Oracle® Healthcare Data Repository

FHIR User's Guide

Release 8.1.4
F73507-01
September 2023

Oracle Healthcare Data Repository FHIR User's Guide, Release 8.1.4

F73507-01

Contents

12 Other Features

13 Partitioning

14 Terminology

Preface

This preface contains the following sections:

Documentation accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Getting Started

This section provides information about the platform requirements for setting up an Oracle Healthcare Data Repository HL7 FHIR server.

Introduction

Fast Healthcare Interoperability Resources (FHIR) is a standard for healthcare data exchange that is published by HL7 (www.hl7.org/fhir/).

Oracle Healthcare Data Repository (HDR) Release 8.1.4 supports the HL7 FHIR specification versions R4 (4.0.1), R4B (4.3.0) and R5. The FHIR server module is distributed as a web application, which can be deployed to standard web containers such as WebLogic. FHIR resources are exposed as a set of REST APIs that can be accessed by REST-based applications.

Platform Requirements

The following software is required for Oracle Healthcare Data Repository and the FHIR server module:

- Operating System: Oracle Enterprise Linux 8.x or 9.x (64 bit)
- Oracle Database 19c Release 1 (12.1.0.2.0) or Release 2 (12.2.1.2.0) or Release 19c. Download from the Oracle Software Delivery Cloud at https://edelivery.oracle.com.
- WebLogic Server 14.1.1.0 with the Coherence option. Download from the Oracle Software Delivery Cloud at https://edelivery.oracle.com.
- JDK (Java Development Kit) 11u17 and later. Download from My Oracle Support.

Download the Oracle Healthcare Data Repository 8.1.4 patch set from My Oracle Support at https://support.oracle.com. Refer to the Oracle Healthcare Data Repository 8.1.4 Release Notes (https://support.oracle.com/epmos/faces/DocumentDisplay?id=2907176.1).

2
Installation

Refer to the Oracle Healthcare Data Repository 8.1.4 Release Notes (https://support.oracle.com/epmos/faces/DocumentDisplay?id=2907176.1) for information on the prerequisites and steps to install the HDR-FHIR module.

3
HDR FHIR Server Architecture

Healthcare functional domains (administrative, clinical, and financial) and core services are exposed as RESTful endpoints.

The client application that uses the HDR FHIR RESTful APIs must properly configure the system in a secure environment to avoid unauthorized access of those APIs. To access protected FHIR APIs, the client application must provide a valid access token. For more information on this, see the HDR 8.1 Secure Configuration Guide and Secure Development Guide.

The FHIR Server can be deployed in a single or multi-node cluster as illustrated below:

Figure 3-1 Single Node

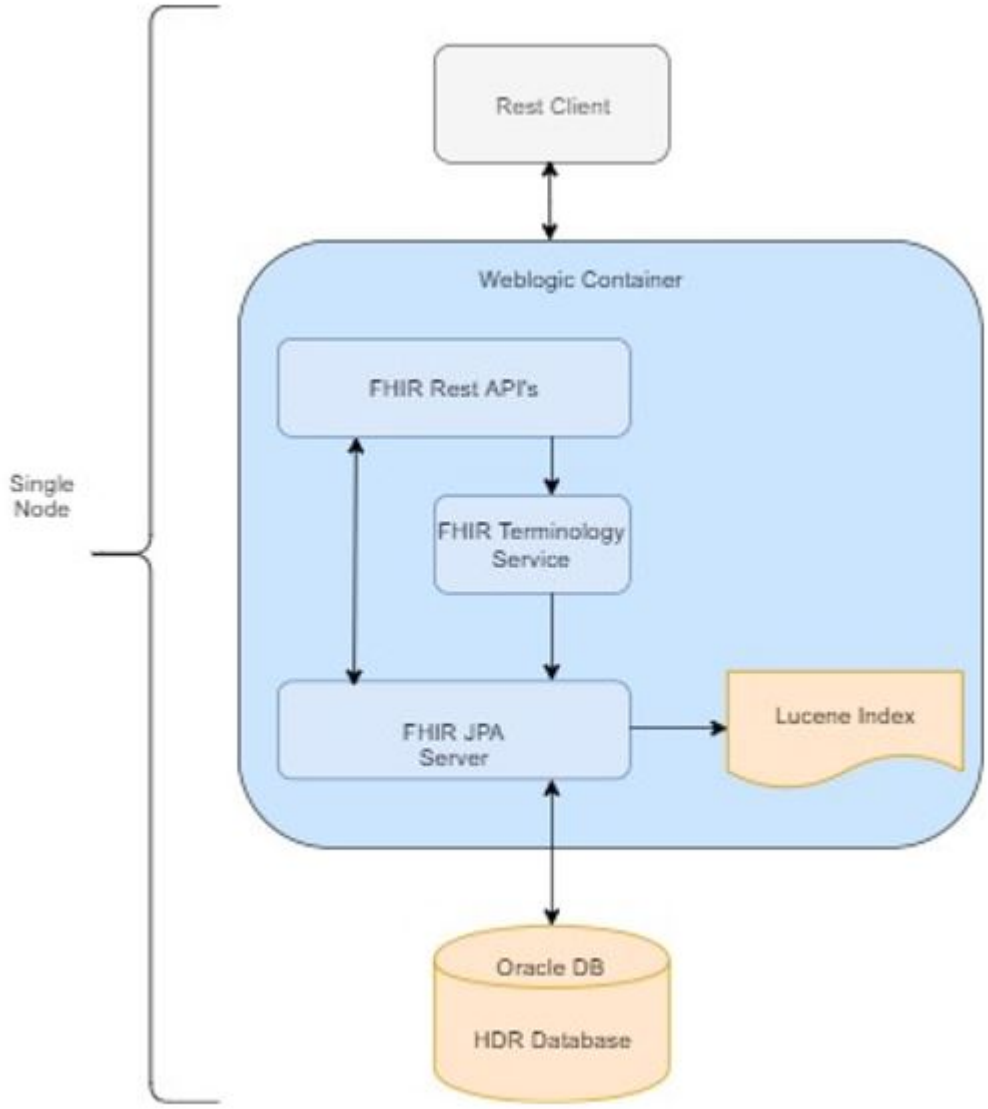
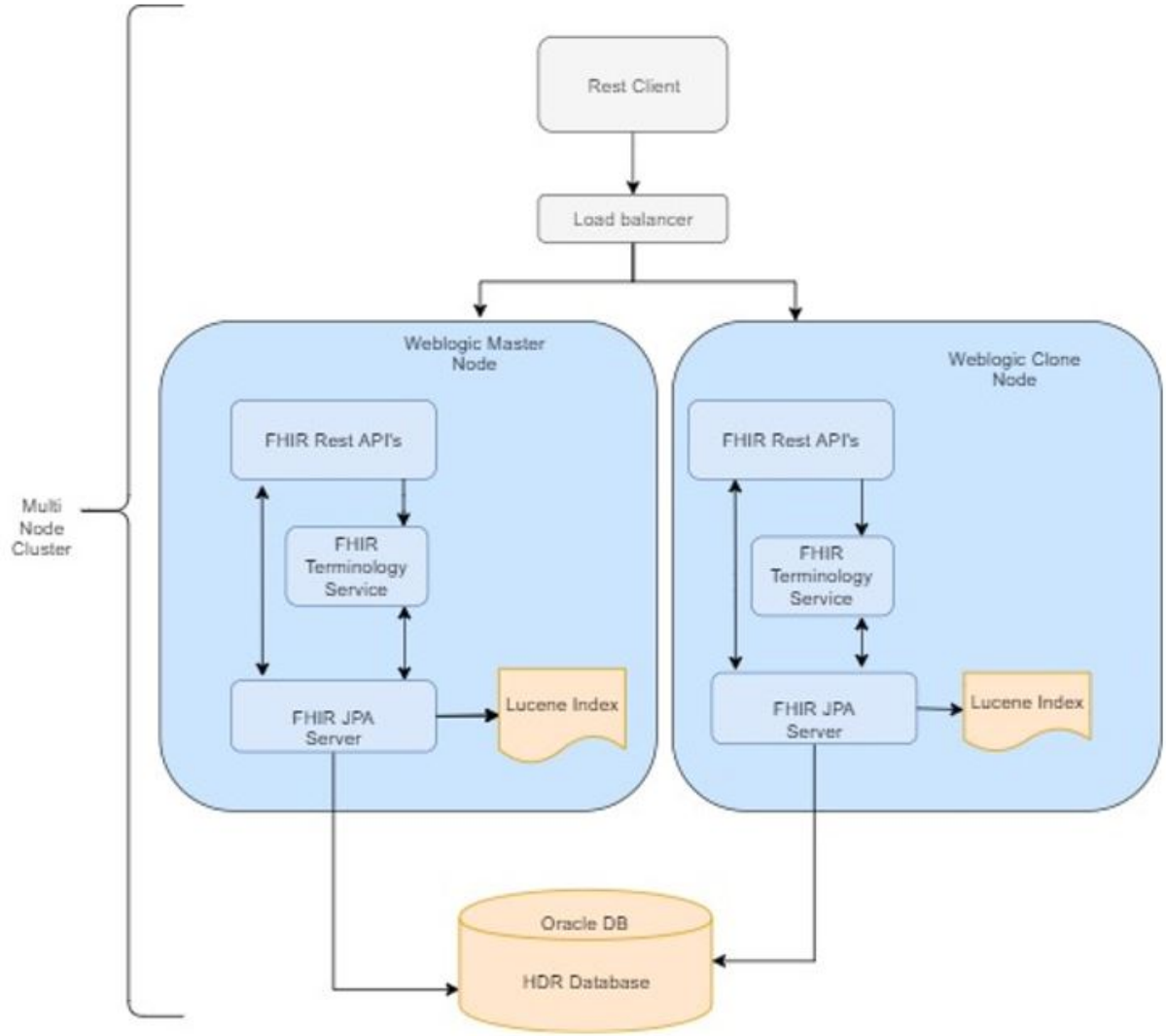


Figure 3-2 Multi-Node Cluster



4

Deployment Details

This section describes the details of the HDR FHIR Server deployment.

Install files

The HDR FHIR Server module is packaged as a deployable web application (.war) file. The war file is distributed along with the HDR 8.1.4 patch set. Once installed, FHIR resources are exposed as a set of REST endpoints.

The other component, the FHIR command line tool, is available in the HDR 8.1.4 patch and copy them under HDR HOME directory on the middle tier. Refer to FHIR Command-Line Utility section for more details.

FHIR Server Base URL and REST Endpoints

FHIR REST APIs can be accessed using the base URL as show below:

http://HOSTNAME:PORT/oracle-fhir-server/fhir

A specific resource can be accessed using the URL format:

<BASE_URL>/<resourceName>

For example, to access the ‘Patient' resource, use the following URL:

http://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient

For the complete list of resources and their corresponding URLs, see Working with FHIR REST APIs .

Configuration Files

Runtime behavior and logging are controlled using configuration files.

Properties File

The runtime behavior of the FHIR JPA server can be managed using the hdr_fhir.yaml file: <HDR_DOMAIN>/config/fhir/hdr_fhir.yaml

Example 4-1 hdr_fhir.yaml file

```
#Adds the option to go to eg. http://localhost:8080/actuator/health for seeing
the running configuration
#see https://docs.spring.io/spring-boot/docs/current/reference/html/
actuator.html#actuator.endpoints
management:
  endpoints:
    web:
      exposure:
        include: "health,prometheus"
spring:
  main:
    allow-circular-references: true
    #allow-bean-definition-overriding: true
  flyway:
    enabled: false
    check-location: false
    baselineOnMigrate: true
#########
# datasrouce - specify datasource here(name of the datasource configured in your
app server)
#########
  datasource:
    jndi_name: jdbc/OrclFhirDataSource
  jpa:
    properties:
      hibernate.format_sql: false
      hibernate.show_sql: false
      #Hibernate dialect is automatically detected except Postgres and H2.
      hibernate.hbm2ddl.auto: none
      hibernate.jdbc.batch_size: 20
      hibernate.cache.use_query_cache: false
      hibernate.cache.use_second_level_cache: false
      hibernate.cache.use_structured_entries: false
      hibernate.cache.use_minimal_puts: false
  ###    These settings will enable fulltext search with lucene or elastic
      hibernate.search.enabled: true
  ### lucene parameters
      hibernate.search.backend.type: lucene
      hibernate.search.backend.analysis.configurer:
ca.uhn.fhir.jpa.search.HapiHSearchAnalysisConfigurers$HapiLuceneAnalysisConfigure
r
      hibernate.search.backend.directory.type: local-filesystem
      hibernate.search.backend.directory.root: target/lucenefiles
      hibernate.search.backend.lucene_version: lucene_current
  ### elastic parameters ===> see also elasticsearch section below <===
#      hibernate.search.backend.type: elasticsearch
#      hibernate.search.backend.analysis.configurer:
ca.uhn.fhir.jpa.search.HapiHSearchAnalysisConfigurers$HapiElasticAnalysisConfigur
er
hapi:
  fhir:
    ### This is the FHIR version. Choose between, DSTU2, DSTU3, R4 or R5
    fhir_version: R4
    ### This enables the swagger-ui at <BaseUrl>/swagger-ui/index.html as well
as the <BaseUrl>/api-docs
    openapi_enabled: true

    ### enable to use the ApacheProxyAddressStrategy which uses X-Forwarded-*
headers
```

```
    ### to determine the FHIR server address
    #   use_apache_address_strategy: false
    ### forces the use of the https:// protocol for the returned server address.
    ### alternatively, it may be set using the X-Forwarded-Proto header.
    #   use_apache_address_strategy_https: false
    ### enables the server to host content like HTML, css, etc. under the url pattern
of /static/**
    ### the deepest folder level will be used. E.g. - if you put file:/foo/bar/bazz as
value then the files are resolved under /static/bazz/**
    #staticLocation: file:/foo/bar/bazz
    ### enable to set the Server URL
    #    server_address: http://hapi.fhir.org/baseR4
    #    defer_indexing_for_codesystems_of_size: 101
##########################
# Repository Validation(resource validation against Implementation Gudie)
# IG can be loaded from remote url(http:), local file(file:) or classpath(classpath:)
# If no url specified, IG is downloaded from central package respository(https://
packages.fhir.org/)
# note: url can be either http:, https:, classpath:, or file:
##########################
    load_ig_on_server_startup: false
    install_transitive_ig_dependencies: false
    implementationguides:
#      australia_core:
#        name: hl7.fhir.au.base
#        version: 4.1.0
#        url:
      us_core:
        name: hl7.fhir.us.core
        version: 6.0.0
#        url:
    #    supported_resource_types:
    #      - Patient
    #      - Observation
###############
# Repository validation enable/disable
###############
    validation_repository_enabled: false
    ################################################
    # Allowed Bundle Types for persistence (defaults are: COLLECTION,DOCUMENT,MESSAGE)
    ################################################
    allowed_bundle_types:
COLLECTION,DOCUMENT,MESSAGE,TRANSACTION,TRANSACTIONRESPONSE,BATCH,BATCHRESPONSE,HISTORY
,SEARCHSET
    allow_cascading_deletes: true
    allow_contains_searches: true
    allow_external_references: true
    allow_multiple_delete: true
    allow_override_default_search_params: true
    auto_create_placeholder_reference_targets: false
    #    cr_enabled: true
    #    ips_enabled: false
    default_encoding: JSON
    #    default_pretty_print: true
    default_page_size: 20
    delete_expunge_enabled: true

    enable_index_missing_fields: false
    #    enable_index_of_type: true
    #    enable_index_contained_resource: false
    ###  !!Extended Lucene/Elasticsearch Indexing is still a experimental feature,
```

```
expect some features (e.g. _total=accurate) to not work as expected!!
    ###  more information here: https://hapifhir.io/hapi-fhir/docs/server_jpa/
elastic.html
    advanced_lucene_indexing: false
    #    bulk_export_enabled: false
    #    bulk_import_enabled: false
    #    enforce_referential_integrity_on_delete: false
    # This is an experimental feature, and does not fully support _total and
other FHIR features.
    enforce_referential_integrity_on_delete: false
    enforce_referential_integrity_on_write: false
    etag_support_enabled: true
    expunge_enabled: true
    #    client_id_strategy: ALPHANUMERIC
    fhirpath_interceptor_enabled: false
    filter_search_enabled: true
    graphql_enabled: true
    narrative_enabled: false
    #    mdm_enabled: true
    #    local_base_urls:
    #      - https://hapi.fhir.org/baseR4
    mdm_enabled: false
    #    partitioning:
    #      allow_references_across_partitions: false
    #      partitioning_include_in_search_hashes: false
    cors:
      allow_Credentials: true
      # These are allowed_origin patterns, see: https://docs.spring.io/spring-
framework/docs/current/javadoc-api/org/springframework/web/cors/
CorsConfiguration.html#setAllowedOriginPatterns-java.util.List-
      allowed_origin:
        - '*'

    # Search coordinator thread pool sizes
    search-coord-core-pool-size: 20
    search-coord-max-pool-size: 100
    search-coord-queue-capacity: 200

    # comma-separated package names, will be @ComponentScan'ed by Spring to
allow for creating custom Spring beans
    #custom-bean-packages:

    # comma-separated list of fully qualified interceptor classes.
    # classes listed here will be fetched from the Spring context when combined
with 'custom-bean-packages',
    # or will be instantiated via reflection using an no-arg contructor; then
registered with the server
    #custom-interceptor-classes:

    # Threadpool size for BATCH'ed GETs in a bundle.
    #    bundle_batch_pool_size: 10
    #    bundle_batch_pool_max_size: 50

    logger:
      error_format: 'ERROR - ${requestVerb} ${requestUrl}'
      format: >-
        Path[${servletPath}] Source[${requestHeader.x-forwarded-for}]
        Operation[${operationType} ${operationName} ${idOrResourceName}]
        UA[${requestHeader.user-agent}] Params[${requestParameters}]
        ResponseEncoding[${responseEncodingNoDefault}]
      log_exceptions: true
```

```
      name: fhirtest.access
    max_binary_size: 104857600
    max_page_size: 200
    retain_cached_searches_mins: 60
    reuse_cached_search_results_millis: -1
#############
#Resource validation configuration
#############
    validation:
      requests_enabled: false
      responses_enabled: false
# if remote terminology validation is set to true, make sure that request validation
is also enabled(requests_enabled property).
      remote_terminology_service_enabled: false
      remote_terminology_server_base_url: https://lforms-fhir.nlm.nih.gov/baseR4
#############
    binary_storage_enabled: true
############
#below property decides whether to store the resource as plain text in RES_TEXT_VC
column.
#if the resource size is below the size as set in the below property, it goes
RES_TEXT_VC column,
#if its larger than the property set value, it goes to RES_TEXT as compressed blob.
# summary -  the resource is stored in either RES_TEXT_VC or RES_TEXT dependeing on
the size set below.
    inline_resource_storage_below_size: 0
#    bulk_export_enabled: true
    subscription:
      resthook_enabled: false
      websocket_enabled: false
#      email:
#       from: some@test.com
#       host: localhost
#        port: 22
#        username:
#        password:
#        auth:
#        startTlsEnable:
#        startTlsRequired:
#        quitWait:
#    lastn_enabled: true
#    store_resource_in_lucene_index_enabled: true
###  This is configuration for normalized quantity search level default is 0
###   0: NORMALIZED_QUANTITY_SEARCH_NOT_SUPPORTED - default
###   1: NORMALIZED_QUANTITY_STORAGE_SUPPORTED
###   2: NORMALIZED_QUANTITY_SEARCH_SUPPORTED
    normalized_quantity_search_level: 0
    metadata:
      implementation_description: Oracle FHIR Server
      software_name: Oracle FHIR Server
      publisher: Oracle Corporation
      software_version: 8.1.4

    resource_count_enabled: false
    resource_compression_enabled: true
    response_highlighter_enabled: true
    allow_placeholder_references: true
    response_timing_log_enabled: false

############
# for extracting unique request id from the incoming REST HTTP request and log the id
```

```
into server log file.
    enable_http_header_logging: true
    request_id_key_http_header: X-Request-ID
############

############
# for controlling scheduled job - resourcecountcache
    enable_resource_count_scheduling_job: false
    resource_count_cache_expiry_time_in_minutes: 240
    resource_count_job_scheduling_time_in_minutes: 10
############

    audit:
      enabled: true
      datastore_type: DB
      savemessagepayload_enabled: false
      standard: CUSTOM

elasticsearch:
  debug:
    pretty_print_json_log: false
    refresh_after_write: false
  enabled: false
  required_index_status: YELLOW
  rest_url: 'localhost:9200'
  protocol: 'http'
  schema_management_strategy: CREATE
  username: SomeUsername
  password: SomePassword

oauth:
  enabled: false
  token_issuer: "https://dev-t9brtcqa.auth0.com/"
  token_audience: "https://fhir-hdr.auth.com/api/v2/"
  token_alg: HS256
  scopes: "fhir.admin,fhir.users,fhir.users.restricted"
  fhir.admin: read,create,update,delete
  fhir.users: read,create
  fhir.users.restricted: read
  #note: add more if needed
  #scopes and allowed fhir resources
  fhir.admin.allowedapis: ALL
  fhir.users.allowedapis: ALL
  #ResearchStudy
  fhir.users.restricted.allowedapis: Patient, Observation, AllergyIntolerance,
Medication, Condition, Procedure, Immunization
```

Log Configurations

HDR FHIR has several logging mechanisms that each serve a distinct purpose. These mechanisms are described in the table below. Oracle HDR FHIR uses the log4j2 logging framework to emit these logs. These logs are generated at runtime by all components of the FHIR. The location of the log4j2.properties is: <HDR_DOMAIN>/ config/fhir/ log4j2.properties.

Table 4-1 Log files

Log	File	Purpose	Retention
Application log	<HDR_DOMAIN>/logs/hdr-fhir.log	Application Logging is a traditional file-based log of events and internal processing details of Oracle HDR FHIR. These logs are useful for troubleshooting. Application logs can be enabled and disabled at runtime by modifying the log4j2 properties file.	Logs are rotated and compressed on a Time basis, although this can be configured using the log4j2.properties file.
Audit Log	<HDR_DOMAIN>/logs/audit-hdr-fhir.log	The audit log is intended to record actions taken by users. This log can be enabled or disabled using “audit.enabled" property defined in the hdr_fhir.properties file.	Logs are rotated and compressed on a Time basis, although this can be configured using log4j2.properties file.

5

Using the OAuth 2.0 protected API

HDR FHIR offers a suite of REST APIs implemented per HL7 FHIR specification and secured using the OAuth 2.0 security framework. This article outlines the steps needed for clients/ admin users to obtain a secure access token from HDR's OAuth Server and use the access token to invoke the HDR FHIR REST APIs.

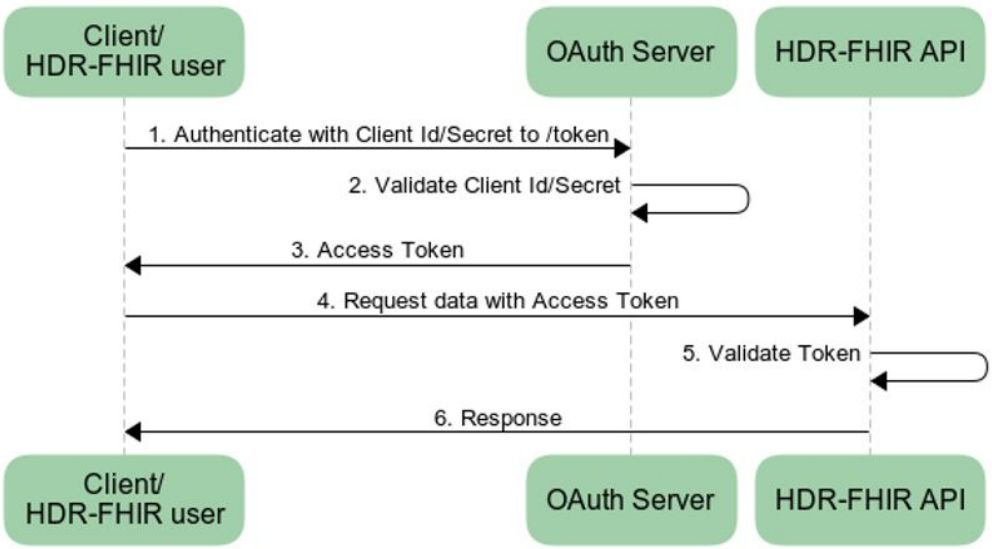
Prerequisites

Prerequisites for using the OAuth protected API are as follows:

- HDR FHIR is successfully registered with an OAuth Server as a Resource Server (that is protecting its endpoints).
- A client representing the HDR FHIR API admin user has been registered with OAuth Server as an OAuth Client and is authorized to invoke HDR FHIR APIs.

How It Works

Figure 5-1 Access Token Process



1. Client application or user authenticates with the OAuth Server (at say, the /ms_oauth/oauth2/endpoints/tokens endpoint) using the client ID and secret. The client ID and secret would have been obtained at the time of registering the OAuth client with OAuth Server.
2. OAuth Server validates the client ID and secret.
3. OAuth Server responds with an Access Token.
4. Client application or user uses the Access Token to call an HDR FHIR API.
5. HDR FHIR server intercepts the request and validates the Access Token.
6. HDR FHIR API responds with requested data.

Obtaining the Access Token from the OAuth Server

The client/user can ask the OAuth Server for tokens for any of the authorized applications by issuing the following API call:

```
curl --request POST \
  --url https://example.oauthserver.com/ms_oauth/oauth2/endpoints/tokens \
  --header 'content-type: application/json' \
  --data
'{"client_id":"CqwUDq2VQ6AH416sf7n42CZ2rNyElkDW","client_secret":"iA6bJ9OQ-
tMWhVNUZylx6Km1_9tMuxVyKC4xNfWtPye72MjXyC3f1GJ38ttQ0oH9","audience":"hdr_fhir_api
","grant_type":"client_credentials"}'
```

In this example, client_id and client_secret are assigned random representative values. You should change these values with the actual client Id and secret, obtained after registering the client with OAuth Server.

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsI........N7KT4ig",
  "token_type": "Bearer",
  "expires_in":600
}
```

You can now extract the access_token property value from the response to make authorized requests to your API.

Calling the HDR FHIR API with an Access Token

You can use this bearer token with an Authorization Header in your request to obtain authorized access to the HDR FHIR API.

```
curl --request GET \
  --url  http://<SERVER BASE URL>/fhir/Medication \
  --header 'accept: application/json' \
  --header 'authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsI........N7KT4ig'
```

Error Messages

Here is a list of a few common OAuth-related error messages that can be thrown by HDR FHIR APIs and the associated remediation steps.

Table 5-1 OAuth-Related Error Messages

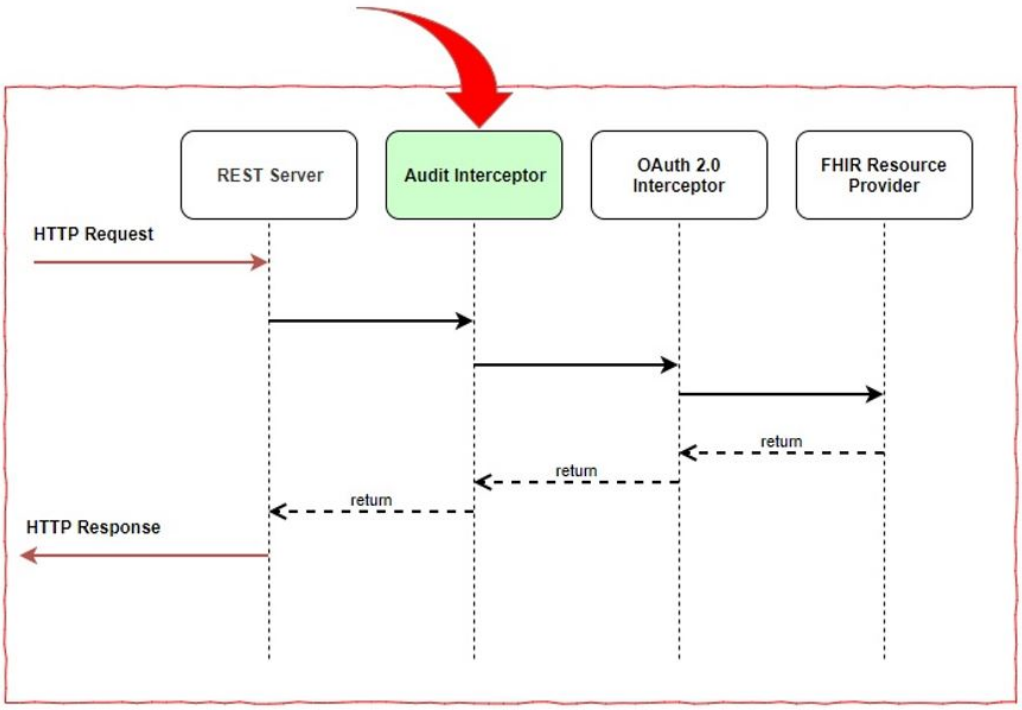
HTTP Status Code	Message	Meaning	Remediation
401	BAD_TOKEN: Invalid Algorithm. Algorithm is empty or not supported.	Signature algorithm is empty or not supported by the FHIR server.	Recommended algorithm is RS256. Make sure JWT header contains - "alg": "RS256".
200, 201	--	Success.	Authentication was successful. Operation was successful.
401	BAD_TOKEN: Invalid JWT token. Bad claims. Expired JWT	Unauthorized - expired OAuth token sent in request.	Current access token has expired. Obtain a fresh access token from OAuth Server and use it.
401	BAD_TOKEN: Invalid JWT token. Token is null or empty.	Unauthorized - no OAuth token sent in request.	Obtain a valid access token from OAuth Server. Pass it in request as bearer token in HTTP Auth header.
401	<Other error messages that start with "BAD_TOKEN: Invalid JWT token" >	Unauthorized - reason to be investigated.	Contact HDR administrator with the error message for further assistance.
401	BAD_TOKEN: Invalid JWT token. Bad claims. Invalid 'aud' attribute. Expected audience '<correct_audience>' does not exist in audience '<incorrect_audience>'	Unauthorized - token sent has incorrect audience value specified.	Ensure that you are using a correct audience value while requesting access token from OAuth Server.

6
Auditing

This module is responsible for collecting and storing audit data from incoming REST request and response. Key details such as user id, IP address, resource name, HTTP request type and request URL etc. are collected from the incoming request and stored in a secure location. Audit records can be stored either in a database table or in a file.

Audit Interceptor Execution Flow

Figure 6-1 Audit Interceptor Execution Flow



The audit module (rendered in green) design follows an interceptor pattern as shown in the above flow diagram. Here, incoming and outgoing REST API transactions are intercepted for extracting audit data elements.

Once the data is extracted, audit information goes to either a database table or a file (depending on storage settings defined in the FHIR server configuration file).

Audit Record Format

Audit record data format is as shown below.

Audit data element	Description
AUDIT_ID	Unique identifier for audit record
USER_ID	User ID
RESOURCE_NAME	FHIR resource name
HTTP_REQ_TYPE	HTTP request type - GET, POST, and so forth
REQUEST_URL	Incoming request URL
HTTP_RES_CODE	HTTP response code - 200, 201, 500, ...
SOURCE_IP_ADDRESS	Source system IP address
PROCESSING_TIME_MILLIS	Time taken to complete REST request
REQUEST_PAYLOAD	Payload
RESPONSE_PAYLOAD	Response payload
EVENT_TIMESTAMP	Timestamp
ATNA_AUDITEVENT	Audit record in the form of AuditEvent json

Settings

Audit service functionality can be controlled using a configuration file. The file is located at <HDR_DOMAIN>/config/fhir/hdr_fhir.yaml.

For example, if there is a requirement to store message payload as part of an audit record, change “savemessagepayload_enabled" property to true. Other important entries in the properties file is as shown below.

```
#audit enabled - true or false
audit:
      enabled: true

   #audit storage type - FILE or DB
      datastore_type: DB
      standard: CUSTOM
```

If ‘FILE' is selected as the storage type, audit data goes to a file named audit-hdr-fhir.log.

If ‘DB' is selected as the storage type, audit data goes to a table called OHF_HDR_FHIR_AUDIT. Refer to the FHIR eTRM document for more information about the Audit table.

```
#collect request/response payload message - true or false

savemessagepayload_enabled=false
```

7

FHIR Command-Line Utility

The HDR FHIR command-line interface (CLI) tool is a standalone command-line tool distributed with HDR 8.1.4 that contains a number of commands to ingest terminology data provided by the LOINC and SNOMED organizations.

The current version of the HDR FHIR command-line interface tool supports LOINC and SNOMED terminology data to be loaded into the HDR FHIR repository.

The tool is distributed in the form of zip file (hdr-fhir-cli-app-8.1.0-SNAPSHOT.zip) and is located in the HDR 8.1.4 patch.

Extract it into a directory where you will keep it, and add this directory to your path.

The zip file **hdr-fhir-cli-app-8.1.0-SNAPSHOT.zip** contains the following files:

File	Description
lib	All runtime executable jar file.
hdr-fhir-cli.sh	Linux script to manage the command line tool.

Prerequisites

The following prerequisites are required to use the command-line utility:

- Install the JDK 11
- Set the environment variable JAVA_HOME
- Extract the hdr-fhir-cli-app-8.1.0-SNAPSHOT.zip file into the middle tier
- Set the environment variable CLI_HOME to the location where hdr-fhir-cli-app-8.1.0-SNAPSHOT.zip file is extracted

Commands

The HDR FHIR command line tool has a number of supported functions, called commands. Each command has a name and a set of supported arguments.

You can see a list of supported commands by simply executing the below command. For example:

```
$>{$CLI_HOME}/hdr-fhir-cli.sh

Usage:
  hdr-fhir-cli {command} [options]

Commands:
  export-conceptmap-to-csv  - Exports a specific ConceptMap resource to a CSV file.
```

```
  import-csv-to-conceptmap - Imports a CSV file to a ConceptMap resource.

  upload-definitions       - Uploads the conformance resources
(StructureDefinition and ValueSet) from the official FHIR
definitions.
  upload-examples          - Downloads the resource example pack from the
HL7.org FHIR specification website, and uploads all of the example resources to
a given server.

  upload-terminology       - Uploads a terminology package (e.g. a SNOMED CT ZIP
file) to a server, using the $upload-external-code-system operation.
```

You can also see the list of supported arguments for a given command by issuing command help [commandname]. For example:

```
$>{$CLI_HOME}/hdr-fhir-cli.sh help upload-terminology

Usage:
  hdr-fhir-cli upload-terminology [options]

Uploads a terminology package (e.g. a SNOMED CT ZIP file) to a server, using the
$upload-external-code-system operation.

Options:

  -d,--data <arg>              Local file to use to upload (can be a raw file
or   a ZIP containing the raw file)
  -l,--logging                 If specified, verbose logging will be used.
  -t,--target <target>         Base URL for the target server (e.g.
                               " http://localhost:7001/oracle-fhir-server/fhir").
  -u,--url <arg>               The code system URL associated with this upload
(e.g. http://snomed.info/sct)
  -v,--fhir-version <version>  The FHIR version being used. Valid values: r4
```

- export-conceptmap-to-csv
- import-csv-to-conceptmap
- upload-definitions and upload-examples
- upload-terminology

export-conceptmap-to-csv

The export-conceptmap-to-csv command can be used to export a ConceptMap resource as a CSV file of terminology mappings.

The first row of the CSV file will include the following headers:

```
SOURCE_CODE_SYSTEM - ConceptMap.group.source
SOURCE_CODE_SYSTEM_VERSION - ConceptMap.group.sourceVersion
TARGET_CODE_SYSTEM - ConceptMap.group.target
TARGET_CODE_SYSTEM_VERSION - ConceptMap.group.targetVersion
SOURCE_CODE - ConceptMap.group.element.code
SOURCE_DISPLAY - ConceptMap.group.element.display
TARGET_CODE - ConceptMap.group.element.target.code
TARGET_DISPLAY - ConceptMap.group.element.target.display
EQUIVALENCE - ConceptMap.group.element.target.equivalence
```

```
ConceptMapEquivalence)
    COMMENT - ConceptMap.group.element.target.comment
```

Usage:

```
hdr-fhir-cli export-conceptmap-to-csv [options]
```

Exports a specific ConceptMap resource to a CSV file.

Options:

-f,--filename <filename>	The path and filename of the CSV file to be exported (./output.csv).
-l,--logging	If specified, verbose logging will be used.
-t,--target <target>	Base URL for the target server (e.g. “http://localhost:7001/oracle-fhir-server/fhir").
u,--url <url>	The URL of the ConceptMap resource to be exported (i.e. ConceptMap.url).
-v,--fhir-version <version>	The FHIR version being used. Valid values: r4.

These terminology mappings could then be exported with the following command:

```
./hdr-fhir-cli.sh export-conceptmap-to-csv --fhir-version R4 -t http://localhost:8080//
oracle-fhir-server/fhir -u http://hl7.org/fhir/ConceptMap/cm-administrative-gender-v2 -
f /u01/output.csv
```

import-csv-to-conceptmap

The import-csv-to-conceptmap command can be used to import a CSV file of terminology mappings and store it as a ConceptMap resource.

The first row of the CSV file is expected to include the following headers:

```
SOURCE_CODE_SYSTEM - ConceptMap.group.source
SOURCE_CODE_SYSTEM_VERSION - ConceptMap.group.sourceVersion
TARGET_CODE_SYSTEM - ConceptMap.group.target
TARGET_CODE_SYSTEM_VERSION - ConceptMap.group.targetVersion
SOURCE_CODE - ConceptMap.group.element.code
SOURCE_DISPLAY - ConceptMap.group.element.display
TARGET_CODE - ConceptMap.group.element.target.code
TARGET_DISPLAY - ConceptMap.group.element.target.display
EQUIVALENCE - ConceptMap.group.element.target.equivalence (ConceptMapEquivalence)
COMMENT - ConceptMap.group.element.target.comment
```

An example CSV file that describes the mapping of FHIR to HL7v2 for Administrative Gender would appear as follows:

```
"SOURCE_CODE_SYSTEM","SOURCE_CODE_SYSTEM_VERSION","TARGET_CODE_SYSTEM","TARGET_CODE_SYS
TEM_VERSION","SOURCE_CODE","SOURCE_DISPLAY","TARGET_CODE","TARGET_DISPLAY","EQUIVALENCE
","COMMENT"
"http://hl7.org/fhir/administrative-gender","","http://hl7.org/
fhir/v2/0001","","male","Male","M","Male","equal",""
```

Usage:

```
hdr-fhir-cli import-csv-to-conceptmap [options]
```

Imports a CSV file to a ConceptMap resource.

Options:

-f,--filename <filename>	The path and filename of the CSV file to be imported (for example, ./input.csv).
-i,--input <input>	The source value set of the ConceptMap to be imported (i.e. ConceptMap.sourceUri).
-l,--logging	If specified, verbose logging will be used.
-o,--output <output>	The target value set of the ConceptMap to be imported (i.e. ConceptMap.targetUri).
-t,--target <target>	Base URL for the target server (e.g. "http://localhost:7001/oracle-fhir-server/fhir").
-u,--url <url>	The URL of the ConceptMap resource to be imported/exported (i.e. ConceptMap.url).
-v,--fhir-version <version>	The FHIR version being used. Valid values: r4.

These terminology mappings could then be imported with the following command:

```
./hdr-fhir-cli.sh import-csv-to-conceptmap --fhir-version R4 -t http://
localhost:8080//oracle-fhir-server/fhir -u http://hl7.org/fhir/ConceptMap/cm-
administrative-gender-v2 -i http://hl7.org/fhir/ValueSet/administrative-gender -
o http://hl7.org/fhir/ValueSet/v2-0001 -f  /u01/sampleInputFile.csv
```

upload-definitions and upload-examples

The upload-definitions command uploads the conformance resources (StructureDefinition and ValueSet) from the official FHIR definitions.

The upload-examples command uploads the example resources from the official FHIR definitions.

Usage:

```
hdr-fhir-cli upload-definitions/upload-examples [options]
```

The conformance rules are available at https://www.hl7.org/fhir/conformance-rules.html.

Options:

-e,--exclude <arg>	Exclude uploading the given resources, such as "-e dicom-dcim,foo".
-t,--target <arg>	Base URL for the target server (such as "http://localhost:7001/oracle-fhir-server/fhir ").
-v,--fhir-version <version>	The FHIR version being used. Valid values: r4.

Command usage:

```
./hdr-fhir-cli.sh upload-definitions --fhir-version R4 -t http://localhost:8181/
baseR4 "-e dicom-dcim,foo"

./hdr-fhir-cli.sh upload-examples --fhir-version R4 -t http://localhost:8181/
baseR4 "-e dicom-dcim,foo"
```

upload-terminology

The HDR FHIR server provides a terminology server, which supports ingestion of terminology data provided by LOINC and SNOMED. For more information on obtaining the above terminology data refer to the respective websites.

The HDR FHIR server provides a repository for terminology content used across the HDR platform, and an API suite to access the content.

The server provides a mechanism for ingestion of the terminology data via the upload-terminology command of the CLI tool. This command supports only LOINC and SNOMED terminologies in the current release.

> **Note:**
>
> The path and exact filename of the terminology files will likely need to be adjusted for your local disk structure.

Usage:

```
hdr-fhir-cli upload-terminology [options]
```

Uploads a terminology package (such as a SNOMED CT ZIP file) to a server, using the $upload-external-code-system operation.

Options:

-d,--data <arg>	Local file to use to upload (can be a raw file or a ZIP containing the raw file).
-l,--logging	If specified, verbose logging will be used.
-t,--target <target>	Base URL for the target server (such as "http://localhost:7001/oracle-fhir-server/fhir").
-u,--url <arg>	The code system URL associated with this upload (such as http://snomed.info/sct).
-v,--fhir-version <version>	The FHIR version being used. Valid values: r4.

Command usage:

```
$>{$CLI_HOME}$./hdr-fhir-cli.sh upload-terminology -d /scratch/fhir/Loinc_2.65.zip -d /
scratch/fhir/loincupload.properties --fhir-version R4 -t http://localhost:8080//oracle-
fhir-server/fhir -u http://loinc.org
```

8
Working with FHIR REST APIs

Oracle Healthcare Data Repository 8.1.4-FHIR offers a suite of REST APIs implemented as per the HL7 FHIR specification and is secured using the OAuth 2.0 security framework. For more information, refer to the HDR FHIR REST API documentation.

https://support.oracle.com/epmos/faces/DocumentDisplay?id=2913016.1

9
HDR FHIR Data Model

For information on the Oracle Healthcare Data Repository 8.1.4-FHIR data model, see the FHIR Technical Reference Manual, available from Oracle Support:

https://support.oracle.com/epmos/faces/DocumentDisplay?id=2913016.1

10

Data Store in Repository

This section describes about how the FHIR resource data gets stored in the HDR FHIR repository.

The HDR FHIR JPA schema relies on the concept of internal persistent IDs on tables, using a Java type of Long.

Many tables use an internal persistent ID as their primary key, allowing flexibility for other more complex business identifiers to be changed and minimizing the amount of data consumed by foreign key relationships. Persistent ID columns are generally assigned using the database sequences. The persistent ID column is generally called PID in the HDR FHIR schema.

- Resources
  Various resources are used in storing data in the HDR FHIR repository.
- Search Indexes
  Search Parameter Index are used to index resources for searching.

Resources

Various resources are used in storing data in the HDR FHIR repository.

Resource Master Table

The table called HFJ_RESOURCE indicates a single resource of any type in the database. For example, the resource Patient/1 will have exactly one row in this table, representing all versions of the resource.

Resource Versions and Contents

The table called HFJ_RES_VER contains individual versions of a resource. If the resource Patient/1 has 3 versions, there will be 3 rows in this table.

The complete raw contents of the resource is stored in either the RES_TEXT or the RES_TEXT_VC column of the resource specific extended table OHF_FHIR_xxx, using the encoding specified in the RES_ENCODING column of the HFJ_RES_VER.

Example: Resource Extended Table OHF_FHIR_PATIENT

The following property decides whether to store the resource as plain text in RES_TEXT_VC column or not.

If the resource size is below the size as set in the following property, it goes to RES_TEXT_VC column and RES_ENCODING columns value is JSON.

If the resource size is larger than the property set value, it goes to RES_TEXT as a compressed blob.

```
inline_resource_storage_below_size: 0
```

Encoding

The resource is serialized using FHIR JSON encoding, and then compressed into a byte stream using GZIP compression. This will be controlled using the following property:

```
resource_compression_enabled: true
```

If the above property is set to **true**, the resource is serialized and then compressed into byte stream using GZIP compression. Then the compressed resource is stored into RES_TEXT column as a blob value with RES_ENCODING set to JSONC.

If the above property is set to **false**, the resource is not compressed and the plain text resource is stored into RES_TEXT column as a blob with RES_ENCODING set to JSON.

Client Assigned Resource IDs

By default, the HFJ_RESOURCE.RES_ID column is used as the resource ID for all server-assigned IDs. For example, if a Patient resource is created in a completely empty database, it will be assigned the ID Patient/1 by the server and RES_ID will have a value of 1.

However, when client-assigned IDs are used, these may contain text values to allow a client to create an ID such as Patient/ABC. When a client-assigned ID is given to a resource, a row is created in the HFJ_RESOURCE table. When an HFJ_FORCED_ID row exists corresponding to the equivalent HFJ_RESOURCE row, the RES_ID value is no longer visible or usable by FHIR clients and it becomes purely an internal ID to the JPA server.

If the server has been configured with a Resource Client ID Strategy of ANY, the server use the Resource Server ID Strategy of UUID and will create a Forced ID for all resources (not only resources having textual IDs).

```
Propety:
client_id_strategy: ANY
```

Resource Links

Resource links will be established between the two resources. When a resource is created or updated, it is indexed for searching. Any search parameters of type Reference are resolved, and one or more rows may be created in the HFJ_RES_LINK table between source Resource Id and Target Resource ID.

Search Indexes

Search Parameter Index are used to index resources for searching.

The table starts with **HFJ_SPIDX** (Search Parameter Index) used to index resources for searching. When a resource is created or updated, a set of rows will be added in the tables. These are used for finding appropriate rows to return when performing FHIR searches. There are dedicated tables for supporting each of the non-reference **FHIR Search Datatypes**: Date, Number, Quantity, String, Token, and URI:

- `HFJ_SPIDX_DATE`
- `HFJ_SPIDX_NUMBER`

- HFJ_SPIDX_QUANTITY
- HFJ_SPIDX_QUANTITY_NRML
- HFJ_SPIDX_STRING
- HFJ_SPIDX_TOKEN
- HFJ_SPIDX_URI

> **Note:**
>
> Reference search parameters are implemented using the HFJ_RES_LINK table above.

The SPIDX tables leverage "hash columns", which contain a hash of multiple columns in order to reduce index size and improve search performance. Hashes currently use the MurmurHash3_x64_128 hash algorithm, keeping only the first 64 bits in order to produce a LongInt value. For example, all search index tables have columns for storing the search parameter name (SP_NAME) and resource type (RES_TYPE). An additional column which hashes these two values is provided:

HASH_IDENTITY

11
FHIR Operations

This section gives an overview of some of the features of HDR FHIR, and it shows you how to manage and configure it for a few basic functionalities.

- FHIR CRUD (Create/Read/Update/Delete) operations
  This section describes how to use the HDR FHIR Endpoint to perform basic CRUD (Create/Read/Update/Delete) operations.
- FHIR Search operations
  This section describes the FHIR Search operations.
- FHIR Bundle transactions and batches
  Oracle FHIR includes a mechanism that can be used by a client to send multiple interactions to a server for processing. This mechanism uses the FHIR bundle resource as the transport, with a collection of one or more interactions grouped inside the bundle.
- Search parameters
  Search parameters are named paths within resources that are indexed by the system so that they can be used to find resources that match a given criteria.
- Search parameter features
  This section describes optional features that you can enable or disable for optimizing how searching works on Oracle HDR-FHIR server.
- Searching for data
  This section contains information about methods for searching for data in Oracle HDR FHIR.
- Creating data
  This section describes creating data in Oracle HDR FHIR repository.
- Reading data
  This section describes methods for reading data from the Oracle HDR FHIR Repository.
- Updating data
  This section describes information about methods for updating data in the HDR FHIR Repository.
- Deleting data
  The FHIR delete operation performs a "logical" delete. This means that data is not physically removed from the database.
- Binary Access Operations
  In many cases, resources such as **DocumentReference** are used to store large files such as scanned PDFs and images. These resources use the Attachment datatype, which ultimately stores a content type and a base 64 encoded representation of the binary content.

FHIR CRUD (Create/Read/Update/Delete) operations

This section describes how to use the HDR FHIR Endpoint to perform basic CRUD (Create/Read/Update/Delete) operations.

- Create
  A client may create a new resource on an HDR FHIR server by performing a Create operation. The Create uses an HTTP POST against the URL [baseUrl]/[resourceName]. This POST should have a Content-Type header which specifies the MIME type of the payload.
- Update
  You can update an existing resource on an FHIR server by performing an Update operation. The Update uses an HTTP PUT against the URL [baseUrl]/[resourceName]/[id]. This PUT should have a Content-Type header that specifies the MIME type of the payload.
- Delete
  You can delete a resource on an FHIR server by performing a Delete operation. The Delete uses an HTTP DELETE against the URL [baseUrl]/[resourceType]/[id].
- Patch
  You can patch a resource on an FHIR server by performing a Patch operation. This operation support allows you to modify a resource in place by supplying a delta. The Patch uses an HTTP PATCH against the URL [baseUrl]/[resourceType]/[id]. This operation requires a Content-Type header that specifies the MIME type of the payload.
- Read
  If you know the ID of a resource on an FHIR server, you can read back the most recent version of that resource by performing a Read operation. The Read uses an HTTP GET against the URL [baseUrl]/[resourceType]/[id].
- vRead
  You can also include a version string in the URL in order to request a specific version of the resource by performing a vRead operation. The vRead uses an HTTP GET against the URL [baseUrl]/[resourceType]/[id]/_history/[versionId].

Create

A client may create a new resource on an HDR FHIR server by performing a Create operation. The Create uses an HTTP POST against the URL [baseUrl]/[resourceName]. This POST should have a Content-Type header which specifies the MIME type of the payload.

The following example shows a simple Patient resource create using a JSON Payload.

Request: FHIR CRUD Create operation

HTTP Method	**POST**
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient
Headers	**Content-Type: application/fhir+json**

Request Body	{
	"resourceType": "Patient",
	"identifier": [{ "system": "
	urn:oid:1.2.36.146.595.217.0.1", "value":
	"12345" }],
	"name": [{
	"family": "Chalmers",
	"given": ["Peter", "James"]
	}],
	"gender": "male",
	"birthDate": "1974-12-25"
	}

Response: FHIR CRUD Create operation

The endpoint responds with a response similar to the following:

```
Status Code: 201 Created
 etag: W/"1"
 location: http://localhost:8001/oracle-fhir-server/fhir/Patient/199963/_history/1
 x-powered-by: Oracle FHIR REST Server (FHIR Server; FHIR 4.0.1/R4)
 x-request-id: ENqsVXqc5SeqLOfU

{
  "resourceType": "Patient",
           "id": "199963",
           "meta": {
            "versionId": "1",
            "lastUpdated": "2022-10-10T07:07:33.086-04:00"
           },
           "text": {
            "status": "generated",
            "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><div
          class=\"hapiHeaderText\">Peter James <b>CHALMERS </b></div><table
          class=\"hapiPropertyTable\"><tbody><tr><td>Identifier</td><td>12345</td></tr><
          tr><td>Date of birth</td><td><span>25 December
          1974</span></td></tr></tbody></table></div>"
           },
           "identifier": [ {
            "system": " urn:oid:1.2.36.146.595.217.0.1",
            "value": "12345"
           } ],
           "name": [ {
            "family": "Chalmers",
            "given": [ "Peter", "James" ]
           } ],
           "gender": "male",
           "birthDate": "1974-12-25"
}
```

Update

You can update an existing resource on an FHIR server by performing an Update operation. The Update uses an HTTP PUT against the URL [baseUrl]/[resourceName]/[id]. This PUT should have a Content-Type header that specifies the MIME type of the payload.

The following example shows a simple Patient resource update using a JSON Payload. This example uses the previously created Patient resource (see Create) and updates it by adding an address.

Request: FHIR CRUD Update operation

HTTP Method	**PUT**
URL	`http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/199963`
Headers	**Content-Type: application/fhir+json**

Request Body	{
	"resourceType": "Patient",
	"id": "**199963**",
	"identifier": [{ "system": "urn:oid:1.2.36.146.595.217.0.1", "value": "12345" }],
	"name": [{
	"family": "Chalmers",
	"given": ["Peter", "James"]
	}],
	"gender": "male",
	"birthDate": "1974-12-25",
	"address": [{
	"line": ["534 Erewhon St"],
	"city": "PleasantVille",
	"state": "Vic",
	"postalCode": "3999"
	}]
	}

Response: FHIR CRUD Update operation

The endpoint responds with a response similar to the following:

```
Status Code: 200 Created
 etag: W/"2"
 content-location: http://localhost:8001/oracle-fhir-server/fhir/Patient/199963/_history/2

{
 "resourceType": "Patient",
 "id": "199963",
 "meta": {
  "versionId": "2",
  "lastUpdated": "2022-10-10T07:51:04.607-04:00"
 },
 "text": {
  "status": "generated",
  "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><div
class=\"hapiHeaderText\">Peter James <b>CHALMERS </b></div><table
class=\"hapiPropertyTable\"><tbody><tr><td>Identifier</td><td>12345</td></tr><tr><td>
Address</td><td><span>534 Erewhon St </span><br/><span>PleasantVille
</span><span>Vic </span></td></tr><tr><td>Date of birth</td><td><span>25 December
1974</span></td></tr></tbody></table></div>"
 },
 "identifier": [ {
  "system": "urn:oid:1.2.36.146.595.217.0.1",
  "value": "12345"
 } ],
 "name": [ {
  "family": "Chalmers",
  "given": [ "Peter", "James" ]
 } ],
 "gender": "male",
 "birthDate": "1974-12-25",
 "address": [ {
  "line": [ "534 Erewhon St" ],
  "city": "PleasantVille",
  "state": "Vic",
  "postalCode": "3999"
 } ]
}
```

Delete

You can delete a resource on an FHIR server by performing a Delete operation. The Delete uses an HTTP DELETE against the URL [baseUrl]/[resourceType]/[id].

This operation performs a logical delete, which has a specific set of semantics:

- The resource is marked as deleted, and it no longer appears in search results.
- The version number of the resource is incremented (that is, a new deleted version is created).
- Previous versions of the resource are not physically deleted.
- The resource may be un-deleted by updating it again.

The following example shows a simple delete of the resource created and updated in the examples in Create and Update.

Request: FHIR CRUD Delete operation

HTTP Method	DELETE
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/199963

Response: FHIR CRUD Delete operation

The endpoint responds with a response similar to the following:

```
HTTP/1.1 200 OK
ETag: W/"3"
 Location: Location: http://localhost:8001/oracle-fhir-server/fhir/Patient/199963/_history/3

{
 "resourceType": "OperationOutcome",
 "issue": [
  {
   "severity": "information",
   "code": "informational",
   "diagnostics": "Successfully deleted 1 resource(s) in 29ms"
  }
 ]
}
```

Patch

You can patch a resource on an FHIR server by performing a Patch operation. This operation support allows you to modify a resource in place by supplying a delta. The Patch uses an HTTP PATCH against the URL [baseUrl]/[resourceType]/[id]. This operation requires a Content-Type header that specifies the MIME type of the payload.

For example, for a JSON patch payload the header value is: application/json-patch+json.

Updating a resource using patch operation: PATCH works by creating a patch document that specifies the changes that the client wishes to make to a resource, and submits that to the server. The server then applies those changes to the indicated resource and optionally returns the updated resource.

Each change in the patch document (and there can be multiple) specifies:

- The operation that is to be performed (for example, add, remove, or replace).
- The location in the resource to apply the operation.
- The new value (if an add or change).

The following example shows how to perform a patch using a FHIR Patch.

Here is the initial patient resource.

```
{

"resourceType": "Patient",

"id" : "149954",

"birthDate": "1974-02-13"

}
```

Example 1: Add Operation

- **Adding the gender.** If a target element is an object (for example, not an array) and already exists then the value is replaced. Otherwise, it is added.

Table 11-1 Request

HTTP Method	PATCH
URL	`http://<HOST>:<PORT>/oracle-fhir-server/fhir/Patient/149954`
Headers	`Content-Type: application/json-patch+json`
Request Body	`[{ "op": "add", "path": "/gender", "value": "male" }]`

> **Note:**
>
> The Patch document is in an array as there can be any number of operations in a single PATCH call.

Response: The FHIR endpoint will respond with a response similar to the following:

```
{
"resourceType": "Patient",
"id": "149954",
"meta": {
"versionId": "2",
"lastUpdated": "2022-04-24T19:05:13.111+05:30",
"source": "#sieqifF8d5n5c2dU"
},
```

```
"text": {
"status": "generated",
"div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><table
class=\"hapiPropertyTable\"><tbody/></table></div>"
},
"gender": "female"
}
```

- **Adding an address.** This element can repeat. It is represented in the resource instance as an array. It is possible to adjust an array, but unlike an object, the array will not be automatically created if it does not exist. So, the contents of the patch document will be different if there is no address already. It needs to be created first.

Table 11-2 Request

HTTP Method	PATCH
URL	`http://<HOST>:<PORT>/oracle-fhir-server/fhir/Patient/149954`
Headers	`Content-Type: application/json-patch+json`
Request Body	`[ { "op": "add", "path": "/address", "value":[]}, { "op": "add", "path": "/address/0", "value": { "use" : "home", "line" : ["<Sample Street Name>","avon"], "city" : "<City_Name>", "district": "<Sample Street Name>", "state": "Vic", "postalCode": "3999", "text":"<Sample Street Name>" } } ]`

Response: The FHIR endpoint will respond with a response similar to the following:

```
{
"resourceType": "Patient",
"id": "149954",
"meta": {
"versionId": "3",
"lastUpdated": "2022-04-24T19:17:02.229+05:30",
```

```
"source": "#2dK6qlDOD2W4vQbh"
},
"text": {
"status": "generated",
"div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><table
class=\"hapiPropertyTable\"><tbody/></table></div>"
},
"gender": "female",
"address": [ {
"use": "home",
"text": "<Sample Street Name>",
"line": [ "<Sample Street Name>", "avon" ],
"city": "<City_Name>",
"district": "<Sample Street Name>",
"state": "Vic",
"postalCode": "3999"
} ]
}
```


Example 2: Replace Operation

Update address content and birthDate object with replace operation.

Table 11-3 Request

HTTP Method	PATCH
URL	`http://<HOST>:<PORT>/oracle-fhir-server/fhir/Patient/149954`
Headers	`Content-Type: application/json-patch+json`
Request Body	`[ { "op": "replace", "path": "/address/0/postalCode", "value": "4000" }, { "op": "replace", "path": "/birthDate", "value": "1974-02-20" } ]`

Response: The FHIR endpoint will respond with a response similar to the following:

```
{
"resourceType": "Patient",
"id": "149954",
"meta": {
"versionId": "4",
"lastUpdated": "2022-04-24T19:37:51.559+05:30",
"source": "#fZYCuXUNmAoZYbDH"
},
"text": {
"status": "generated",
"div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><table
class=\"hapiPropertyTable\"><tbody/></table></div>"
},
"gender": "female",
"birthDate": "1974-02-20",
"address": [ {
"use": "home",
"text": "<Sample Street Name>t",
"line": [ "<Sample Street Name>", "avon" ],
"city": "<City_Name>",
"district": "<Sample Street Name>",
"state": "Vic",
"postalCode": "4000"
} ]
}
```

Example 3: Remove Operation

```
[
{"op": "remove", "path": "/address/0"}
]
```

For adjusting the array, you need to know the position of the element in the array to remove. If you want to be sure, then you can require that the client submit the actual address that they are removing so the server can check. This can be done using the test operation. The document below will remove the address only if its value matches.

```
[
{"op": "test", "path": "/address/0","value": { "use" : "home", "line" :
["<Sample Street Name>","avon"], "city" : "<City_Name>", "district":
"<Sample Street Name>", "state": "Vic", "postalCode": "4000",
"text":"<Sample Street Name>"}},
{"op": "remove", "path": "/address/0"}
]
```

Table 11-4 Request

HTTP Method	PATCH
URL	`http://<HOST>:<PORT>/oracle-fhir-server/fhir/Patient/149954`
Headers	`Content-Type: application/json-patch+json`
Request Body	`[ {"op": "test", "path": "/address/0","value": { "use" : "home", "line" : ["<Sample Street Name>","avon"], "city" : "<City_Name>", "district": "<Sample Street Name>", "state": "Vic", "postalCode": "4000", "text":"<Sample Street Name>"}}, {"op": "remove", "path": "/address/0"} ]`

Response: The FHIR endpoint will respond with a response similar to the following:

```
{
"resourceType": "Patient",
"id": "149954",
"meta": {
"versionId": "5",
"lastUpdated": "2022-04-24T19:54:03.769+05:30",
"source": "#Bu5gN7pTIzZgLrM9"
},
"text": {
"status": "generated",
"div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><table
class=\"hapiPropertyTable\"><tbody/></table></div>"
},
"gender": "female",
"birthDate": "1974-02-20"
}
```

Read

If you know the ID of a resource on an FHIR server, you can read back the most recent version of that resource by performing a Read operation. The Read uses an HTTP GET against the URL [baseUrl]/[resourceType]/[id].

The following example shows a simple read of the resource created and updated in the examples in Create, Update, Delete, and Patch.

Request: FHIR CRUD Read operation

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/199963

Response: FHIR CRUD Read operation

The FHIR endpoint responds with a response similar to the following:

```
HTTP/1.1 200 OK ETag: W/"3"
Location: http://localhost:8001/oracle-fhir-server/fhir/Patient/199963/_history/3

{
  "resourceType": "Patient",
  "id": "199963",
  "meta": {
    "versionId": "3",
    "lastUpdated": "2019-07-12T01:58:07.164+00:00"
  },
  "identifier": [
    {
      "system": "urn:oid:1.2.36.146.595.217.0.1",
      "value": "12345"
    }
  ],
  "name": [
    {
      "family": "Chalmers",
      "given": [
        "Peter",
        "James"
      ]
    }
  ],
  "gender": "male",
  "birthDate": "1974-02-13",
  "address": [
    {
      "line": [
        "534 Erewhon St"
      ],
      "city": "PleasantVille",
      "state": "Vic",
      "postalCode": "M5C 2X8"
    }
  ]
}
```

vRead

You can also include a version string in the URL in order to request a specific version of the resource by performing a vRead operation. The vRead uses an HTTP GET against the URL [baseUrl]/[resourceType]/[id]/_history/[versionId].

The following example shows a simple vRead of the original resource created in Create.

Request: FHIR vRead operation

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/199963/ history/3

Response: FHIR vRead operation

The FHIR endpoint responds with a response similar to the following:

```
HTTP/1.1 200 OK ETag: W/"3" Location: http://localhost:8001/oracle-fhir-server/fhir/Patient/199963/_history/3

{
  "resourceType": "Patient",
  "id": "199963",
  "meta": {
    "versionId": "3",
    "lastUpdated": "2019-07-12T01:58:07.164+00:00"
  },
  "identifier": [
    {
      "system": "urn:oid:1.2.36.146.595.217.0.1",
      "value": "12345"
    }
  ],
  "name": [
    {
      "family": "Chalmers",
      "given": [
        "Peter",
        "James"
      ]
    }
  ],
  "gender": "male",
  "birthDate": "1974-02-13",
  "address": [
    {
      "line": [
        "534 Erewhon St"
      ],
      "city": "PleasantVille",
      "state": "Vic",
      "postalCode": "M5C 2X8"
    }
  ]
}
```

FHIR Search operations

This section describes the FHIR Search operations.

- Basic searching: Finding patients
  The most basic form of a search is a search with no parameters, which matches all resources of a given type. You can also search with a name parameter or multiple parameters.
- References: Finding encounters
  Patients often have one or more encounters. You can search for all encounter resources for a specific patient using the subject search parameter.
- Quantities: Finding laboratory values
  There are a few options to search Observation resources for lab tests. You can search for all lab tests or specific lab tests for all patients or a specific patient.
- Dates and times: Narrowing your search
  Searching by date in FHIR is quite powerful. If you want to search for anything matching a date, you can use that date as the search parameter value.
- Paging search results
  When returning search results, the server pages them by default.
- Sorting search results
  FHIR makes it easy to sort the results of your search query according to whatever criteria you require. Add the _sort parameter followed by the name of a search parameter.
- Full text searching
  You can perform a full text search across the textual contents of resources. Using the _content parameter, the various textual fields in resources are matched.
- Patient search $everything
  Oracle FHIR provides a special query known as the $everything operation, which returns all resources associated with a specific patient. The query searches the patient's entire chart.

Basic searching: Finding patients

The most basic form of a search is a search with no parameters, which matches all resources of a given type. You can also search with a name parameter or multiple parameters.

Request: Search with no parameters

The following search returns all patients in the system.

HTTP Method	**POST**
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient
Headers	**Content-Type: application/fhir+json**
Request Body	{ "resourceType": "Patient", "identifier": [{ "system": " urn:oid:1.2.36.146.595.217.0.1", "value": "12345" }], "name": [{ "family": "Chalmers", "given": ["Peter", "James"] }], "gender": "male", "birthDate": "1974-12-25" }

Response: Search with no parameters

The response to the query only returns a small amount of patient records (the default is 20). You can however, narrow your search using a name parameter or multiple parameters.

Request: Search with a name parameter

The following search query finds any patients with the family name "Chalmers."

HTTP Method	**GET**
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?name=chalmers

Request: Search with multiple parameters (as an AND combination)

You can combine multiple parameters to narrow down your search even further (multiple parameters are interpreted as an AND combination).

HTTP Method	**GET**
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?name=chalmers&gender=male

Request: Search with multiple parameters as comma separated values (as an OR search)

Comma separated values are treated as an OR search (either value may match). For example, the following search query finds people with the family name "Chalmers" and a given name of either "James" or "Peter."

HTTP Method	**GET**
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?family=chalmers&given=peter,james

References: Finding encounters

Patients often have one or more encounters. You can search for all encounter resources for a specific patient using the subject search parameter.

An encounter has several details contained within the resource. Most importantly, it has a subject, which is a reference to the patient for whom this encounter exists. It also generally has references to practitioner resources who are a part of the encounter in some way.

Request: Search query for encounters

The following example uses the **_sort parameter** so the results are returned in order by date from oldest to newest.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Encounter?subject=Patient/k62d9d82-3f2b-44db-b808-04159be709fd&_sort=date

Quantities: Finding laboratory values

There are a few options to search Observation resources for lab tests. You can search for all lab tests or specific lab tests for all patients or a specific patient.

Request: Search query for all lab tests for a patient

The following search query uses the laboratory category to find all lab tests for a given patient.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation?subject=Patient/h0e270d9-ed56-4041-9dec-a45a73461c66&category=http://hl7.org/fhir/observation-category|laboratory

Request: Search for a specific lab test for all patients

To find a specific lab test, you can use the LOINC code for that test and the code search parameter. For example, the following search query finds all Potassium tests (LOINC code 6298-4) across all patients.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation?code=http://loinc.org|6298-4

Request: Search for a value and quantity

You can also add a value-quantity search parameter to find, for example, potassium tests with a specific value. The following example searches for any potassium tests with a value below 4.0 mmol/L.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation?code=http://loinc.org|6298-4 &value-quantity=lt4.0|http://unitsofmeasure.org/|mmol/L

Dates and times: Narrowing your search

Searching by date in FHIR is quite powerful. If you want to search for anything matching a date, you can use that date as the search parameter value.

Request: Search query for all patients with a specific birthday

The following example shows a search query for all patients with a given birthday.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?birthdate=1974-02-13

Request: Search query for all patients born in a specific year

Partial dates are used to narrow the search, and they will match any values within the part. For example, the following search finds all patients born in 1974.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?birthdate=1974

Request: Search query for a date range

Date searches also support a set of qualifiers, which are prefixes on the date that act as a comparator. For example, the value ge2011-01-01 matches any date on or after 2011-01-01. A number of other qualifiers are available under the Date section of the FHIR Search page. The following search query request matches any Encounters for the given patient that started or ended between June 2009 and July 2009.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Encounter?subject=Patient/ef2c19c4-ea06-473d-8781-368e4441c5c0&date=ge2009-06-01&date=le2009-07-31

Request: Search query for last updated

FHIR also supports a special parameter called _lastUpdated that can be used on any resource type. This parameter filters search results to only match resources that were last updated at or after the given date/time (or optionally before). The _lastupdate parameter considers the value found in Resource.meta.lastUpdated. It is also important to remember that this value is always provided by the server, and you do not have any control over the resource last updated date. The following search request finds Patient resources updated on or after midnight on Jan 1, 2017.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?_lastUpdated=ge2017-01-01T00:00:00Z

Paging search results

When returning search results, the server pages them by default.

Request: Search query for paging search results

Consider the following search query.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?birthdate=1974-02-13

Response: Search query for paging search results

The following example show that the response returns a Bundle. Let's say, there are 200 matching search results in total. However, the Bundle actually contains only the first page of results. The URL marked **next** provides a link to fetch the next page of results. This allows client applications to fetch data in manageable amounts, which improves performance in many cases.

```
{
 "resourceType": "Bundle",
 "id": "3c4b5bfc-a03e-474c-9cb2-bd640022cf20",
 "meta": {
  "lastUpdated": "2022-10-13T14:07:20.613+00:00"
 },
 "type": "searchset",
 "link": [
  {
   "relation": "self",
   "url": "http://localhost:8001/oracle-fhir-
server/fhir/Observation?code=http%3A%2F%2Floinc.org%7C718-7"
  },
  {
   "relation": "next",
   "url": "http://localhost:8001/oracle-fhir-server/fhi?_getpages=3c4b5bfc-a03e-474c-
9cb2-
bd640022cf20&_getpagesoffset=20&_count=20&_pretty=true&_bundletype=searchset"
  }
 ],
 (...more...)
}
```


Request: Search query to control the number of returned results

You can control the number of returned results by using the _**count parameter**.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation?code=http://loinc.org|718-7&_count=20

Sorting search results

FHIR makes it easy to sort the results of your search query according to whatever criteria you require. Add the _sort parameter followed by the name of a search parameter.

Request: Search query for sorting single parameter search results

For example, you can search for patients with a single search parameter, such as sorting by family name (ascending and descending).

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?_sort=family

Request: Search query for sorting multiple parameter search results

You can also sort by multiple search parameters by combining them with a comma.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?_sort=family, given

Full text searching

You can perform a full text search across the textual contents of resources. Using the _content parameter, the various textual fields in resources are matched.

Request: Search query for full text searching

The following search query returns all Condition resources that contain the string "diabetes." The search is not case-sensitive.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Condition?_content=diabetes

> **Note:**
>
> FHIR also provides a second full text parameter, **_text** that you can use to search across the resource.

Patient search $everything

Oracle FHIR provides a special query known as the $everything operation, which returns all resources associated with a specific patient. The query searches the patient's entire chart.

Request: Search query for patient using search $everything

To invoke this operation on a specific patient, call $everything on a patient resource instance. For example:

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/1228032/$everything

Request: Search query for patient using search $everything by count

The result of this operation can be paged in order to avoid overwhelming the client and server. You can use the _count parameter and increase the page size from the resulting bundle.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/1228032/$everything?_count=50

FHIR Bundle transactions and batches

Oracle FHIR includes a mechanism that can be used by a client to send multiple interactions to a server for processing. This mechanism uses the FHIR bundle resource as the transport, with a collection of one or more interactions grouped inside the bundle.

FHIR bundles are sent to the FHIR server using an HTTP POST to the base URL of the server. There are two modes of processing, as determined by the **Bundle.type** value.

- **Transaction:** All operations in the bundle are executed as a single atomic database transaction. If any failures occur, the entire transaction is rolled back.
- **Batch:** Each operation in the bundle is executed as an independent database transaction. Any processing failures may cause the specific interaction to be rolled back but will not affect other operations.

Mostly, transaction bundles are commonly used to post data to a server, since any HTTP REST operation is a process to be included in a transaction.

- Basic bundle transaction
  Oracle FHIR transaction bundle has a few important elements.
- Bundle multiple related resources
  A common use for FHIR bundle transactions is to persist a collection of related resources to a server. For example, if you have a collection of Observation resources with the same patient as subject, you could place them inside a single FHIR bundle transaction and send them together to a server.
- Placeholder IDs and references
  Creating references between resources is easy if you know the ID of the reference target (such as in the example above with client-assigned IDs) but it is also possible to have references between resources in a single bundle even if you don't yet know the target resource ID.
- Conditional Create
  The FHIR Conditional Create mechanism allows the client to specify a FHIR search URL alongside a resource to create. When processing the create, the server will first check if any resource already exists matching the given search. If no resources match, the create proceeds. If a resource does match, no resource is created.
- Conditional Update
  The FHIR Conditional Update mechanism allows a resource to be transmitted to the server for updating, but instead of supplying an ID to update, the client supplies a search URL similar to the URL used for the Conditional Create example.
- Delete
  FHIR logical deletes can be performed as a part of a transaction, as well. In a transaction bundle, deletes will be applied as a group, meaning that if one fails, all will be rolled back.
- Patch
  The FHIR Patch operation can also be performed in a transaction bundle. When using the FHIR Patch mechanism for patching, the FHIR Patch document is placed in Bundle.entry.resource. In the case of JSON Patch the contents are placed in a Binary

resource and then placed into Bundle.entry.resource. In all cases, the HTTP verb/method is PATCH.

Basic bundle transaction

Oracle FHIR transaction bundle has a few important elements.

- **Bundle.type:** This value specifies the processing mode (transaction or batch).
- **Bundle.entry:** This is an array. Each repetition of the this array contains a single interaction, and is the equivalent to a single HTTP REST interaction.
- **Bundle.entry.request:** The standard HTTP REST parameters (i.e. the verb, the request URL, and the request headers) are communicated in the element Bundle.entry.request.
- **Bundle.entry.resource:** Interactions which accept a resource as the payload body (such as create and update) place the payload in the element Bundle.entry.resource.
- **Bundle.entry.fullUrl:** An additional element in each entry has a full URL or URI associated with the entry, in the element Bundle.entry.fullUrl.

Sample FHIR bundle transaction. In this example, the fullUrl is a randomly generated UUID.

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [
    {
      "fullUrl": "urn:uuid:850bc2ca-d9ab-467b-9924-0e08d0a6e586",
      "resource": {
        "resourceType": "Patient",
        "identifier": [
          {
            "system": "https://github.com/synthetichealth/synthea",
            "value": "8ffd03ee-fb56-441d-a3ef-01cdc9f94d89"
          } ],
        "name": [
          {
            "use": "official",
            "family": "Trantow673",
            "given": [
              "Matthew562"
            ],
            "prefix": [
              "Mr."
            ]
          }
        ],
      "request": {
        "method": "POST",
        "url": "Patient"
      }
    } }
 }
```

Bundle multiple related resources

A common use for FHIR bundle transactions is to persist a collection of related resources to a server. For example, if you have a collection of Observation resources with the same patient as subject, you could place them inside a single FHIR bundle transaction and send them together to a server.

The simplest way to send related resources in a single bundle is to use client-assigned IDs. When using this form, the server is instructed to use the IDs supplied in the requests. Any references between resources simply use these client-assigned IDs. Resources may also have references to other resources which already exist on the server.

Sample Transaction bundle with client-assigned ID on three resources: a patient, and two observations for this patient.

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [ {
  "fullUrl": "Patient/PTA",
    "resource": {
      "resourceType": "Patient",
      "id": "PTA",
      "identifier": [ {
        "system": "http://acme.org/mrns",
        "value": "013872"
      } ],
      "name": [ {
        "family": "Simpson",
        "given": [ "Homer" ]
      } ]
    },
    "request": {
      "method": "PUT",
      "url": "Patient/PTA"
    }
      }, {
    "fullUrl": "Observation/OB1",
    "resource": {
      "resourceType": "Observation",
      "id": "OB1",
      "status": "final",
      "code": {
        "coding": [ {
          "system": "http://loinc",
          "code": "29463-7",
          "display": "Body Weight"
        } ]
      },
      "subject": {
        "reference": "Patient/PTA"
      },
      "effectiveDateTime": "2022-02-23",
      "valueQuantity": {
```

```
        "value": 67.1,
        "unit": "kg",
        "system": "http://unitsofmeasure.org",
        "code": "kg"
      }
    },
    "request": {
      "method": "PUT",
      "url": "Observation/OB1"
    }
      }, {
    "fullUrl": "Observation/OB2",
    "resource": {
      "resourceType": "Observation",
      "id": "OB2",
      "status": "final",
      "code": {
        "coding": [ {
          "system": "http://loinc",
          "code": "29463-7",
          "display": "Body Weight"
        } ]
      },
      "subject": {
        "reference": "Patient/PTA"
      },
      "effectiveDateTime": "2019-12-29",
      "valueQuantity": {
        "value": 72.4,
        "unit": "kg",
        "system": "http://unitsofmeasure.org",
        "code": "kg"
      }
    },
    "request": {
      "method": "PUT",
      "url": "Observation/OB2"
    }
      } ]
}
```

Placeholder IDs and references

Creating references between resources is easy if you know the ID of the reference target (such as in the example above with client-assigned IDs) but it is also possible to have references between resources in a single bundle even if you don't yet know the target resource ID.

For example, if an entry in your bundle is using a normal FHIR create (for example, HTTP POST) then it will assign an ID to the resource and your client will not know this ID until after the transaction has been processed.

FHIR solves this issue by using a feature called Placeholder IDs, which are UUIDs generated by the client to associate with each resource. These UUIDs are temporary

and should be randomly generated. They serve only to link resources in the Bundle together, and are thrown away by the server once it has determined the actual resource IDs.

The following example shows a transaction that creates two resources: a Patient and an Observation referring to this patient.

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [ {
  "fullUrl": "urn:uuid:e16eac01-a5ee-4904-b1c8-f4bd56e338d5",   >
Placeholder ID for Patient resource
        "resource": {
      "resourceType": "Patient",
      "identifier": [ {
        "system": "http://acme.org/mrns",
        "value": "013872"
      } ],
      "name": [ {
        "family": "Simpson",
        "given": [ "Homer" ]
      } ]
    },
    "request": {
      "method": "POST",
      "url": "Patient"
    }
  }, {
    "fullUrl": "urn:uuid:499733fe-7ced-4d15-81ce-8a433a1fb71e",
    "resource": {
      "resourceType": "Observation",
      "status": "final",
      "code": {
        "coding": [ {
          "system": "http://loinc",
          "code": "29463-7",
          "display": "Body Weight"
        } ]
      },
      "subject": {
        "reference": "urn:uuid:e16eac01-a5ee-4904-b1c8-f4bd56e338d5"   >
Reference to Patient placeholder ID. This will be automatically replaced
with the real resource ID by the server.
          },
      "effectiveDateTime": "2022-02-23",
      "valueQuantity": {
        "value": 67.1,
        "unit": "kg",
        "system": "http://unitsofmeasure.org",
        "code": "kg"
      }
    },
    "request": {
      "method": "POST",
      "url": "Observation"
```

```
    }
  } ]
}
```

Conditional Create

The FHIR Conditional Create mechanism allows the client to specify a FHIR search URL alongside a resource to create. When processing the create, the server will first check if any resource already exists matching the given search. If no resources match, the create proceeds. If a resource does match, no resource is created.

In a standard REST mechanism, conditional creates place the search URL in the In-None-Exist request header. In a FHIR bundle transaction, this URL goes in Bundle.entry.request.ifNoneExist. Like a normal create, we use the HTTP method/verb of POST.

For example:

```
{
   "resourceType": "Bundle",
   "type": "transaction",
   "entry": [ {
    "fullUrl": "urn:uuid:95dbbf93-5829-46ba-9021-2545a1da3aa5",   >
Placeholder ID for Patient
 "resource": {
         "resourceType": "Patient",
         "identifier": [ {
            "system": "http://acme.org/mrns",
            "value": "013872"
         } ],
         "name": [ {
            "family": "Simpson",
            "given": [ "Homer" ]
         } ]
      },
      "request": {
         "method": "POST",
         "url": "Patient",
       "ifNoneExist": "Patient?identifier=http://acme.org/mrns|
013872"   > Search URL for Conditional Create
         }
   }, {
      "fullUrl": "urn:uuid:124ff3c8-f251-4bd9-8c44-cc6568180eae",
      "resource": {
         "resourceType": "Observation",
         "identifier": [ {
            "system": "http://acme.org/obs",
            "value": "46252"
         } ],
         "status": "final",
         "code": {
            "coding": [ {
               "system": "http://loinc",
               "code": "29463-7",
               "display": "Body Weight"
```

```
            } ]
         },
         "subject": {
          "reference": "urn:uuid:95dbbf93-5829-46ba-9021-2545a1da3aa5"  >
Reference to Patient Placeholder ID
            },
         "effectiveDateTime": "2022-02-23",
         "valueQuantity": {
            "value": 67.1,
            "unit": "kg",
            "system": "http://unitsofmeasure.org",
            "code": "kg"
         }
      },
      "request": {
         "method": "POST",
         "url": "Observation",
         "ifNoneExist": "Observation?identifier=http://acme.org/obs|46252"
> Search URL for Conditional Create
          }
   } ]
}
```

Conditional Update

The FHIR Conditional Update mechanism allows a resource to be transmitted to the server for updating, but instead of supplying an ID to update, the client supplies a search URL similar to the URL used for the Conditional Create example.

For more information, see Conditional Create.

Before performing the update, the server first performs the search. If no resources match the search, a new resource is created. If a resource already matches the search, it is updated using the contents in Bundle.entry.resource. In this case, the method/verb will be PUT and the search URL is specified in Bundle.request.url.

For example:

```
{
   "resourceType": "Bundle",
   "type": "transaction",
   "entry": [ {
   "fullUrl": "urn:uuid:95dbbf93-5829-46ba-9021-2545a1da3aa5",  >
Placeholder ID for Patient
      "resource": {
         "resourceType": "Patient",
         "identifier": [ {
            "system": "http://acme.org/mrns",
            "value": "013872"
         } ],
         "name": [ {
            "family": "Simpson",
            "given": [ "Homer" ]
         } ]
      },
```

```
      "request": {
         "method": "PUT",
         "url": "Patient?identifier=http://acme.org/mrns|013872"   >
Search URL for Conditional Update
          }
   }, {
      "fullUrl": "urn:uuid:124ff3c8-f251-4bd9-8c44-cc6568180eae",
      "resource": {
         "resourceType": "Observation",
         "identifier": [ {
            "system": "http://acme.org/obs",
            "value": "46252"
         } ],
         "status": "final",
         "code": {
            "coding": [ {
               "system": "http://loinc",
               "code": "29463-7",
               "display": "Body Weight"
            } ]
         },
         "subject": {
          "reference":
"urn:uuid:95dbbf93-5829-46ba-9021-2545a1da3aa5"    > Reference to
Patient Placeholder ID
             },
         "effectiveDateTime": "2022-02-23",
         "valueQuantity": {
            "value": 67.1,
            "unit": "kg",
            "system": "http://unitsofmeasure.org",
            "code": "kg"
         }
      },
      "request": {
         "method": "PUT",
         "url": "Observation?identifier=http://acme.org/obs|46252"   >
Search URL for Conditional Update
          }
   } ]
}
```

Delete

FHIR logical deletes can be performed as a part of a transaction, as well. In a transaction bundle, deletes will be applied as a group, meaning that if one fails, all will be rolled back.

The following example shows deletes in a FHIR transaction.

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [ {
```

```
  "request": {
      "method": "DELETE",
      "url": "Patient/A0"
    }
      }, {
   "request": {
      "method": "DELETE",
      "url": "Patient/A1"
        }
  } ]
}
```

Patch

The FHIR Patch operation can also be performed in a transaction bundle. When using the FHIR Patch mechanism for patching, the FHIR Patch document is placed in Bundle.entry.resource. In the case of JSON Patch the contents are placed in a Binary resource and then placed into Bundle.entry.resource. In all cases, the HTTP verb/method is PATCH.

The following example shows a simple FHIR Patch in a transaction.

```
{
  "resourceType": "Bundle",
  "type": "transaction",
  "entry": [ {
    "resource": {
 "resourceType": "Parameters", > The FHIR Patch document
      "parameter": [ {
        "name": "operation",
        "part": [ {
          "name": "type",
          "valueCode": "replace"
        }, {
          "name": "path",
          "valueString": "Patient.identifier"
        }, {
          "name": "value",
          "valueIdentifier": {
            "system": "http://new-system",
            "value": "0001"
          }
        } ]
      } ]
        },
    "request": {
      "method": "PATCH",
      "url": "Patient/123" > The identity of the resource to patch
        }
  } ]
}
```

Search parameters

Search parameters are named paths within resources that are indexed by the system so that they can be used to find resources that match a given criteria.

For example, the FHIR specifications define the gender search parameter on the Patient resource, giving it a path of Patient.gender. This means that every time a new Patient resource is created—or an existing one is updated—the value found at path Patient.gender will be indexed. Clients may then use a URL search parameter named gender to find a resource with the given gender.

Request: Search parameters

HTTP Method	GET
URL	`http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?gender=male`

- Default search parameters
  For a general-purpose repository, using the default search parameters is useful since these parameters represent a wide variety of use cases. Additionally, using the default parameters is good for interoperability since clients may expect standard parameters to be supported across different servers.
- Managing search parameters
  Each search parameter is represented by a SearchParameter resource in the database. When Oracle HDR starts for the first time, the database will be preseeded with search parameter resources that correspond to the various default parameters.
- Manual indexing
  It is possible to trigger a manual reindexing of data in the Oracle HDR-FHIR repository.
- Reindex operation
  The **$reindex** operation requests that all data on the server, or a selected subset of the data on the server, be reindexed.

Default search parameters

For a general-purpose repository, using the default search parameters is useful since these parameters represent a wide variety of use cases. Additionally, using the default parameters is good for interoperability since clients may expect standard parameters to be supported across different servers.

The default search parameters are:

- *name* (Search for patient by name)
- *birthdate* (Search for patient by date of birth)
- *identifier* (Search for patient by identifier)

However, it is often useful to customize the supported search parameters. For example, you may want to:

- Add additional search parameters that will index fields that do not have a standard search parameter defined.
- Add additional search parameters that will index extensions used by your clients.
- Disable search parameters that are not used in order to improve performance and conserve space (disabling unnecessary search parameters can have a dramatic impact on write performance in some cases).

Managing search parameters

Each search parameter is represented by a SearchParameter resource in the database. When Oracle HDR starts for the first time, the database will be preseeded with search parameter resources that correspond to the various default parameters.

Using FHIR Endpoint—If you want to customize the available search parameters, there are several ways to do so.

For example, POSTing the following resource to an Oracle HDR-FHIR server that supports custom search parameters creates a new search parameter on the **Patient** resource named **eyecolour**.

```
{
        "resourceType": "SearchParameter",
        "title": "Eye Colour",
        "base": [ "Patient”],
        "status": "active",
        "code": "eyecolour",
        "type": "token",
        "expression": "Patient.extension('http://acme.org/eyecolour')",
        "xpathUsage": "normal"
}
```

Manual indexing

It is possible to trigger a manual reindexing of data in the Oracle HDR-FHIR repository.

To perform a manual reindex, invoke the following operation at the server (root) level of the FHIR Endpoint (that is, the base URL of the FHIR endpoint). To specify parameters, a resource of type Parameters must be included as the body of a POST request.

- **Operation:**

  $mark-all-resources-for-reindexing

- **Parameter:**

  type: Optional. If supplied, specifies the specific resource type to reindex. If not specified, all resource types are reindexed.

See the following examples for more details.

To perform reindexing on specific resource type, ex Patient, here is the sample payload. We can also include multiple resources.

```
Content-Type: application/fhir+json
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "type",
    "valueString": "Patient"
  } ]
}
```

For all resource types, here is the payload example.

```
Content-Type: application/fhir+json
{
  "resourceType": "Parameters",
  "parameter": []
}
```

The status of the reindexing will be recorded in this HFJ_RES_REINDEX_JOB table while re-indexing. The record will be purged once the reindexing operation is completed.

Reindex operation

The **$reindex** operation requests that all data on the server, or a selected subset of the data on the server, be reindexed.

The **$reindex** operation can be called in one of two ways, either with a list of **urls** to be reindexed or with everything=true. If everything=true then the urls parameter is ignored and all resources will be reindexed. Otherwise, only the resources matching the urls are reindexed. The reindex operation uses search URLs to identify resources that should be reindexed. A search URL takes the form:

```
{resourceType}?[optional search parameters]
```

For example, the following URL indicates that all resources of type Patient should be reindexed:

```
Patient?
```

The following URL indicates that all Observations whose subject is in active should be reindexed:

```
Observation?subject.active=true
```

If no URLs are included, then all resources of all types will be reindexed.

Example 1: Reindex a specific set of URLs

```
POST <baseURL>/$reindex
Content-Type: application/fhir+json

{
```

```
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "url",
    "valueString": "Patient?active=true"
  }, {
    "name": "url",
    "valueString": "Observation?subject.active=true"
  } ]
}
```

Example 2: Reindex all Practitioner and all Patient resources

```
POST baseURL/$reindex
Content-Type: application/fhir+json

{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "url",
    "valueString": "Practitioner?"
  }, {
    "name": "url",
    "valueString": "Patient?"
  } ]
}
```

Example 3: Reindex all resources with no parameters or with everything = true

```
POST baseURL/$reindex
```

OR

```
POST baseURL/$reindex
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "everything",
    "valueBoolean": "true"
  }]
}
```

- Reindex response
  The **$reindex** operation creates a batch job that can be executed asynchronously.

Reindex response

The **$reindex** operation creates a batch job that can be executed asynchronously.

When you invoke a reindex operation, the system responds with a jobId and that can be used to know the status of the reindex operation.

Example Response:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "jobId",
    "valueString": "18256337-5a0d-4b9d-a2af-0927d08201c6"
  } ]
}
```

You can use the **$export-poll-status** operation to know the status of the job using the following call:

```
GET baseURL/$export-poll-status?_jobId=18256337-5a0d-4b9d-
a2af-0927d08201c6
```

Search parameter features

This section describes optional features that you can enable or disable for optimizing how searching works on Oracle HDR-FHIR server.

- Index missing search parameter (: missing)
  When enabled, the **: missing** modifier can be used on a SearchParameter to find resources where that search parameter has found (or not found) any data in the given resource.
- Index contained resources
  FHIR search parameters of type reference supports a concept called chaining.

Index missing search parameter (: missing)

When enabled, the **: missing** modifier can be used on a SearchParameter to find resources where that search parameter has found (or not found) any data in the given resource.

This feature defaults to: **Disabled**

Parameter name: **enable_index_missing_fields**

For example, take the Patient:birthdate search parameter, which indexes the value of Patient.birthDate. The following URL finds all patients having a birthdate:

Request: Search query for all patients with a birth date

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?birthdate:missing=false

Request: Search query for all patients without a birth date

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient?birthdate:missing=true

Indexing for missing search parameters can add a great deal of extra index space, and slow down write operations on servers with many enabled search parameters. This feature is therefore disabled by default and must be specifically enabled if it is needed.

Index contained resources

FHIR search parameters of type reference supports a concept called chaining.

This feature defaults to: **Disabled**

Parameter name: **enable_index_contained_resource**

Consider the following resources:

Patient resource:

```
{
  "resourceType": "Patient",
  "id": "1",
  "name": [{
    "family": "Smith"
    }]
}
```

Observation resource with reference to Patient:

```
{
  "resourceType": "Observation",
  "id": "2",
  "subject": { "reference": "Patient/1" }
}
```

Request: Search query for an Observation resource

With the Patient and Observation resources stored in your repository, the following search returns the Observation resource.

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation?subject.name=Smith

This works because of the chain, which is the dot notation followed by a second search parameter. This syntax means "find me any Observation resources where the subject reference links to a target resource where a search for **name=Smith** would match."

By default, chained searches only consider top-level resources. Consider the following Observation resource with a reference to a contained Patient.

```
{
  "resourceType": "Observation",
  "id": "3",
  "contained": [
    {
      "resourceType": "Patient",
      "id": "contained-patient",
      "name": [{
        "family": "Smith"
      }]
    }
  ],
  "subject": { "reference": "#contained-patient" }
}
```

The search given above will not return this Observation, because the contained resource has not been included in the search indexes. If the Index Contained Resources feature is enabled, indexes will be generated for the contained resources, and Observation/3 would be included in the search results.

Searching for data

This section contains information about methods for searching for data in Oracle HDR FHIR.

- FHIR Search extensions
  In addition to implementing most of the FHIR Search specification, Oracle HDR FHIR server implements the following extensions: _source, %now, and %today.

FHIR Search extensions

In addition to implementing most of the FHIR Search specification, Oracle HDR FHIR server implements the following extensions: _source, %now, and %today.

_source—An additional search parameter called _source can be used to search for resources based on information about the system or request that created the resource.

Request: Search parameter _source

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Account/99952?_source=#sOx1E9Vj7ntsPAaH

%now—Date searches can be performed relative to "now" using the %now parameter value.

For example, to search for Procedures with a date later than now, you can search for /Observation?date=le%now.

> **Note:**
>
> The "%" needs to be URL escaped so the actual URL will be /Observation?date=le%25now.

Request: Search parameter _source now

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/ Observation?date=le%25now

%today—Similarly date searches can be performed relative to "today" using the "%today" parameter value. "%today" works the same as "%now" except that it searches as a "date" type as opposed to a "dateTime" type.

Creating data

This section describes creating data in Oracle HDR FHIR repository.

- Validating references and referential integrity
  When a resource is written to the repository (create/update/etc), local resource references that are indexed by at least one SearchParameter can be checked to ensure that the target of the reference is valid.
- Transactions and submitting bundles
  If you **POST** a bundle message to **[baseUrl]/Bundle** you are submitting the bundle for storage as-is.
- Auto-creating reference targets
  Often when batch processing data from multiple sources, you have data from one source that has references to data from other sources.
- FHIR Transaction with conditional create
  The conditional create is roughly described as “use an existing resource that matches specific criteria if one exists (and do not modify that resource), or create a new one if not.”
- Auto-create placeholders for reference targets
  If the Auto-Create Placeholder Reference Targets setting (auto_create_placeholder_reference_targets) is enabled in the Oracle HDR FHIR server, it is possible to have the server automatically create an empty "placeholder" resource with a pre-assigned ID.
- Auto-create placeholder reference targets with identifier
  If the Auto-Create Placeholder Reference Targets setting is enabled in the FHIR Storage module configuration (as shown above), and the Allow Inline Match URL References Enabled setting is also enabled, you can refine the behavior shown above further.

Validating references and referential integrity

When a resource is written to the repository (create/update/etc), local resource references that are indexed by at least one SearchParameter can be checked to ensure that the target of the reference is valid.

References that are not indexed by at least one SearchParameter are never checked for target existence. Non-local references (that is, references where the base URL of the reference target refers to a different FHIR server) are never checked for target existence.

Reference validation

If the config Enforce Referential Integrity on Write (enforce_referential_integrity_on_write) is enabled, reference targets are checked.

For example, if a patient contains a reference to managing organization Organization/HealthOrg but HealthOrg is not a valid ID for an organization on the server, the operation is blocked unless this property has been disabled.

This property can cause confusing results for clients of the server since searches, includes, and other FHIR features may not behave as expected when referential integrity is not preserved. In particular, resource references to target resources that do not exist at the time that the source resource is created **will not be indexed**, even if the target resource is created later. **Disable with caution.**

Referential integrity

If the config Enforce Referential Integrity on Delete (enforce_referential_integrity_on_delete) is enabled on HDR FHIR Server, resources can only be deleted if there are no other resources with indexed references to the candidate resource for deletion.

Transactions and submitting bundles

If you **POST** a bundle message to **[baseUrl]/Bundle** you are submitting the bundle for storage as-is.

In other words, the bundle is stored as a Bundle, and the contents inside aren't looked at by the server (aside from any validation that is enabled). This mode is generally used to store Bundle resources with **Bundle.type** values such as **document** and **collection**. In its default configuration, Oracle HDR prohibits storing a Bundle with a type value of transaction or batch as this is generally a sign that the client is attempting to perform the operations described below but with an incorrect request URL.

- **Bundle Type—Transaction**
  - If you **POST** to [baseUrl] and your Bundle has a **Bundle.type** value of **transaction** you are performing a FHIR "transaction operation", meaning that all of the individual resources inside the bundle will be processed. It is also possible to include other REST operations such as searches in this kind of bundle. The processing works as an atomic unit, meaning that if anything fails (for example, invalid data in an individual element) the entire thing will be rolled back.
  - This operation is referred to as an **FHIR Transaction** operation.
- **Bundle Type—Batch**

- If you **POST** to [baseUrl] and your Bundle has a **Bundle.type** value of **batch**, the same processing as the transaction applies, except those individual operations are executed in individual database transactions, so an individual failure doesn't cause the entire operation to be rolled back. In this case, the response Bundle returned by the server includes status entries indicating the outcome for the individual operations within. Note that the batch operation does require the entire Bundle to be valid FHIR at a minimum. This means that it can't have non-existent resource types in it, malformed datatypes, and so on.
- This operation is referred to as an **FHIR Batch** operation.

Auto-creating reference targets

Often when batch processing data from multiple sources, you have data from one source that has references to data from other sources.

For example, a collection of Observation resources could be imported from a lab system data source at the same time that a collection of Patient resources is created from a patient administration data source. The Observation resources would have references to the Patient resources. Under ideal conditions, the Patient resource would process first and be present for the Observation to link to. In the real world however, often it is hard to control the order that transactions occur, and so it might be possible for an Observation to be processed before its Patient. By default, this causes an error since the Observation has an invalid reference, and nothing is stored.

The following topics describe helpful strategies for solving this issue:

- FHIR Transaction with conditional create
- Auto-create placeholders for reference targets
- Auto-create placeholder reference targets with identifier

FHIR Transaction with conditional create

The conditional create is roughly described as "use an existing resource that matches specific criteria if one exists (and do not modify that resource), or create a new one if not."

The specific criteria in question can be any set of FHIR search parameters that could be used to otherwise locate the resource to use. The resource identifier field/search parameter is often used for this purpose, but other search parameters can also be used.

In an FHIR Transaction operation, an operation is performed using a conditional create.

This involves creating a Transaction Bundle with the following properties (an Observation being created with a reference to a Conditionally Created Patient is being used for this example):

- One or more entries containing an Observation resource with a request.method value of POST. This means that the server should create the new Observation resources, and automatically assign them new IDs.
- The Observation resources contain a reference where the target is the fullUrl UUID for the Patient entry. If the Patient target was created (because it did not already exist) the reference will automatically be replaced with a reference to the newly created resource. If the Patient target was not created (because it already existed), the reference will automatically be replaced with a reference to the pre-existing Patient resource.
- An entry containing a Patient resource with:

- A **request.method** value of POST
- A **fullUrl** value containing a temporary UUID. This is used as the target for references to this resource from other resources. A **request.ifNoneExist** value containing a search URL that could be used to find this resource (in the example below, a search for the Patient by identifier). This indicates to the server that this resource should only be created if no existing resource already matches the given search criteria.

Example: Transaction Bundle

An example Transaction Bundle is shown below. It should be POSTed to the root of the FHIR Endpoint module server.

```
"resourceType": "Bundle",
  "meta": {
    "lastUpdated": "2014-08-18T01:43:30Z"
  },
  "type": "transaction",
  "entry": [
    {
      "fullUrl": "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a",
      "resource": {
        "resourceType": "Patient",
        "text": {
          "status": "generated",
          "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Some
narrative</div>"
        },
        "active": true,
        "name": [
          {
            "use": "official",
            "family": "Chalmers",
            "given": [
              "Peter",
              "James"
            ]
          }
        ],
        "gender": "male",
        "birthDate": "1974-12-25"
      },
      "request": {
        "method": "POST",
        "url": "Patient"
      }
    },
    {
      "fullUrl": "http://example.org/fhir/Patient/100058",
      "resource": {
        "resourceType": "Patient",
        "id": "123",
        "text": {
          "status": "generated",
          "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Some
```

```
narrative</div>"
        },
        "active": true,
        "name": [
          {
            "use": "official",
            "family": "Chalmers",
            "given": [
              "Peter",
              "James"
            ]
          }
        ],
        "gender": "male",
        "birthDate": "1974-12-25"
      },
      "request": {
        "method": "PUT",
        "url": "Patient/100058"
      }
    },
    {
      "request": {
        "method": "GET",
        "url": "Patient?name=Peter"
      }
    },
    {
      "request": {
        "method": "GET",
        "url": "Patient/1",
        "ifNoneMatch": "W/\"4\"",
        "ifModifiedSince": "2015-08-31T08:14:33+10:00"
      }
    }
  ]
}
```

Auto-create placeholders for reference targets

If the Auto-Create Placeholder Reference Targets setting (auto_create_placeholder_reference_targets) is enabled in the Oracle HDR FHIR server, it is possible to have the server automatically create an empty "placeholder" resource with a pre-assigned ID.

This technique is somewhat less complex than the example described in FHIR Transaction with conditional create, since it does not require a transaction bundle to be created. With this technique, the ID (not the identifier) of the target resource must be known. For example, the following payload could be POSTed to [baseUrl]/Observation, and would result in the creation of an empty resource with the ID Patient/ABC if one does not already exist. Note that if you want to be able to use this technique with purely numeric resource IDs you will also need to adjust the Client ID Mode.

Example: Auto-create placeholders for reference targets

```
{
  "resourceType": "Observation",
  "status": "final",
  "code": {
    "coding": [ {
      "system": "http://loinc.org",
      "code": "789-8"
    } ]
  },
  "subject": {
    "reference": "Patient/ABC"
  },
  "valueQuantity": {
    "value": 4.12,
    "system": "http://unitsofmeasure.org",
    "code": "10*12/L"
  }
}
```

Auto-create placeholder reference targets with identifier

If the Auto-Create Placeholder Reference Targets setting is enabled in the FHIR Storage module configuration (as shown above), and the Allow Inline Match URL References Enabled setting is also enabled, you can refine the behavior shown above further.

In this case, it is possible to use an inline match URL instead of a hardcoded resource ID, and you can then achieve similar behavior to the Transaction Bundle use case.

Consider the following Observation being POSTed to /Observation.

Example: Auto-create placeholders for reference targets with identifiers

```
{
 "resourceType": "Observation",
 "status": "final",
 "code": {
  "coding": [ {
   "system": "http://loinc.org",
   "code": "789-8"
  } ]
 },
 "subject": {
  "reference": "Patient?identifier=http://example|10008",
  "identifier": {
   "system": "http://example",
   "value": "10008"
  }
 },
 "valueQuantity": {
  "value": 4.12,
  "system": "http://unitsofmeasure.org",
  "code": "10*12/L"
 }
}
```

In this case, the reference is treated as a local search (in this case for a Patient with the identifier included in the inline match URL) and executed as such.

If the search finds zero results, a new Patient resource is created. If the inline match URL uses an identifier as it does here, Patient.identifier is populated with the inline match URL's identifier system and value. The reference is then automatically replaced with a reference to this new Patient.

If the search finds one result, the reference is automatically replaced with a reference to the found Patient and no placeholder reference target is created.

> **Note:**
>
> We use Patient as an example; this applies to any reference targets that include an Identifier element.

Reading data

This section describes methods for reading data from the Oracle HDR FHIR Repository.

- Diff operation
  Use the **$diff** operation to generate a differential between two versions of a resource, or between two different resources of the same type.

- Diff at Instance Level
  When you invoke the $diff operation at the instance level (meaning it is invoked on a specific resource ID), it compares two versions of the given resource.
- Diff at Type Level
  When the $diff operation is invoked at the type level (meaning it is invoked on a specific resource type), it will compare two different resources of the same type.
- $everything operation
  The Oracle HDR-FHIR jpa server supports the Patient/$everything operation and accepts all the IN parameters defined in the documentation. Additionally, Oracle HDR allows you to provide an _id parameter, in order to filter the set of patients you wish to get everything for.

Diff operation

Use the **$diff** operation to generate a differential between two versions of a resource, or between two different resources of the same type.

Differentials generated by this operation are in FHIR Patch format.

In generated differentials, where a value has changed (that is, a replace operation), an additional part value is present on the given operation called **previousValue**. This part shows the value as it was in the from version of the resource.

Diff at Instance Level

When you invoke the $diff operation at the instance level (meaning it is invoked on a specific resource ID), it compares two versions of the given resource.

Parameters

- **fromVersion=[versionId]**—Optional. If specified, compare using this version as the source. If not specified, the immediately previous version is compared.
- **includeMeta=true**—Optional. If specified, changes to Resource.meta are included in the diff. This element is omitted by default.

Request: Reading Data with a Diff at Instance Level

HTTP Method	GET
URL	`http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/100065/$diff`

Response: Reading Data with a Diff at Instance Level

The server produces a response similar to the following:

```
{
 "resourceType": "Parameters",
 "parameter": [ {
  "name": "operation",
  "part": [ {
   "name": "type",
   "valueCode": "replace"
  }, {
   "name": "path",
   "valueString": "Patient.name.family"
  }, {
   "name": "previousValue",
   "valueId": "Smyth"
  }, {
   "name": "value",
   "valueId": "SmithB"
  } ]
 } ]
}
```


Diff at Type Level

When the $diff operation is invoked at the type level (meaning it is invoked on a specific resource type), it will compare two different resources of the same type.

Parameters

- **from=[reference]**—Specifies the source of the comparison. The value must include a resource type and a resource ID, and can optionally include a version (for example, Patient/123 or Patient/123/_history/2).
- **to=[reference]**—Specifies the target of the comparison. The value must include a resource type and a resource ID, and can optionally include a version (for example, Patient/123 or Patient/123/_history/2).
- **includeMeta=true**—Optional. If specified, changes to Resource.meta are included in the diff. This element is omitted by default.

Request: Reading Data with a Diff at Type Level

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/$diff?from=Patient/100065&to=Patient/100004

Response: Reading Data with a Diff at Type Level

The server produces a response similar to the following:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "operation",
    "part": [ {
      "name": "type",
      "valueCode": "replace"
    }, {
      "name": "path",
      "valueString": "Patient.id"
    }, {
      "name": "previousValue",
      "valueId": "100065"
    }, {
      "name": "value",
      "valueId": "100004"
    } ]
  } ]
}
```

$everything operation

The Oracle HDR-FHIR jpa server supports the Patient/$everything operation and accepts all the IN parameters defined in the documentation. Additionally, Oracle HDR allows you to provide an _id parameter, in order to filter the set of patients you wish to get everything for.

The following requests are all equivalent, and these example queries fetch everything for Patient/1, Patient/2, and Patient/3.

Request: $everything operation using GET

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/$everything

Request: $everything operation using GET with alternate _id parameter

The server produces a response similar to the following:

HTTP Method	GET
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/$everything?_id=1,2,3

Request: $everything operation using a POST

HTTP Method	POST
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/$everything

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "_id",
    "valueString": "1"
  }, {
    "name": "_id",
    "valueString": "2"
  },{
    "name": "_id",
    "valueString": "3"
  }
  ]
}
```

Updating data

This section describes information about methods for updating data in the HDR FHIR Repository.

- Patching data
  You can patch data to make a small change to a resource without needing to re-upload the entire content. For example, a resource status field might be changed by an **application** with no other changes needed.
- Tag retention
  According to the FHIR rules on updating resources, by default when a resource is updated, any tags and security labels are carried forward even if they are not explicitly listed in the new version.

Patching data

You can patch data to make a small change to a resource without needing to re-upload the entire content. For example, a resource status field might be changed by an **application** with no other changes needed.

Oracle HDR FHIR support the following syntaxes:

- **FHIR Patch**—This is the most expressive syntax for patching and is recommended for use. It uses a format described in the FHIR specification as FHIR Patch.
- **JSON Patch**—This syntax expresses a change set in JSON using RFC 6902.

Patch using FHIR Patch

The FHIR Patch format can be used to specify rules for inserting, modifying, and removing data from FHIR resources. See FHIR Patch for details about the format.

Example: FHIR Patch used to update a patient birth date

The following example shows a FHIR Patch being used to update a patient birth date.

Table 11-5 FHIR Patch to update patient birth date

Category	Description
HTTP Method	PATCH
Content-Type	application/fhir+json
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/123/

Request Body	<pre>{
	  "resourceType": "Parameters",
	  "parameter": [{
	  "name": "operation",
	  "part": [{
	  "name": "type",
	  "valueCode": "replace"
	  }, {
	  "name": "path",
	  "valueString":
	"Patient.birthDate"
	  }, {
	  "name": "value",
	  "valueDate": "1975-09-09"
	  }]
	  }]
	}</pre>

Example: FHIR Patch as part of FHIR transaction

A FHIR Patch may also be submitted as a part of a FHIR transaction.

Table 11-6 FHIR Patch as part of FHIR transaction

Category	Description
HTTP Method	POST
Content-Type	application/fhir+json
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/

Table 11-6 (Cont.) FHIR Patch as part of FHIR transaction

Category	Description

Request Body	<pre>{
	  "resourceType": "Bundle",
	  "type": "transaction",
	  "entry": [{
	  "fullUrl": "Patient/123",
	  "resource": {
	  "resourceType":
	"Parameters",
	  "parameter": [{
	  "name": "operation",
	  "part": [{
	  "name": "type",
	  "valueCode": "replace"
	  }, {
	  "name": "path",
	  "valueString":
	"Patient.birthDate"
	  }, {
	  "name": "value",
	  "valueDate":
	"1930-01-01"
	  }]
	  }]
	  },
	  "request": {
	  "method": "PATCH",
	  "url": "Patient/123"
	  }
	  }]
	}</pre>

Example: Patch Using JSONPatch

The following example shows a JSONPatch being used to update an Observation status:

Table 11-7 Patch Using JSONPatch

Category	Description
HTTP Method	PATCH
Content-Type	application/json-patch+json
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Observation/123

Table 11-7 (Cont.) Patch Using JSONPatch

Category	Description

Request Body	<pre>[
	  {
	  "op": "replace",
	  "path": "/status",
	  "value": "in-progress"
	  }
	]</pre>

> **Note:**
>
> If you are using "op": "add", you need to suffix the path with /- (e.g. "path": "/identifier/-").

Example: JSONPatch also be submitted as a part of a FHIR transaction

A JSONPatch may also be submitted as a part of a FHIR transaction using a Binary resource as the payload in order to update the contents.

Table 11-8 JSONPatch submitted as part of a FHIR transaction

Category	Description
HTTP Method	POST
Content-Type	application/fhir+json
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/

Table 11-8 (Cont.) JSONPatch submitted as part of a FHIR transaction

Category	Description

Request Body	{
	"resourceType": "Bundle",
	"type": "transaction",
	"entry": [
	{
	"fullUrl": "Patient/1",
	"resource": {
	"resourceType": "Binary",
	"contentType":
	"application/json-patch+json",
	"data":
	"WyB7ICJvcCI6InJlcGxhY2UiLCAicGF0a
	CI6Ii9hY3RpdmUiLCAidmFsdWUiOmZhbHN
	lIH0gXQ=="
	},
	"request": {
	"method": "PATCH",
	"url": "Patient/1"
	}
	}
	]
	}

Tag retention

According to the FHIR rules on updating resources, by default when a resource is updated, any tags and security labels are carried forward even if they are not explicitly listed in the new version.

For example, suppose a resource is created by a client, and in that resource a tag "foo" is listed in **Resource.meta.tag**. Then, an update is performed by a client but this update does not contain a value in **Resource.meta**.

According to the FHIR rules, in this case the tag is copied to the new version of the resource even though it was not explicitly requested.

If a client wishes to override this behavior, they may do so using the **X-Meta-Snapshot-Mode** header. This header indicates that Tags and/or Security Labels and/or Profile Declarations should be treated as snapshots, meaning that any values not already present should be removed.

The value is a comma-separated list containing the metadata components that should be treated in snapshot mode:

- TAG—Resource tags

- PROFILE—Resource profile declarations
- SECURITY_LABEL—Security Labels.

Example: Tag retention—Metadata components using snapshot mode

```
X-Meta-Snapshot-Mode: TAG, PROFILE, SECURITY_LABEL
```

Deleting data

The FHIR delete operation performs a "logical" delete. This means that data is not physically removed from the database.

For example, Patient resource with ID 123 is created (via an HTTP POST /Patient) and subsequently deleted (via an HTTP DELETE Patient/123). This causes a second version of the Patient/123 resource to be created with version Patient/123/_history/2 that is marked as deleted.

This patient no longer appears in search results and attempts to read the resource (using an HTTP GET Patient/123) fails with an "HTTP 410 Gone" response.

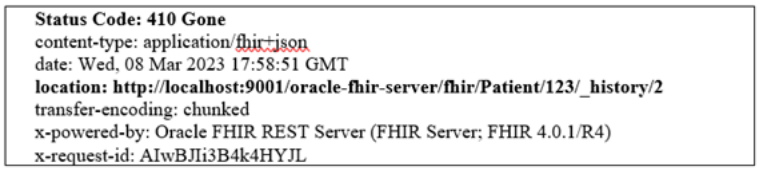
The original content of the resource is not destroyed, however. It can still be found using two FHIR operations:

- Using a FHIR version-specific read: GET Patient/123/_history/1
- Using a FHIR instance-history: GET Patient/123/_history

Note that HTTP 410 Gone responses include a Location header containing the fully qualified resource ID as well as the version ID.

Example: DELETE 410 Gone response with a Location Header

Figure 11-1 Status Code: 410 Gone Response



Example: DELETE 410 Gone response with latest non-deleted version

In this example, we can see that the deleted version of the resource is version 2. This means that the last non-deleted version is version 1, and this could be accessed using a version-specific read to the following URL:

```
http(s)://HOSTNAME: PORT/oracle-fhir-server/fhir/Patient/49957/_history/1
```

- Deletes and referential integrity
  The FHIR delete operation performs a "logical" delete. This means that data is not physically removed from the database.
- Transactional delete
  Use the FHIR Transaction operation to delete multiple resources at the same time.
- Referential integrity
  By default, Oracle HDR-FHIR server blocks the deletion of a resource if any other resources have indexed references to the resource being deleted.
- Cascading deletes
  By enabling the cascading delete, a user can perform the delete on parent resource, then all the corresponding child resources will be deleted as well.
- The $expunge operation
  In some cases, you may need to completely delete data from the HDR-FHIR repository after performing the DELTE operation. In those cases, the $expunge operation will be used and is a powerful operation that can physically delete old versions of resources, deleted resources, or even all data in the database.

Deletes and referential integrity

The FHIR delete operation performs a "logical" delete. This means that data is not physically removed from the database.

For example, suppose the HDR-FHIR server containing a patient resource with the ID Patient/1. And HDR-FHIR server also contains Encounter/1 and Encounter/2, as well as Observation/3 and MedicationAdministration/4, and all these resources have a reference to the resource Patient/1. We will call these resources are child resources.

If you try to DELETE Patient/1 (using a standard FHIR DELETE operation), the request is denied, assuming that there is a resource link between the child resources and the patient.

This results in an HTTP 409 Conflict with a response similar to the following:

Example: DELETE 410 Gone response with an HTTP 409 Conflict


```
{
 "resourceType": "OperationOutcome",
 "issue": [
  {
   "severity": "error",
   "code": "processing",
   "diagnostics": "Unable to delete Patient/1 because at least one resource has a reference
to this resource. First reference found was resource Observation/3 in path
Observation.subject.where(resolve() is Patient)"
  }
 ]
}
```


If you want to force a delete of Patient/1, you have several options:

- You can manually delete all the child resources before trying to delete the Patient.

- You can disable referential integrity checking in the HDR-FHIR server by changing enforce referential integrity check in delete setting on the FHIR Server. This means that any delete on resource is allowed, even if other resources still have references left.
- You can use a transactional delete.
- You can use cascading deletes.

Transactional delete

Use the FHIR Transaction operation to delete multiple resources at the same time.

This is useful if you have chains or collections of resources to delete at once, but also can be used to delete circular references.

To delete multiple resources in a transaction, POST a Bundle such as the following to the root of your FHIR endpoint.

Example: Transactional delete

```
{
 "resourceType": "Bundle",
 "type": "transaction",
 "entry": [
  {
   "request": {
    "method": "DELETE",
    "url": "Organization/1"
   }
  },
  {
   "request": {
    "method": "DELETE",
    "url": "Organization/2"
   }
  }
 ]
}
```

Referential integrity

By default, Oracle HDR-FHIR server blocks the deletion of a resource if any other resources have indexed references to the resource being deleted.

For example:

- The resource Patient/102 has been saved in the repository, and a second resource Observation/558 is then saved as well, where the Observation.subject reference is a reference to the Patient.
- In this situation, attempts to delete Patient/102 are blocked unless resources with references to this resource are deleted first (or are deleted as a part of the same transaction in the case of a transactional delete).

Disabling Referential Integrity on Delete

You can disable this referential integrity check on delete operation using below configuration property on the Oracle HDR-FHIR Server.

```
enforce_referential_integrity_on_delete: false
```

If it is enabled, resources can only be deleted if there are no other resources with indexed references to the candidate resource for deletion.

Disabling Referential Integrity on Write

You can disable this referential integrity check on write operation using below configuration property on the Oracle HDR-FHIR Server.

```
enforce_referential_integrity_on_write.: false
```

If it is enabled, reference targets will be checked. For example, if a patient contains a reference to managing organization Organization/FOO but FOO is not a valid ID for an organization on the server then the operation will be blocked unless this property has been disabled. This property can be used with caution since searches and other FHIR features may not behave as expected when referential integrity is not preserved. In cases, resource references to target resources that do not exist at the time that the source resource is created will not be indexed, even if the target resource is created later.

Cascading deletes

By enabling the cascading delete, a user can perform the delete on parent resource, then all the corresponding child resources will be deleted as well.

In order to perform a cascading delete:

- First, the **`allow_cascading_deletes`** property must be enabled on the HDR-FHIR Server.
- And then, to perform a cascaded delete, the client HTTP request must include either a special URL parameter (_cascade) or a special header to indicate that a cascading delete is desired.

The following example shows how to delete using a URL parameter.

Table 11-9 Delete using URL parameter

HTTP Method	DELETE
URL	`http(s)://<HOSTNAME>:<PORT>/oracle-fhir-server/fhir/Patient/100004?_cascade=delete`

The following example shows how to delete using HTTP header.

Table 11-10 Delete using HTTP Header

HTTP Method	DELETE
URL	`http(s)://<HOSTNAME>:<PORT>/oracle-fhir-server/fhir/Patient/100004?_cascade=delete`
Http Header	`X-cascade.delete`

The $expunge operation

In some cases, you may need to completely delete data from the HDR-FHIR repository after performing the DELTE operation. In those cases, the $expunge operation will be used and is a powerful operation that can physically delete old versions of resources, deleted resources, or even all data in the database.

Table 11-11 Input Parameters

Name	Type	Usage	Default
limit	Number	This parameter specifies the maximum number of entries (resource versions and/or resources) that will be deleted in a single batch before exiting.	1000
expungeDeletedResources	Boolean	If set to true, deleted resources will be expunged (including all previous versions of the resource).	false
expungePreviousVersions	Boolean	If set to true, non-current versions of resources will be expunged.	false
expungeEverything	Boolean	If set to true, current versions of resources will also be expunged.	false

Instance level expunge

You can invoke the $expunge operation against a single resource instance, or even an individual version of a resource instance. If invoked at the instance level (shown below), previous versions of the resource may be deleted (if expungePreviousVersions is set to true) and the current version may be deleted (if the resource is deleted and expungeDeletedResources is set to true).

Request: Instance level expunge against a single resource instance

HTTP Method	POST
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/100004/$expunge
HTTP Header	Content-Type: application/fhir+json
Content	{ "resourceType": "Parameters", "parameter": [{ "name": "limit", "valueInteger": 1000 },{ "name": "expungeDeletedResources", "valueBoolean": true },{ "name": "expungePreviousVersions", "valueBoolean": true }] }

Request: Instance level expunge at the instance version level

You can also invoke the $expunge operation at the instance version level (shown below). Use this to expunge an individual version of a resource without affecting other versions.

HTTP Method	POST
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/100004/_history/2/$expunge
HTTP Header	Content-Type: application/fhir+json
Content	{ "resourceType": "Parameters", "parameter": [{ "name": "expungeDeletedResources", "valueBoolean": true }] }

Type level expunge

You can invoke the $expunge operation at the type level. In this mode, all resources of a given type are processed with the same rules as at the instance level.

Request: Type level expunge at the type level

HTTP Method	POST
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient/$expunge
HTTP Header	Content-Type: application/fhir+json
Content	{ "resourceType": "Parameters", "parameter": [{ "name": "expungeDeletedResources", "valueBoolean": true },{ "name": "expungePreviousVersions", "valueBoolean": true }] }

System level expunge

You can invoke the $expunge operation at the system level. In this mode, all resources on the server are processed with the same rules as at the instance level.

Request: System level expunge

HTTP Method	POST
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/$expunge
HTTP Header	Content-Type: application/fhir+json
Content	{ "resourceType": "Parameters", "parameter": [{ "name": "expungeDeletedResources", "valueBoolean": true },{ "name": "expungePreviousVersions", "valueBoolean": true }] }

Delete expunge

If you need to quickly delete all data associated with a set of resources, you can combine the **DELETE and $expunge** operations into a single step. Oracle HDR FHIR supports this combination.

You can call the usual DELETE with a special parameter **`_expunge=true`**. This results in starting a Delete Expunge Batch Job that deletes and expunges the requested details in the background.

In order to perform a Delete Expunge, three settings need to be enabled on the Oracle HDR FHIR server:

- Expunge Enabled (**expunge_enabled=true**)
- Delete Expunge Enabled (**delete_expunge_enabled=true**)
- Allow multiple Delete Enabled (**allow_multiple_delete=true**)

When the _expunge parameter is provided to the DELETE operation, the matched resources and all of their history will be both deleted and expunged from the database. This will perform considerably faster than doing the delete and expunge separately.

DELETE with _expunge=true

The following example shows how to perform a delete expunge using the DELETE method:

Request: DELETE with _expunge=true

HTTP Method	DELETE
URL	http(s)://HOSTNAME:PORT/oracle-fhir-server/fhir/Patient? id=100104& expunge=true

Binary Access Operations

In many cases, resources such as **DocumentReference** are used to store large files such as scanned PDFs and images. These resources use the Attachment datatype, which ultimately stores a content type and a base 64 encoded representation of the binary content.

To deal with these binary content, HDR FHIR provides two custom FHIR operations that can be used to interact directly with binary content contained within resources such as **DocumentReference**. These operations can be used both to write and read back binary content.

These operations can be enabled/disabled using the **binary_storage_enabled** property. If this property is enable, the plain Binary Access Provider class can be registered with the server to provide the $binary-access-read and $binary-access-write operations that can be used to access attachment data as a raw binary.

- Binary Access Write Operation ($binary-access-write)
  The process of writing a binary payload using the Binary Access Write Operation is done in a two-step process.
- Binary Access Read Operation ($binary-access-read)
  The Binary Access Read Operation can be used to read back binary content from Attachment elements in a similar way to the write operation.

Binary Access Write Operation ($binary-access-write)

The process of writing a binary payload using the Binary Access Write Operation is done in a two-step process.

First, the resource must be created on the server, with a placeholder Attachment that will be populated later.

Example: POST DocumentReference resource with a placeholder Attachment.

```
POST http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/
DocumentReference/DocumentReference
Content-Type: application/fhir+json


{
  "resourceType": "DocumentReference",
    "subject": {
        "reference": "Patient/P123"
    },
  "content": [
    {
      "attachment": {
        "contentType": "image/jpeg"
      }
    }
  ]
}
```

The server will reply with a **content-location** header containing the ID of the newly created resource.

```
content-location: http://localhost:7001/oracle-fhir-server/fhir/
DocumentReference/499952/_history/1
```

This ID is then used in the Binary Access Write Operation to set the binary content.

Second, the Binary Access Write Operation is invoked to directly store the content.

The path parameter, that specifies a FHIRPath expression to the attachment element within the DocumentReference resource. It is important to provide the appropriate content type via the Content-Type header in the operation HTTP request.

```
POST : http://localhost:7001/oracle-fhir-server/fhir/DocumentReference/
499952/$binary-access-write?path=DocumentReference.content.attachment
```

Content-Type: image/png

HTTP Body - (... binary content ...)

> **Note:**
>
> The below property decides whether to store the resource as plain text in the RES_TEXT_VC column of resource extended tables.

inline_resource_storage_below_size: 0

If the resource size is below the size as set in the above property, it goes RES_TEXT_VC column, otherwise it goes to RES_TEXT as compressed blob.

Binary Access Read Operation ($binary-access-read)

The Binary Access Read Operation can be used to read back binary content from Attachment elements in a similar way to the write operation.

Example of read operation:

```
GET http://localhost:7001/oracle-fhir-server/fhir/DocumentReference/
499952/$binary-access-read?path=DocumentReference.content.attachment
```

> **Note:**
>
> The $binary-access-read operation works on any resource that contains base64 data.

Example:

```
/DocumentReference/[id]/$binary-access-read?
path=DocumentReference.content.attachment
GET /Binary/[id]/$binary-access-read?path=Binary.data
GET /Media/[id]/$binary-access-read?path=Media.content
```

> **Note:**
>
> If the data is stored as blob, it is required to switch your response type to "blob" in the client call.

12

Other Features

Oracle Healthcare Data Repository includes other features such as Repository Validation support, Remote Terminology Service Validation support, Patient :identifier Search Parameter support, and Lucene/Elasticsearch indexing.

- Repository Validation Support
  Repository Validation provides another way to validate against the core FHIR specification and the resource against it.
- Remote Terminology Service Validation Support
  This service validates codes using a remote FHIR-based terminology server.
- Patient :identifier Search Parameter Support
  HDR FHIR supports patient identifier search for all resources.
- Lucene/Elasticsearch Indexing
  The HDR-FHIR JPA Server supports optional indexing via Hibernate Search when configured to use Lucene or Elasticsearch to support the _content, or _text search parameters and also the extended Lucene string search indexing supports the default search, as well as :contains, :exact, and :text modifiers.

Repository Validation Support

Repository Validation provides another way to validate against the core FHIR specification and the resource against it.

The default implementation uses the built-in official FHIR definitions to validate the resource against it and, in many cases, this is good enough. However, if you have needs beyond validating against the core FHIR specification, you may want to use another validation method called Repository Validation.

When using a HDR FHIR JPA server as a FHIR Repository, it is often desirable to enforce specific rules about which specific FHIR profiles can or must be used.

For example, if an organization created a FHIR Repository for the purposes of hosting and serving US Core Data, that organization might want to enforce rules that data being stored in the repository must actually declare conformance to US Core Profiles such as the US Core Patient Profile and US Core Observation Profile.

In this situation, it would also be useful to require that any data being stored in the repository be validated, and rejected if it is not valid. The Repository Validating method can be used for this purpose. The HDR-FHIR server will be able to configure to validate resources based on external profiles or implementation guides.

- Configuration
  The required configuration properties are added to the hdr_fhir.yaml file.

Configuration

The required configuration properties are added to the hdr_fhir.yaml file.

load_ig_on_server_startup: false

If this property is set to true, the HDR-FHIR server dynamically loads configured IG packages during FHIR server startup. By default this property is set to false.

```
implementationguides:
   us_core:
        name: hl7.fhir.us.core
        version: 6.0.0
        url: http://hl7.org/fhir/us/core/STU6/package.tgz
```

To load a different IG, change the url, name, and version to point to the specific IG package location, add one more section like above that includes name, version and url.

install_transitive_ig_dependencies: false

This property allows loading of dependencies required by the IG. Usually, this value is set to false as most of the IGs depends on hl7 fhir core which is implied and doesn't require loading. Set this value to false.

validation_repository_enabled: false

If this property set to true, the incoming FHIR resource is validated against a configured profile. Set this value to false to disable repository validation.

> **Note:**
>
> Make sure that the incoming message contains meta tag with profile information to be validated.

Sample Message:

```
{
  "resourceType" : "Patient",
  "id" : "patient-example-female",
     "meta": {
      "profile": [
        "http://hl7.org/fhir/us/core/StructureDefinition/us-core-
patient"
      ]
   },
  "text" : {
    "status" : "generated",
    "div" : "<div xmlns=\"http://www.w3.org/1999/xhtml\"><p><b>Martha
DeLarosa </b> female, DoB: 1992-05-01 ( id: 574687583)</p></div>"
  },
```

```
  "identifier" : [
    {
      "system" : "urn:oid:2.16.840.1.113883.2.4.6.3",
      "value" : "574687583"
    }
  ],
  "active" : true,
  "name" : [
    {
      "family" : "DeLarosa",
      "given" : [
        "Martha"
      ]
    }
  ],
  "telecom" : [
    {
      "system" : "phone",
      "value" : "+31788700800",
      "use" : "home"
    }
  ],
  "gender" : "female",
  "birthDate" : "1992-05-01",
  "address" : [
    {
      "line" : [
        "Laan Van Europa 1600"
      ],
      "city" : "Dordrecht",
      "postalCode" : "3317 DB",
      "country" : "NL"
    }
  ],
  "contact" : [
    {
      "relationship" : [
        {
"coding": [
            {
              "system": "http://terminology.hl7.org/CodeSystem/v2-0131",
              "code": "N"
            }
          ]
        }
      ],
      "name" : {
        "family" : "Mum",
        "given" : [
          "Martha"
        ]
      },
      "telecom" : [
        {
          "system" : "phone",
```

```
          "value" : "+33-555-20036",
          "use" : "home"
        }
      ],
      "address" : {
        "line" : [
          "Promenade des Anglais 111"
        ],
        "city" : "Lyon",
        "postalCode" : "69001",
        "country" : "FR"
      }
    }
  ]
}
```

Remote Terminology Service Validation Support

This service validates codes using a remote FHIR-based terminology server.

The remote terminology service must be configured on HDR FHIR server that is being used for validation support. When enabled, profile StructureDefinition resources will still be fetched directly from the validation support repository, but any validation request operations (e.g. $lookup, $validate-code, $expand, $translate) will be forwarded to the remote terminology server.

- Configuration
  The required configuration properties are added to the hdr_fhir.yaml file.

Configuration

The required configuration properties are added to the hdr_fhir.yaml file.

remote_terminology_service_enabled: false

Set this property to true to enable validate codes using the configured FHIR-based remote terminology server. By default it is false.

remote_terminology_server_base_url=http://hapi.fhir.org/baseR4

Specify FHIR-based remote terminology server url to validate codes against this server.

> **Note:**
>
> If remote terminology validation is set to true, make sure that request validation is also enabled (requests_enabled property).

Patient :identifier Search Parameter Support

HDR FHIR supports patient identifier search for all resources.

The FHIR standard defines a modifier :identifier that can be used to limit the reference search parameter. In this manner, the system allows for searching by identifier rather than literal reference.

Example:

```
GET [base]/Consent?patient:identifier=http://
ns.electronichealth.net.au/id/hi/ihi/1.0|8003608833630130
```

The example is a search for all Consent resources that reference a patient by a particular patient identifier. When the :identifier modifier is used, the search value works as a token search.

In this case, patient resource will not be created before, and it will be attached while persisting the resource.

Sample Message:

```
{
  "resourceType": "Consent",
  "id": "consent-example-basic1255",
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/
xhtml\">\n\t\t\t<p>\n\tAuthorize Normal access for Treatment\n\t\t\t</
p>\n\t\t\t<p>\n    Patient &quot;P. van de Heuvel&quot; wishes to have all
of the PHI collected at the Good Health Psychiatric Hospital \n   available
for normal treatment use.\n\t\t\t</p>\n\t\t</div>"
  },
  "status": "active",
  "scope": {
    "coding": [
      {
        "system": "http://terminology.hl7.org/CodeSystem/consentscope",
        "code": "patient-privacy"
      }
    ]
  },
  "category": [
    {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "59284-0"
        }
      ]
    }
  ],
  "patient": [{
        "identifier": {
```

```
            "system": "http://test.com",
            "value": "12345"
        }
  }],
  "dateTime": "2016-05-11",
  "organization": [
    {
      "reference": "Organization/rrrrf001"
    }
  ],
  "sourceAttachment": {
    "title": "The terms of the consent in lawyer speak."
  },
  "policyRule": {
    "coding": [
      {
        "system": "http://terminology.hl7.org/CodeSystem/v3-ActCode",
        "code": "OPTIN"
      }
    ]
  },
  "provision": {
    "period": {
      "start": "1964-01-01",
      "end": "2016-01-01"
    }
  }
}
```

When the above message is persisted, indexed patient references will generate an index in the HFJ_SPIDX_TOKEN table for the patient identifier.

Now you can search Consent resources that reference a patient by a patient identifier using the GET operation like below.

```
GET [base]/Consent?patient:identifier=http://test.com|12345
```

Lucene/Elasticsearch Indexing

The HDR-FHIR JPA Server supports optional indexing via Hibernate Search when configured to use Lucene or Elasticsearch to support the _content, or _text search parameters and also the extended Lucene string search indexing supports the default search, as well as :contains, :exact, and :text modifiers.

- Configuration
  The settings below will be enabled to support lucene or elastic search in the hdr_fhir.yaml file.
- String search
  The Extended Lucene string search indexing supports the default search, as well as :contains, :exact, and :text modifiers.

Configuration

The settings below will be enabled to support lucene or elastic search in the hdr_fhir.yaml file.

hibernate.search.enabled: true

For Lucene:

```
 ### lucene parameters
      hibernate.search.backend.type: lucene
      hibernate.search.backend.analysis.configurer:
ca.uhn.fhir.jpa.search.HapiHSearchAnalysisConfigurers$HapiLuceneAnalysisConfi
gurer
      hibernate.search.backend.directory.type: local-filesystem
      hibernate.search.backend.directory.root: target/lucenefiles
      hibernate.search.backend.lucene_version: lucene_current
```

For Elastic Search:

```
     hibernate.search.backend.type: elasticsearch
     hibernate.search.backend.analysis.configurer:
ca.uhn.fhir.jpa.search.HapiHSearchAnalysisConfigurers$HapiElasticAnalysisConf
igurer

elasticsearch:
  debug:
    pretty_print_json_log: false
    refresh_after_write: false
  enabled: false
  required_index_status: YELLOW
  rest_url: 'localhost:9200'
  protocol: 'http'
  schema_management_strategy: CREATE
  username: SomeUsername
  password: SomePassword
```

> **Note:**
>
> To work with elastic search, the required elastic search client jar files need to be placed under the WebLogic domain classpath (Example: <FHIR_DOMAIN>/lib) **or add those jar files to HDR FHIR application war file under <HDR_FHIR_WAR>\WEB-INF\lib folder directly**. These jar files are not bundled with the HDR FHIR application due to Oracle's commercial license procedure/ restrictions.

String search

The Extended Lucene string search indexing supports the default search, as well as :contains, :exact, and :text modifiers.

The default (unmodified) string search matches by prefix, insensitive to case or accents.

:exact matches the entire string, matching case and accents.

:contains match any substring of the text, ignoring case and accents.

:text provides a rich search syntax.

For more information, see:

https://www.hl7.org/fhir/search.html#string

Configuration

```
###  !!Extended Lucene/Elasticsearch Indexing is still a experimental
feature, expect some features (e.g. _total=accurate) to not work as
expected!!
    ###  more information here: https://hapifhir.io/hapi-fhir/docs/
server_jpa/elastic.html
    advanced_lucene_indexing: false
```

13
Partitioning

Oracle Healthcare Data Repository FHIR 8.1.4 introduced a new feature called Partitioning. Partitioning allows each resource on the server to be placed in a partition grouping a set of resources together.

Partitioning Outcomes

Partitioning is designed so that it can be used to achieve different outcomes.

Multitenancy

Partitioning could be used to achieve **multitenancy**, where there are multiple logically separate resources on the server. Traditionally this kind of setup is called **tenant wise**, and each of these tenants should not be able to access or modify data belonging to another tenant.

Separate Data

Partitioning could also be used to logically **separate data coming from distinct sources** within an organization. For example, patient records might be placed in one partition, lab data sourced from a lab system might be placed in a second partition and patient surveys from a survey app might be placed in another. In this situation, data does not need to be completely segregated (Lab Observation records may have references to Patient records in the patient partition) but these partitions might be used to support security groups, retention policies, etc.

Partitioning in HDR FHIR JPA means that every resource has a partition identity. This identity consists of the following attributes:

- **Partition Name**: This is a short textual identifier for the partition that the resource belongs to. This might be a customer ID, a description of the type of data in the partition, or something else.
- **Partition ID**: This is an integer ID that corresponds 1:1 with the partition Name. It is used in the database as the partition identifier.
- **Partition Date**: This is an additional partition discriminator that can be used to implement partitioning strategies using a date axis.

At the database level, partitioning involves the use of two dedicated columns to many tables within the HDR_FHIR Schema

PARTITION_ID – This is an integer indicating the specific partition that a given resource is placed in. This column can also be NULL, meaning that the given resource is in the Default Partition.

PARTITION_DATE – This is a date/time column that can be assigned an arbitrary value depending on your use case.

When partitioning is used, these two columns will be populated with the same value for a given resource on all resource-specific tables (this includes HFJ_RESOURCE and all tables that have a foreign key relationship to it including HFJ_RES_VER, HFJ_RESLINK, HFJ_SPIDX_*, extended tables like OHF_FHIR_*etc.)

When a new resource is created, the partition ID and date is assigned to the resource.

When a resource is updated, the partition ID and date from the previous version is used.

Partition Operations

Several operations can be used to manage partitions.

Before a partition can be used, it must be registered.

- Creating a Partition
  The $partition-management-create-partition operation can be used to create a new partition.
- Updating a Partition
  The $partition-management-update-partition operation can be used to update an existing partition.
- Deleting a Partition
  The $partition-management-delete-partition operation can be used to delete an existing partition.
- Reading a Partition
  The $partition-management-read-partition operation can be used to read an existing partition.
- Listing all Partitions
  The $partition-management-list-partitions operation can be used to list all existing partitions.

Creating a Partition

The $partition-management-create-partition operation can be used to create a new partition.

An HTTP POST to the following URL can be used to invoke this operation. Notice that we use the DEFAULT partition, as it always exists by default.

Example:

```
http://localhost:9001/oracle-fhir-server/fhir/DEFAULT/$partition-
management-create-partition
```

The following request body could be used:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "id",
    "valueInteger": 123
  }, {
    "name": "name",
    "valueCode": "PARTITION-123"
  }, {
    "name": "description",
    "valueString": "a description"
  } ]
}
```

Updating a Partition

The $partition-management-update-partition operation can be used to update an existing partition.

An HTTP POST to the following URL can be used to invoke this operation.

Example:

```
http://localhost:9001/oracle-fhir-server/fhir/DEFAULT/$partition-management-update-partition
```

The following request body could be used:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "id",
    "valueInteger": 123
  }, {
    "name": "name",
    "valueCode": "PARTITION-123"
  }, {
    "name": "description",
    "valueString": "a description"
  } ]
}
```

Deleting a Partition

The $partition-management-delete-partition operation can be used to delete an existing partition.

An HTTP POST to the following URL can be used to invoke this operation.

Example:

```
http://localhost:9001/oracle-fhir-server/fhir/DEFAULT/$partition-
management-delete-partition
```

The following request body could be used:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "id",
    "valueInteger": 123
  } ]
}
```

Reading a Partition

The $partition-management-read-partition operation can be used to read an existing partition.

An HTTP POST to the following URL can be used to invoke this operation.

```
http://localhost:9001/oracle-fhir-server/fhir/DEFAULT/$partition-
management-read-partition
```

The following request body could be used:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "id",
    "valueInteger": 123
  } ]
}
```

Listing all Partitions

The $partition-management-list-partitions operation can be used to list all existing partitions.

An HTTP POST to the following URL can be used to invoke this operation.

Example:

```
http://localhost:9001/oracle-fhir-server/fhir/DEFAULT/$partition-
management-list-partitions
```

This operation returns a Parameters resource that looks like the following:

```
{
    "resourceType": "Parameters",
    "parameter": [ {
       "name": "partition",
       "part": [ {
          "name": "id",
          "valueInteger": 1
        }, {
          "name": "name",
          "valueCode": "PARTITION-1"
        }, {
          "name": "description",
          "valueString": "a description1"
        } ]
      }, {
        "name": "partition",
        "part": [ {
          "name": "id",
          "valueInteger": 2
        }, {
          "name": "name",
          "valueCode": "PARTITION-2"
        }, {
          "name": "description",
          "valueString": "a description2"
        } ]
    } ]
}
```

Enabling Partitioning

To enable partitioning on the HDR FHIR server, uncomment and enable the properties in the hdr_fhir.yaml file as shown below.

partitioning:

allow_references_across_partitions: true

When this flag is not set (as is the default), when a search requests a specific partition, an additional SQL WHERE predicate is added to the query to explicitly request the given partition ID. When this flag is set, this additional WHERE predicate is not necessary since the partition is factored into the hash value being searched on. Setting this flag avoids the need to manually adjust indexes against the HFJ_SPIDX tables.

partitioning_include_in_search_hashes: true

This setting controls whether resources in one partition should be allowed to create references to resources in other partitions.

14

Terminology

HDR FHIR supports a terminology feature to use and manage code.

The general pattern for representing coded values in resources uses the following four elements:

- **system** - A URI that identifies the system
- **version** - A string value representing a version of the original code system
- **code** - A string value that identifies a concept as defined by the code system
- **display** - A description of the concept as defined by the code system

Example:

```
{
  "system" : "http://loinc.org",
  "version" : "2.62",
  "code" : "55423-8",
  "display" : "Number of steps in unspecified time Pedometer"
}
```


Uploading CodeSystems

In most cases, it is useful to upload your own CodeSystems for use in HDR FHIR Repository. This might mean uploading external Terminologies such as LOINC/SNOMED or loading custom terminologies such as local CodeSystems defined for organizational specific use.

Uploading External Terminologies

HDR FHIR has the ability to upload several standard terminology code systems like LOINC/SNOMED using their native distribution formats. This is done using the **$upload-external-code-system** operation on the CodeSystem Resource and can be invoked using the HDR FHIR command line utility. Refer to the FHIR Command-Line Utility chapter in this guide.

Table 14-1 External CodeSystem Formats

CodeSystem	URL	File Format
LOINC	http://loinc.org	LOINC Complete Download ZIP
SNOMED CT	http://snomed.info/sct	RF2 Distribution in ZIP

Uploading Custom Terminologies Using CSV Files

HDR FHIR has the ability to upload custom local CodeSystem using the supported CSV file formats. This is done using the **$upload-external-code-system** operation on the CodeSystem Resource and can be invoked directly using the REST operation.

To upload terminology via the REST endpoint, perform an HTTP POST to the following URL:

```
http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/CodeSystem/$upload-external-code-system
```

The request body for this operation is a Parameters resource with the following parts:

Table 14-2 Parameters

Parameter	Description
system	This parameter should have a uri value containing the CodeSystem URI.
file	This parameter should have an attachment value containing the value of the file (encoded in Base64) in Attachment.data and the filename in Attachment.url.
contentType	Specifies the payload type (should be text/csv, application/zip etc.).
data	Contains the raw Base64 encoded payload.
url	Used by the upload processor to determine the type of file being uploaded (e.g. concepts.csv vs hierarchy.csv). See the CSV input Files below for descriptions of the files to upload.

Example: CSV Files Content

```
CODE,DISPLAY

CHEM,Chemistry
HB,Hemoglobin
NEUT,Neutrophils

MICRO,Microbiology
C&S,Culture and Sensitivity
```

Example: Request Body

```
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "system",
      "valueUri": "http://example.com/csv"
    },
    {
      "name": "file",
      "valueAttachment": {
        "contentType": "text/csv",
        "data":
"Q09ERSxESVNQTEFZCgpDSEVNLENoZW1pc3RyeQpIQixIZW1vZ2xvYmluCk5FVVQsTmV1dHJvcGhp
bHMKCk1JQ1JPLE1pY3JvYmlvbG9neQpDJlMsQ3VsdHVyZSBhbmQgU2Vuc2l0aXZpdHkK",
        "url": "file:/concepts.csv"
      }
    },
    {
      "name": "file",
      "valueAttachment": {
        "contentType": "text/csv",
        "data":
"UEFSRU5ULENISUxECgpDSEVNLEhCCkNIRU0sTkVVVAoKTUlDUk8sQyZTCg==",
        "url": "file:/hierarchy.csv"
      }
    }
  ]
}
```

Applying Deltas to CodeSystems

A pair of FHIR operations can be used to add or remove codes from the CodeSystems. These operations directly add or remove the codes from the specified CodeSystem.

Delta Add Operation: $apply-codesystem-delta-add

The $apply-codesystem-delta-add operation can be used to add concepts to an existing Codeystem.

The following example shows an invocation of the Delta Add operation. This payload should be a POST to the following URL:

```
http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/CodeSystem/$apply-
codesystem-delta-add
```

The payload is shown below:

```
{
  "resourceType": "Parameters",
```

```
  "parameter": [
    {
      "name": "system",
      "valueUri": "http://example.com/csv"
    },
    {
      "name": "file",
      "valueAttachment": {
        "contentType": "text/csv",
        "data":
"Q09ERSxESVNQTEFZCgpDSEVNVFJZLENoZW1pc3RyeQpIQjEsSGVtb2dsb2Jpbg==",
        "url": "file:/concepts.csv"
      }
    }
  ]
}
```

Delta Remove Operation: $apply-codesystem-delta-remove

The $apply-codesystem-delta-remove operation can be used to remove concepts from an existing CodeSystem.

The following example shows an invocation of the Delta Remove operation. This payload should be a POST to the following URL:

```
http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/CodeSystem/$apply-
codesystem-delta-remove
```

The payload is shown below:

```
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "system",
      "valueUri": "http://example.com/csv"
    },
    {
      "name": "file",
      "valueAttachment": {
        "contentType": "text/csv",
        "data":
"Q09ERSxESVNQTEFZCgpDSEVNVFJZLENoZW1pc3RyeQpIQjEsSGVtb2dsb2Jpbg==",
        "url": "file:/concepts.csv"
      }
    }
  ]
}
```

ValueSet

The FHIR Specification defines two resource types that are used as a part of defining and using codes.

- The **CodeSystem** resource defines a collection of codes.
- The **ValueSet** resource creates a collection of codes drawn from one or more CodeSystems.

ValueSets are defined by a collection of rules, it means the composition. These rules can be simple rules (e.g. include codes P, Q, and R) or much more complex rules (ex: include any codes that are a child of code P, etc).

Pre-Calculation for ValueSet Expansion

When a ValueSet is uploaded into HDR FHIR Repository, a scheduler job will be triggered in the background to calculate the expansion of the valueset and store it in a dedicated set of database tables. When you perform a ValueSet expansion, an extension will be added to the **ValueSet.meta** element. This extension shows the status of the pre-calculation.

Pre-Calculation Status

To test whether a ValueSet has been pre-calculated, simply request the expansion using the $expand operation. For example, the following request can be used to request the expansion of the my_custom_value_set ValueSet:

```
GET ValueSet/$expand?url= http://example.com/my_custom_value_set
```

If this ValueSet has not been precalculated, the system will send a response similar to the following:

```
{
  "resourceType": "ValueSet",
  "id": "200007",
  "meta": {
    "extension": [ {
      "url": "http://hapifhir.io/fhir/StructureDefinition/valueset-expansion-
message",
      "valueString": "ValueSet \"ValueSet.url[http://example.com/
t_custom_value_set]\" has not yet been pre-expanded. Performing in-memory
expansion without parameters. Current status: NOT_EXPANDED | The ValueSet is
waiting to be picked up and pre-expanded by a scheduled task."
    } ],
    "versionId": "1"
  },
  "url": "http://example.com/my_custom_value_set",
  "status": "active",
[ ... Remaining Fields Not Shown ... ]
}
```

After the pre-calculation has completed, you will see a response similar to the following:

```
{
  "resourceType": "ValueSet",
  "id": "200007",
```

```
  "meta": {
    "extension": [ {
      "url": "http://hapifhir.io/fhir/StructureDefinition/valueset-
expansion-message",
      "valueString": "ValueSet was expanded using an expansion that
was pre-calculated at 2023-08-16T20:14:35.055+05:30 (00:24:06 ago)"
    } ],
    "versionId": "1"
  },
  "url": "http://example.com/t_custom_value_set",
  "status": "active",
[ ... Remaining Fields Not Shown ... ]
}
```

Invalidating Pre-Calculated Expansion

It is also possible to manually request that the existing pre-calculated valueset expansion be invalidated and a new one can be calculated. This is useful in cases where the underlying CodeSystem has been changed.

To invalidate an existing Pre-Calculated valueset expansion, use the $invalidate-expansion operation by using a POST against the ValueSet resource ID such as the following:

```
POST ValueSet/200007/$invalidate-expansion
```

The response will be similar to the following:

```
{
  "resourceType": "Parameters",
  "parameter": [ {
    "name": "message",
    "valueString": "ValueSet with URL \"http://example.com/
my_custom_value_set\" precaluclated expansion with 4 concept(s) has
been invalidated"
  } ]
}
```

- Expanding Hierarchical CodeSystems and ValueSets
  Many CodeSystem resources define concepts in a hierarchy, ex: parent code and child codes.
- Requesting A ValueSet Expansion
  ValueSet expansion can be invoked using the HTTP GET on the following URL:
- Requesting a ValueSet Hierarchical Expansion
  If you would like to request parent-child relationships to be reflected in the response, you can add the includeHierarchy parameter in your request.

Expanding Hierarchical CodeSystems and ValueSets

Many CodeSystem resources define concepts in a hierarchy, ex: parent code and child codes.

The hierarchy in a CodeSystem often indicates an "is-a" relationship between the parent code and the child codes. This is not always the case; the hierarchy can imply different kinds of relationships depending on the specific system.

ValueSets can be used to retrieve all of the codes that are a child of a specific code in a CodeSystem.

For example, suppose you have the following CodeSystem and there are 3 codes at the root level (P , Q and R) and each of these codes have children, some of which have further children.

```
{
  "resourceType": "CodeSystem",
  "url": "http://example.com/my_custom_codeSystem",
  "content": "complete",
  "concept": [ {
    "code": "P",
    "display": "Code P",
    "concept": [ {
      "code": "PP",
      "display": "Code PP",
      "concept": [ {
        "code": "PPP",
        "display": "Code PPP"
      } ]
    }, {
      "code": "PQ",
      "display": "Code PQ"
    } ]
  }, {
    "code": "Q",
    "display": "Code Q",
    "concept": [ {
      "code": "QP",
      "display": "Code QP"
    }, {
      "code": "QQ",
      "display": "Code QQ"
    } ]
  } , {
    "code": "R",
    "display": "Code R",
    "concept": [ {
      "code": "RP",
      "display": "Code RP"
    }, {
      "code": "RQ",
      "display": "Code RQ"
    } , {
      "code": "RR",
      "display": "Code RR"
    }]
  } ]
}
```

The hierarchy for the codes above can be visualized as follows:

```
|-- P
|   |-- PP
|   |   \-- PPP
|   \-- PQ
\-- Q
|   |-- QP
|   \-- QQ
\-- R
    |-- RP
    \-- RQ
    \-- RR
```

To create a ValueSet containing all of the children of a specific code in this CodeSystem, a ValueSet with a filter can be defined:

```
{
   "resourceType": "ValueSet",
   "url": "http://example.com/my_custom_value_set",
   "status": "active",
   "compose": {
      "include": [ {
         "system": "http://example.com/my_custom_codeSystem",
         "filter": [ {
            "property": "concept",
            "op": "is-a",
            "value": "P"
         } ]
      } ]
   }
}
```

> **Note:**
>
> Note that the "is-a" filter will exclude the concept itself.

The following example combines an "is-a" filter with a simple explicit code inclusion in order to include the code "P" as well as all of its descendants.

```
{
   "resourceType": "ValueSet",
   "url": "http://example.com/my_custom_value_set",
   "status": "active",
   "compose": {
      "include": [ {
         "system": "http://example.com/my_custom_codeSystem",
         "filter": [ {
            "property": "concept",
            "op": "is-a",
            "value": "P"
         } ]
```

```
      }, {
         "system": "http://example.com/my_custom_codeSystem",
         "concept": [ {
           "code": "P"
         } ]
      } ]
   }
}
```

Requesting A ValueSet Expansion

ValueSet expansion can be invoked using the HTTP GET on the following URL:

```
http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/ValueSet/$expand?
url=my_custom_value_set
```

This will return the following response. Note that the hierarchy is not included in the response.

```
{
  "resourceType": "ValueSet",
  "id": "200007",
  "meta": {
    "extension": [ {
      "url": "http://hapifhir.io/fhir/StructureDefinition/valueset-expansion-
message",
      "valueString": "ValueSet was expanded using an expansion that was pre-
calculated at 2023-08-16T20:14:35.055+05:30 (00:28:24 ago)"
    } ],
    "versionId": "1"
  },
  "url": "http://example.com/my_custom_value_set",
  "status": "active",
  "compose": {
    "include": [ {
      "system": "http://example.com/my_custom_codeSystem",
      "filter": [ {
        "property": "concept",
        "op": "is-a",
        "value": "P"
      } ]
    }, {
      "system": "http://example.com/my_custom_codeSystem",
      "concept": [ {
        "code": "P"
      } ]
    } ]
  },
  "expansion": {
    "identifier": "d802bee9-5447-4ecd-92f9-062b704ca566",
    "timestamp": "2023-08-16T20:42:59+05:30",
```

```
    "total": 4,
    "offset": 0,
    "parameter": [ {
      "name": "offset",
      "valueInteger": 0
    }, {
      "name": "count",
      "valueInteger": 1000
    } ],
    "contains": [ {
      "system": "http://example.com/my_custom_codeSystem",
      "code": "PP",
      "display": "Code PP"
    }, {
      "system": "http://example.com/my_custom_codeSystem",
      "code": "PPP",
      "display": "Code PPP"
    }, {
      "system": "http://example.com/my_custom_codeSystem",
      "code": "PQ",
      "display": "Code PQ"
    }, {
      "system": "http://example.com/my_custom_codeSystem",
      "code": "P",
      "display": "Code P"
    } ]
  }
}
```

Requesting a ValueSet Hierarchical Expansion

If you would like to request parent-child relationships to be reflected in the response, you can add the includeHierarchy parameter in your request.

Example:

```
http://<HOST_NAME>:<PORT>/oracle-fhir-server/fhir/ValueSet/$expand?
url=my_custom_value_set &includeHierarchy=true
```

This will return the following response:

```
{
  "resourceType": "ValueSet",
  "id": "200007",
  "meta": {
    "extension": [ {
      "url": "http://hapifhir.io/fhir/StructureDefinition/valueset-
expansion-message",
      "valueString": "ValueSet was expanded using an expansion that
was pre-calculated at 2023-08-16T20:14:35.055+05:30 (00:29:44 ago)"
    } ],
    "versionId": "1"
```

```
  },
  "url": "http://example.com/my_custom_value_set",
  "status": "active",
  "compose": {
    "include": [ {
      "system": "http://example.com/my_custom_codeSystem",
      "filter": [ {
        "property": "concept",
        "op": "is-a",
        "value": "P"
      } ]
    }, {
      "system": "http://example.com/my_custom_codeSystem",
      "concept": [ {
        "code": "P"
      } ]
    } ]
  },
  "expansion": {
    "identifier": "415a2a3c-02d5-463d-a810-e9d907146234",
    "timestamp": "2023-08-16T20:44:18+05:30",
    "total": 4,
    "offset": 0,
    "parameter": [ {
      "name": "offset",
      "valueInteger": 0
    }, {
      "name": "count",
      "valueInteger": 1000
    } ],
    "contains": [ {
      "system": "http://example.com/my_custom_codeSystem",
      "code": "P",
      "display": "Code P",
      "contains": [ {
        "system": "http://example.com/my_custom_codeSystem",
        "code": "PP",
        "display": "Code PP",
        "contains": [ {
          "system": "http://example.com/my_custom_codeSystem",
          "code": "PPP",
          "display": "Code PPP"
        } ]
      }, {
        "system": "http://example.com/my_custom_codeSystem",
        "code": "PQ",
        "display": "Code PQ"
      } ]
    } ]
  }
}
```