeOpValidate</td>
<td><a href="http://fhirserver.org/fhir/Observation/$validate">http://fhirserver.org/fhir/Observation/$validate</a></td>
</tr>
<tr>
<td>FHIRInstanceOpEverything</td>
<td><a href="http://fhirserver.org/fhir/Patient/1/$everything">http://fhirserver.org/fhir/Patient/1/$everything</a></td>
</tr>
</tbody>
</table>

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:

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

### 7.2 Adding the Operation to Capability Statement

The **capability statement** of the FHIR server must include all of the operations that the server supports. If the FHIR server is using a custom capability statement, manually update the capability statement to include the new operations.

Most FHIR server use a capability statement that has been customized in some way. However, if the FHIR server is using the default capability statement provided with your InterSystems product, 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:

```csharp
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:
ClassMethod AddSupportedOperations(pMap As %DynamicObject)
{
    Do #class(HS.FHIRServer.MySuperclass.Validate).AddSupportedOperations(pMap)
    Do pMap.tSet("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 re-generate the capability statement:

   **Important**: Do not use the command-line utility to update the capability statement if the FHIR server is using a custom capability statement because the custom one will be overwritten. In these cases, manually update the capability statement with the new operations.

   a. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:

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

   b. Run the installation and configuration utility:

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

   c. Choose option 5) Update the CapabilityStatement Resource.

   d. Select the endpoint you are configuring.

   e. Specify whether you want the endpoint to be read-only.

   f. Confirm your selection.
8

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.

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

8.1 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.

8.1.1 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 a 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 security, see the Security Administration Guide.

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 Creating or Editing 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 Creating 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 Giving 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 Assigning Users or Roles to the Current Role.

### 8.1.1.1 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. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```

2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 3) **Configure a FHIRServer Endpoint.**

4. Choose the endpoint of the FHIR server that you are configuring.

5. Accept the default configuration options until you get to the **RequiredResource:** prompt.

6. At the **RequiredResource:** prompt, enter the name of the security resource that controls access to the FHIR server.

7. Continue with the prompts and save your changes.

### 8.2 OAuth 2.0 Authorization

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 the OAuth 2.0 authorization server. 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. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```

2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 3) **Configure a FHIRServer Endpoint.**

4. Choose the endpoint of the FHIR server that you are configuring.

5. Accept the default configuration options until you get to the **OAuthClientName:** prompt.

6. At the **OAuthClientName:** prompt, enter the **Application Name** of the resource server as defined in the Management Portal.
7. Continue with the prompts and save your changes.

**CAUTION:** The default FHIR server does not respect the scopes passed in an OAuth access token; by default, these scopes are ignored. InterSystems strongly recommends that you use InterSystems IRIS for Health to create an application or use an API management tool, such as InterSystems API Manager, that enforces scopes consistent with the FHIR standard to provide resource-based and identity-based access control. Failing to write custom code or implement another mechanism to handle scopes can result in privacy violations where FHIR clients can access or perform functions on data beyond or inconsistent with the scope that was specified in the access token.

### 8.3 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. By setting the server’s DebugMode configuration to 4, you can allow all FHIR requests to reach the server, temporarily ignoring authentication and authorization strategies. For more information about setting the DebugMode to allow unauthenticated requests, see [Debug Mode](#).
Debugging and Maintaining a FHIR Server

9.1 Debugging the FHIR Server

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

9.1.1 Debug Mode

The installation and configuration utility can put the FHIR server into debug mode during development. To set the debug mode:

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   ```
   Where FHIRNamespace is the name of the endpoint’s namespace.

2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```

3. Choose option 3) Configure a FHIRServer Endpoint.

4. Accept the default options until you reach the option DebugMode.

5. Enter a value to set the debug mode for the endpoint. The possible values are:

<table>
<thead>
<tr>
<th>DebugMode Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable all DebugMode options.</td>
</tr>
<tr>
<td>1</td>
<td>Include traceback, which responds to a FHIR request by sending a stack trace in an OperationOutcome resource.</td>
</tr>
<tr>
<td>2</td>
<td>Creates a new instance for the FHIR Service on each REST request.</td>
</tr>
<tr>
<td>4</td>
<td>Allow unauthenticated requests.</td>
</tr>
</tbody>
</table>

To enable more than one debug mode, enter a value that is the sum of multiple option values. For example, to include traceback (option 1) and allow unauthenticated requests (option 4), specify 5 as the DebugMode value.
To disable debugging, run the configuration utility again, choosing 0 as the DebugMode.

9.2 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.

9.2.1 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:
   ```
   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:
   ```
   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:
   ```
   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`.

9.2.1.1 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.Json.Interactions|SQL|Parameters: (2)"

9.2.1.2 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:

```
kill ^FSLogChannel
```

9.2.2 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:
   ```
   set ^%ISCLOG=5
   set ^%ISCLOG("Category","HSFHIR")=5
   set ^%ISCLOG("Category","HSFHIRServer")=5
   ```

9.2.2.1 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.

9.2.2.2 Disabling Logging

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

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

9.3 Maintaining the FHIR Server

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

To stop and re-start an endpoint:

1. From the InterSystems Terminal, change to the FHIR server’s namespace. For example:
   ```
   set $namespace = "FHIRNamespace"
   
   Where FHIRNamespace is the name of the endpoint’s namespace.
   ```
2. Run the installation and configuration utility:
   ```
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()
   ```
3. Choose option 3) Configure a FHIRServer Endpoint.
4. Select your endpoint.
5. At the **Endpoint enabled** option, specify `n` to stop the endpoint. If you want to re-enable the endpoint, specify `y`. 
10 Resource Repository

The Resource Repository is the default InteractionsStrategy for InterSystems IRIS for Health, 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. Of course, you can extend the Resource Repository's classes, HS.FHIRServer.Storage.Json.Interactions and HS.FHIRServer.Storage.Json.InteractionsStrategy, to refine how the FHIR server handles the FHIR data. The Resource Repository also comes with default FHIR operations in the HS.FHIRServer.Storage package.

**Important:** The Resource Repository is not supported in Health Connect. Though you can create custom Interactions and InteractionsStrategy classes for a FHIR server in Health Connect, in most cases you are accepting FHIR into an interoperability production for other purposes. For more information, see FHIR Productions.

For more information about programmatically retrieving data from or storing data in the Resource Repository, see ObjectScript Applications. For more information about working with FHIR data once it is retrieved from the Resource Repository, see Working with FHIR Data.

10.1 What is Supported?

When using the Resource Repository strategy provided with InterSystems IRIS for Health, the FHIR server supports the following interactions and operations. If your custom FHIR server extends the Resource Repository, it also supports these interactions and operations.

10.1.1 Interactions

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. If an interaction is not listed, it is not supported.
<table>
<thead>
<tr>
<th>Interaction</th>
<th>Limitations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>Fully supported, including conditional create.</td>
</tr>
<tr>
<td>read</td>
<td>The _summary and _elements parameters are not supported. Conditional read is not supported.</td>
</tr>
<tr>
<td>vread</td>
<td>The _summary and _elements parameters are not supported. Conditional read is not supported.</td>
</tr>
<tr>
<td>update</td>
<td>Fully supported, including conditional update.</td>
</tr>
<tr>
<td>delete</td>
<td>Supported, but conditional delete is not supported.</td>
</tr>
<tr>
<td>history</td>
<td>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.</td>
</tr>
<tr>
<td>batch</td>
<td>Fully supported</td>
</tr>
<tr>
<td>transaction</td>
<td>Circular references within the bundle are not supported.</td>
</tr>
<tr>
<td>search</td>
<td>Supported with some limitations. For details, see Search Interaction.</td>
</tr>
</tbody>
</table>

**10.1.1.1 Search Interaction**

FHIR clients use the search interaction to retrieve resources from the Resource Repository. This section summarizes the default support for the search interaction when the FHIR server is using or extending the Resource Repository.

**General Limitations**

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

- Does not support searching across multiple resource types. For example GET [base]?_id=1 is not supported.
- You cannot perform a search on all resource types within a compartment. For example, you cannot search for [base]/Patient/10000001/?_id=008. Therefore, searches within the context of a compartment must specify a resource type in that compartment. For example, you can use [base]/Patient/10000001/Observation to return all Observations in the specific patient’s compartment or [base]/Patient/10000001/Observation?status=final to search for a subset of Observations within the compartment. If you want to retrieve a Patient’s entire compartment, use the $everything operation (for example, [base]/Patient/10000001/$everything).

**Search Parameter Types**

Each search parameter has a search parameter type that determines how the parameter behaves. The following search parameter types are supported. If a search parameter type is not listed, it is not supported.
Parameters

The following standard search parameters are supported by the FHIR server when retrieving resource from the Resource Repository. If a parameter is not listed, it is not supported.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Limitations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
<tr>
<td>_lastUpdated</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
<tr>
<td>_tag</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
<tr>
<td>_profile</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
<tr>
<td>_security</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
<tr>
<td>_source</td>
<td>Fully supported.</td>
</tr>
<tr>
<td>_has</td>
<td>Fully supported as described in the FHIR specification</td>
</tr>
</tbody>
</table>

Modifiers

Modifiers can be added to the end of a parameter to affect the results of the search. The following modifiers are supported.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Limitations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>:exact</td>
<td>Supported for strings</td>
</tr>
<tr>
<td>:contains</td>
<td>Supported for strings</td>
</tr>
<tr>
<td>:above</td>
<td>Supported for uri</td>
</tr>
<tr>
<td>:below</td>
<td>Supported for uri</td>
</tr>
<tr>
<td>:type</td>
<td>Supported for references</td>
</tr>
</tbody>
</table>

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 exactly 100. The following prefixes are supported.
### Search Result Parameters

Search result parameters help manage the resources returned by a search. The following search result parameters are supported. If a parameter is not listed, it is not supported.

<table>
<thead>
<tr>
<th>Search result parameter</th>
<th>Limitations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_sort</code></td>
<td>Fully supported as described in the <a href="https://spec.healthinteroperability.org/">FHIR specification</a></td>
</tr>
<tr>
<td><code>_count</code></td>
<td>Fully supported as described in the <a href="https://spec.healthinteroperability.org/">FHIR specification</a></td>
</tr>
<tr>
<td><code>_summary</code></td>
<td>Supports <code>_summary=count</code> only. For details, see the <a href="https://spec.healthinteroperability.org/">FHIR specification</a>.</td>
</tr>
<tr>
<td><code>_include</code></td>
<td>Fully supported as described in the <a href="https://spec.healthinteroperability.org/">FHIR specification</a></td>
</tr>
<tr>
<td><code>_revinclude</code></td>
<td>Fully supported as described in the <a href="https://spec.healthinteroperability.org/">FHIR specification</a></td>
</tr>
</tbody>
</table>

### 10.1.2 Operations

For InterSystems IRIS for Health using or extending the default Resource Repository, the following operations are supported:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Limitations/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$everything</code></td>
<td>Fully supported.</td>
</tr>
<tr>
<td><code>$validate</code></td>
<td>The validation modes (create, update, delete) are supported. Validation by profile is not supported. When a FHIR <code>$validate</code> request includes a resource payload, the resource may be enclosed in a Parameters resource.</td>
</tr>
</tbody>
</table>

### 10.2 Migrating from Legacy Resource Repository

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

1. Open the InterSystems Terminal and navigate to the namespace of your legacy FHIR server.
2. Create a STU3 endpoint that will work with the data in the existing Resource Repository.
3. Run the installation and configuration utility:
   do ##class(HS.FHIRServer.ConsoleSetup).Setup()


5. Select the STU3 endpoint and confirm the migration.
11

FHIR Clients

InterSystems products come with FHIR® client technology that can be used to send a FHIR request to an external FHIR server over HTTP and process the response. For more information, see the legacy FHIR books that are available at InterSystems Legacy Documentation.
For details about using legacy FHIR® technology, see the legacy FHIR books that are available at InterSystems Legacy Documentation.