National Resource Centre
for EHR Standards
India

Implementation Guide for Adoption of FHIR in ABDM and NHCX

Created by:
National Resource Centre for EHR Standards,
Centre for Development of Advanced Computing (C-DAC), Pune, India

Published: September 2024

National Resource Centre for EHR Standards India

TABLE OF CONTENTS

Introduction ____ *4*

Brief Introduction to FHIR ____ *4*

Scope ____ *4*

Key Concepts of FHIR ____ *5*

- **Resource** ____ 5
- **Bundle** ____ 5
- **Profile and Extension** ____ 5
- **Terminology** ____ 5
- **Validation** ____ 5
- **Interoperability** ____ 5
- **Modularity and Scalability** ____ 5
- **Exchange Paradigm** ____ 5

Resource ____ *6*

Key Part of Resource ____ 7

Data types ____ *7*

Key datatypes ____ 7

FHIR Paradigm ____ *8*

Bundle ____ *9*

Use of Bundle with 'type' as 'Document' ____ 9

How it works ____ 9

Example Use Case ____ 9

Use of Bundle with 'type' as 'Collection' ____ 9

Example Use Case ____ 9

Key Elements of a FHIR Bundle resource ____ 10

FHIR Implementation Guide ____ *10*

FHIR Implementation Guide for Ayushman Bharat Digital Mission ____ 11

Reading FHIR Profiles ____ 11

1. Statistics/References ____ 11
2. Differential View ____ 11
3. Mandatory Element ____ 11
4. Must Support ____ 11

FHIR Profiles for ABDM ____ 12

FHIR Profiles for NHCX ____ 12

Implementing and Validating FHIR using Java ____ *14*

HAPI FHIR Library ____ 14

Key Features of HAPI FHIR Library 14
- Complete FHIR Support 14
- Built-in Validators 14
- Serialization & Parsing 14
- Extensive Resource Coverage 14

HAPI FHIR Dependencies 14
- hapi-fhir-structures-r4 14
- hapi-fhir-validation 14
- hapi-fhir-validation-resources-r4 14

Creating FHIR Resources Programmatically 15
Prerequisites 15
Setting Up the Development Environment 15
Creating Resource 16
Patient Resource in Java and JSON Representation 17
Creating FHIR Bundles Programmatically in ABDM and NHCX 17
Creating a Document Bundle for ABDM 17
Creating a Collection Bundle for NHCX 18
Common Pitfalls in Creating FHIR Bundles 19

Validating FHIR Resources 19
Validation Tools 19
Validation Aspects 20
Validation Process 20

Implementation Reference ***23***

Annexure: FHIR Bundles utilized by each NHCX API based on specific Use Cases ***24***

INTRODUCTION

Intending to build a national digital health ecosystem that provides diverse data and infrastructure services by leveraging open, interoperable systems, adopting FHIR in Ayushman Bharat Digital Mission (ABDM) has been a cornerstone of ABDM's digital healthcare initiatives.

FHIR has been identified as a data structure standard defining health information structures that represent different health records to achieve continuity of care along with standard structures defined for claim processing. The adoption of FHIR in ABDM and NHCX results in enhanced healthcare delivery and fosters innovation in health services. It streamlines data exchange across different health systems, leading to better interoperability and more accurate patient records. It also facilitates real-time access to health information, which can support more timely and informed decision-making by healthcare providers. FHIR's standardized protocols streamline claims processing and administrative workflows leading to faster claim processing and reduced errors.

BRIEF INTRODUCTION TO FHIR

Fast Healthcare Interoperability Resources (FHIR), developed by Health Level Seven (HL7), is a modern standard designed to streamline the electronic exchange of healthcare information. By using widely adopted web standards such as RESTful APIs, XML, and JSON, FHIR provides a flexible framework that simplifies healthcare data integration and ensures seamless interoperability between different healthcare systems.

One of the primary goals of FHIR is to simplify healthcare data exchange by reducing technical barriers. Using familiar internet-based technologies, it allows real-time sharing of healthcare information through discrete “Resources” such as patient data, medications, and observations. These modular resources can be easily combined and extended to suit various healthcare needs, supporting both data sharing and the development of new applications. FHIR’s standardized framework promotes interoperability across systems, enabling faster innovation and more accurate clinical decision-making, ultimately improving patient outcomes and healthcare delivery.

Refer: Index - FHIR v4.0.1 (hl7.org)

SCOPE

This document provides an overview of FHIR and its usage within the ABDM. It serves as a reference for understanding FHIR and its adoption in ABDM and NHCX, outlining how to create and validate FHIR resources using available libraries. While the approach to implementation may vary based on application requirements, resources, and scope, this document aims to assist healthcare

stakeholders in integrating interoperable, secure, and standardized healthcare data exchange solutions effectively within the healthcare ecosystem.

KEY CONCEPTS OF FHIR

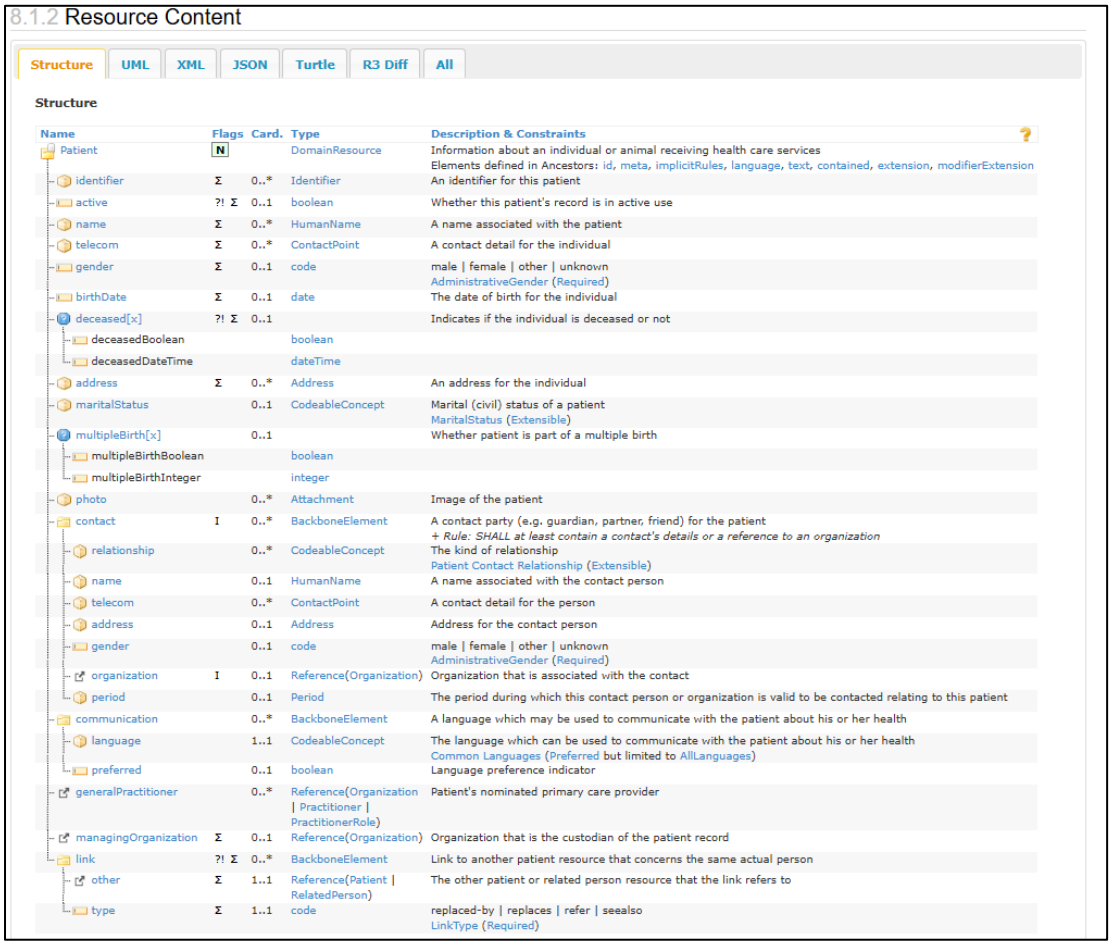
- **Resource**: FHIR is resource-centric, meaning all healthcare-related data is represented as a set of modular components called "resource." These resources are the building blocks of FHIR and can represent anything from a patient, medication, or observation, to complex care plans. Resources can be combined or extended to suit specific use cases, making FHIR adaptable to various needs.
- **Bundle**: FHIR supports the use of Bundle, which is collection of resources that can be sent or retrieved in a single transaction.
- **Profile and Extension**: FHIR resources can be customized using profiles to meet specific implementation needs. Extensions allow adding new data elements or modifying existing ones without altering the core resource structure.
- **Terminology**: FHIR integrates with standardized terminologies such as SNOMED CT, LOINC, and ICD, allowing consistent use of codes and classifications for clinical concepts.
- **Validation**: FHIR provides mechanisms to validate resources against profiles, ensuring that the data conforms to specific rules and constraints, improving data quality and consistency across systems.
- **Interoperability**: FHIR is designed to ensure systems can communicate seamlessly by providing a standardized data format. It promotes interoperability between different healthcare systems, enabling them to share data efficiently.
- **Modularity and Scalability**: FHIR resources can be used individually or in combination, which allows incremental adoption. This flexibility ensures that FHIR can scale from simple to complex healthcare systems.
- **Exchange Paradigm**: The FHIR exchange paradigm refers to the methods used for sharing healthcare data between systems, leveraging modern web standards like RESTful APIs, messaging, and documents. It enables real-time, secured, and scalable exchange of modular data units called resources. This approach facilitates seamless interoperability between healthcare applications, improving data accessibility and patient care.

RESOURCE

FHIR resource is a fundamental component of this standard, representing a specific type of healthcare information in a structured and standardized way. Each resource represents a distinct type of healthcare data element, such as patient information, medications, or observations. These resources are structured entities with defined attributes and properties, which ensure that the data is captured consistently across different systems. Each resource also includes relationships with other resources, allowing them to interact and form a comprehensive representation of a healthcare process. These resources are modular and can be combined or extended to suit specific clinical workflows, enabling flexibility in how they are applied in various healthcare settings while maintaining standardization and interoperability across systems.

Some important resource categories include:

- Clinical Resources: Allergy, Problem, Procedure.
- Administrative Resources: Practitioner, CareTeam, Device, Organization.
- Financial Resources: Claim, Coverage, PaymentNotice.



8.1.2 Resource Content

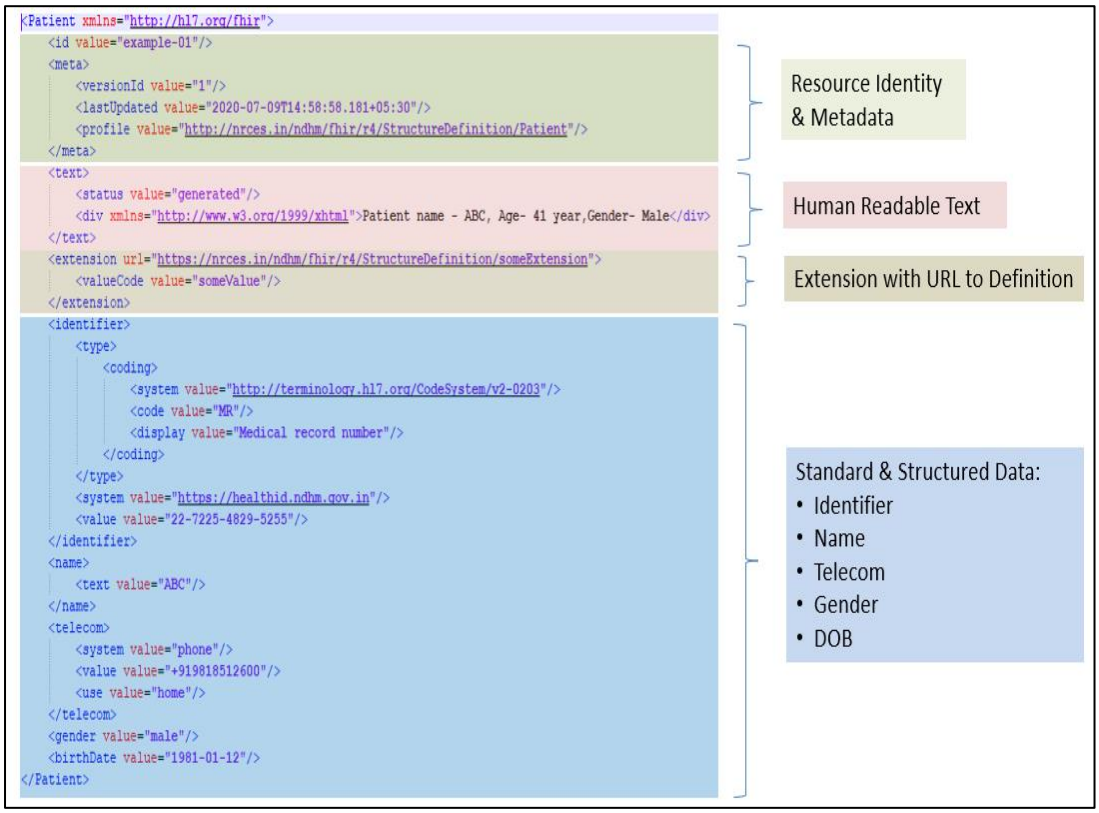
Structure | UML | XML | JSON | Turtle | R3 Diff | All

Structure

Name	Flags	Card.	Type	Description & Constraints
Patient	N		DomainResource	Information about an individual or animal receiving health care services Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
identifier	Σ	0..*	Identifier	An identifier for this patient
active	?! Σ	0..1	boolean	Whether this patient's record is in active use
name	Σ	0..*	HumanName	A name associated with the patient
telecom	Σ	0..*	ContactPoint	A contact detail for the individual
gender	Σ	0..1	code	male | female | other | unknown AdministrativeGender (Required)
birthDate	Σ	0..1	date	The date of birth for the individual
deceased[x]	?! Σ	0..1		Indicates if the individual is deceased or not
deceasedBoolean			boolean	
deceasedDateTime			dateTime	
address	Σ	0..*	Address	An address for the individual
maritalStatus		0..1	CodeableConcept	Marital (civil) status of a patient MaritalStatus (Extensible)
multipleBirth[x]		0..1		Whether patient is part of a multiple birth
multipleBirthBoolean			boolean	
multipleBirthInteger			integer	
photo		0..*	Attachment	Image of the patient
contact	I	0..*	BackboneElement	A contact party (e.g. guardian, partner, friend) for the patient *+ Rule: SHALL at least contain a contact's details or a reference to an organization*
relationship		0..*	CodeableConcept	The kind of relationship Patient Contact Relationship (Extensible)
name		0..1	HumanName	A name associated with the contact person
telecom		0..*	ContactPoint	A contact detail for the person
address		0..1	Address	Address for the contact person
gender		0..1	code	male | female | other | unknown AdministrativeGender (Required)
organization	I	0..1	Reference(Organization)	Organization that is associated with the contact
period		0..1	Period	The period during which this contact person or organization is valid to be contacted relating to this patient
communication		0..*	BackboneElement	A language which may be used to communicate with the patient about his or her health
language		1..1	CodeableConcept	The language which can be used to communicate with the patient about his or her health Common Languages (Preferred but limited to AllLanguages)
preferred		0..1	boolean	Language preference indicator
generalPractitioner		0..*	Reference(Organization | Practitioner | PractitionerRole)	Patient's nominated primary care provider
managingOrganization	Σ	0..1	Reference(Organization)	Organization that is the custodian of the patient record
link	?! Σ	0..*	BackboneElement	Link to another patient resource that concerns the same actual person
other	Σ	1..1	Reference(Patient | RelatedPerson)	The other patient or related person resource that the link refers to
type	Σ	1..1	code	replaced-by | replaces | refer | seealso LinkType (Required)

For a full list of resources, refer: Resource list - FHIR v4.0.1 (hl7.org)

Key Part of Resource

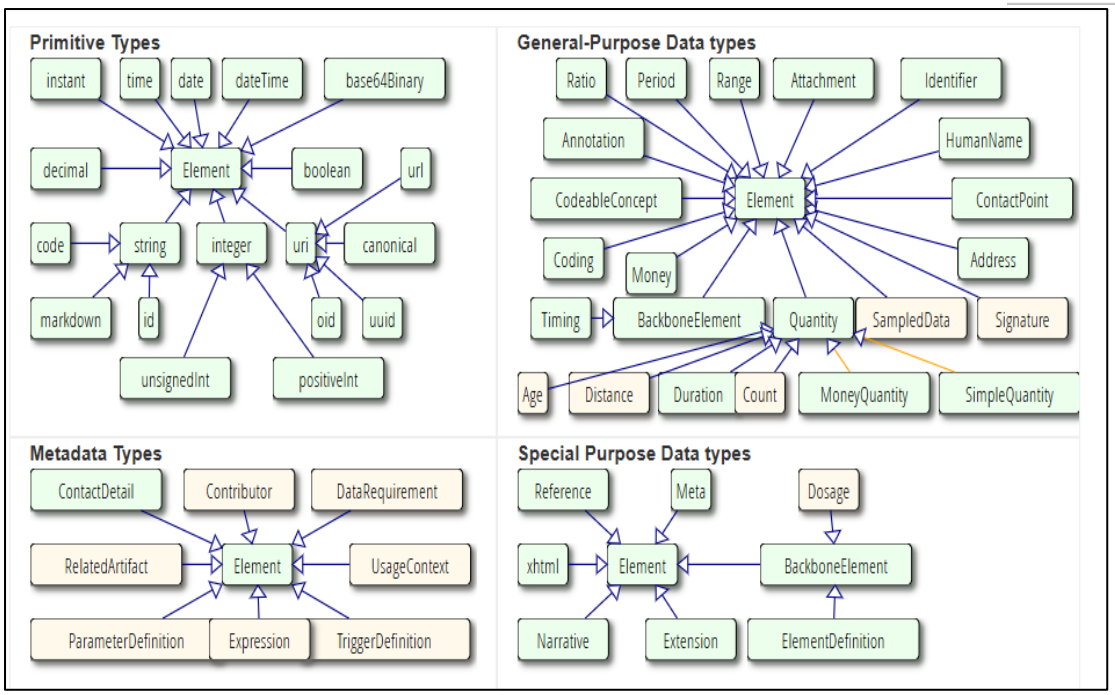


DATA TYPES

Datatypes define the structure and nature of data elements within resources, ensuring consistent representation of information across healthcare systems. They specify how data fields are formatted, such as strings, numbers, or more complex structures. This standardization is crucial for maintaining data integrity and enabling seamless data exchange between different healthcare applications and platforms. By using predefined datatypes, FHIR ensures that both simple and complex data elements are accurately captured and interpreted, enhancing interoperability and data consistency across systems.

Key datatypes

- **Simple/Primitive Types**: Basic data types with a single, indivisible value like Boolean, integer, string, or date.
- **General-purpose Complex Types**: Reusable clusters of elements that represent structured data like Address, HumanName, or Identifier.
- **Metadata Types**: Used to describe metadata associated with resources.
- **Special Purpose Data Types**: Types created for specific healthcare-related use cases, such as Dosage, Reference, and Meta.



Refer: Datatypes - FHIR v4.0.1 (hl7.org)

FHIR PARADIGM

FHIR is designed to support a variety of paradigms or approaches to healthcare data exchange, enabling flexible and interoperable communication between systems. It combines different data exchange methods such as **RESTful APIs**, **documents**, **messages**, and **services** to accommodate various workflows and requirements in healthcare settings.

- **RESTful API**: FHIR's most widely used paradigm, where data is exchanged using standard HTTP operations (GET, POST, PUT, DELETE), making it efficient and easy to implement.
- **Documents**: In scenarios where a complete, self-contained set of resources needs to be transmitted, FHIR supports the use of structured documents (e.g., discharge summaries).
- **Messaging**: FHIR supports messaging paradigms, enabling systems to exchange event-driven data, such as lab results or admission notifications.
- **Services**: For more complex interactions, FHIR allows for service-oriented exchanges, and supporting workflows like decision support or scheduling.

These paradigms offer flexibility in how healthcare data is shared and managed, making FHIR adaptable to a wide range of clinical, administrative, and regulatory use cases.

Refer: Exchange module - FHIR v4.0.1 (hl7.org)

BUNDLE

A FHIR Bundle is a structured container that holds a collection of related resources. It is used for grouping multiple healthcare resources, making it easier to transport them as a single unit. Bundles are commonly used in healthcare data exchange, allowing different resources, like patient records or clinical documents, to be transmitted together.

In ABDM, Health Information Types (HI Types/Clinical Artefacts) that represent discrete documents essential for continuity of care are defined using FHIR Bundle with 'type' as *'Document'*. However, the information needed for various claim processing workflows is represented using FHIR Bundle with the 'type' as '*Collection'*.

Use of Bundle with 'type' as 'Document'

The FHIR Bundle type *'Document'* is used to represent clinical artifacts that are discrete and self-contained documents, crucial for maintaining continuity of care in the healthcare system.

How it works

- A **Bundle** contains a **Composition** resource as its first entry. This Composition serves as the root or header, summarizing the overall document structure.
- The other resources within the Bundle are **referenced by the Composition**, such as Patient, Practitioner, Observation, etc.
- The Bundle ensures that the document, along with its related resources, is exchanged as a cohesive unit.

Example Use Case

A diagnostic report that includes the test results, interpretation, and conclusion. This information is structured using various FHIR resources (e.g. Patient, Diagnostic Report, Observation) and grouped within a Bundle as a document.

Use of Bundle with 'type' as 'Collection'

The FHIR Bundle type *'Collection'* is used to represent sets of related resources in a single package for ease of distribution. A *'Collection'* Bundle functions to organize resources relevant to a specific workflow.

Example Use Case

A healthcare claim submission that includes details about the patient, services provided, diagnosis, and associated costs. This information is structured using various FHIR resources (e.g., Claim, Patient, Coverage, Practitioner, Procedure, Condition) and grouped within a Bundle of type *'Collection'* to represent the complete claim request.

Key Elements of a FHIR Bundle resource

- **Bundle.type**: Defines the purpose of the bundle, indicating how it should be processed (e.g., 'document', 'collection').
- **Bundle.timestamp**: The exact date and time when the bundle was created. It ensures accurate tracking of when the information within the bundle was assembled.
- **Bundle.identifier**: A unique value that distinguishes a specific Bundle from others, ensuring it can be identified across systems. It plays a key role in maintaining data integrity and traceability in healthcare exchanges.
- **Bundle.entry:** Each **entry** in a Bundle represents an individual resource that is part of the overall collection. A Bundle may contain one or more entries, depending on how many resources are being grouped. Each entry consists of several components:
  - **Full URL**: A reference to the specific resource, often providing a resolvable URL where the resource can be accessed.
  - **Resource**: The actual FHIR resource (e.g., Patient, Observation, or Encounter) that is being included in the Bundle.

Refer: Bundle - FHIR v4.0.1 (hl7.org)

FHIR IMPLEMENTATION GUIDE

FHIR Implementation Guide (IG) is a document that provides specific guidelines on how to implement the HL7 Fast Healthcare Interoperability Resources (FHIR) standard in a particular healthcare context or for a specific use case. It describes how the standard should be applied, customized, or extended for particular needs, ensuring interoperability and consistency in implementations across different systems.

Typically, a FHIR Implementation Guide contains:

- **Profile**: Customized versions of standard FHIR resources that specify constraints, extensions, and usage guidelines for the particular context.
- **Extension**: Custom fields or data points added to FHIR resources that aren't covered by the base standard.
- **ValueSet**: Defined sets of allowable codes or terminologies (such as ICD, SNOMED CT) that can be used in certain elements of FHIR resources.
- **Example**: Sample FHIR resource instances to demonstrate correct usage.
- **Narrative and Guidance**: Detailed explanations about how and why certain decisions were made, and how the FHIR resources should be used together.

By following an IG, healthcare organizations ensure that their systems are capable of exchanging information in a standardized manner, promoting interoperability.

FHIR Implementation Guide for Ayushman Bharat Digital Mission

The FHIR Implementation Guide for ABDM is built on FHIR Version R4 (4.0.1), it establishes the minimum conformance requirements for accessing health data to ensure continuity of care in ABDM. This guide defines the essential health record artifacts to be captured and exchanged in line with the ABDM.
It references key standards and coding systems from the National Digital Health Blueprint (NDHB), EHR Standards for India (2016), and regulatory bodies like the Medical Council of India (MCI), Pharmacy Council of India (PCI), and Health Claim Exchange Platform (NHCX).

Refer: Home - FHIR Implementation Guide for ABDM

Reading FHIR Profiles

The ABDM FHIR Implementation Guide includes several profiles to capture and exchange health data. Understanding how to read and interpret these profiles is essential for developers and healthcare providers. Below are key concepts to help understand ABDM FHIR profiles:

1. Statistics/References

- Provides a **human-readable summary** of changes made to the base FHIR resources. It refers to the **Differential View**, showing which elements have been modified, constrained, or extended in the profile.

2. Differential View

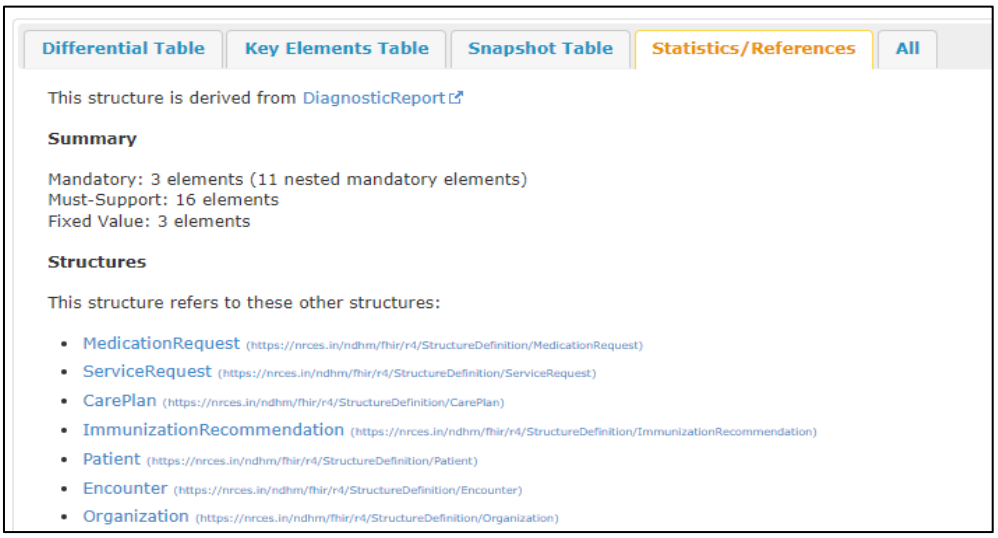
- The **Differential View** lists the specific changes applied to the FHIR resource while creating a profile. This includes constraints, extensions, and customizations tailored for ABDM.
- Only the modified elements are displayed, making it easier for implementers to focus on what has been changed from the standard FHIR resource.

3. Mandatory Element

- Elements with cardinality **'1..1'** or **'1..*'** are mandatory and must always be present in the resource. These elements are critical for the proper functioning of the data exchange and cannot be omitted.

4. Must Support S

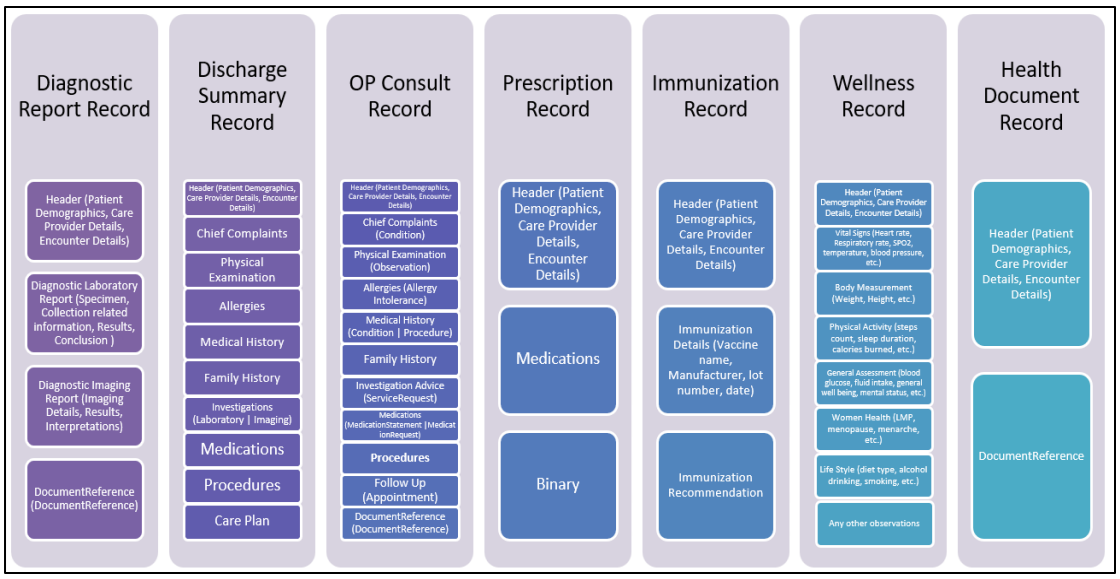
- **'MUST Support'** elements are optional to include but must be supported by receiving systems. The **Healthcare Information User (HIU)** must be able to process these elements if present, while the **Healthcare Information Provider (HIP)** can choose whether to include them. This ensures that systems can handle optional data when available.



Refer: [Formats - FHIR v4.0.1 (hl7.org)]

FHIR Profiles for ABDM

The ABDM artifacts aim to cover a wide range of health record document sharing within care settings. These artifacts ensure comprehensive data capture and exchange to support continuity of care. This includes 07 Clinical Artifacts, 01 Billing Artifacts, 38 Core Profiles, 42 Terminology ValueSets, and 92 examples.

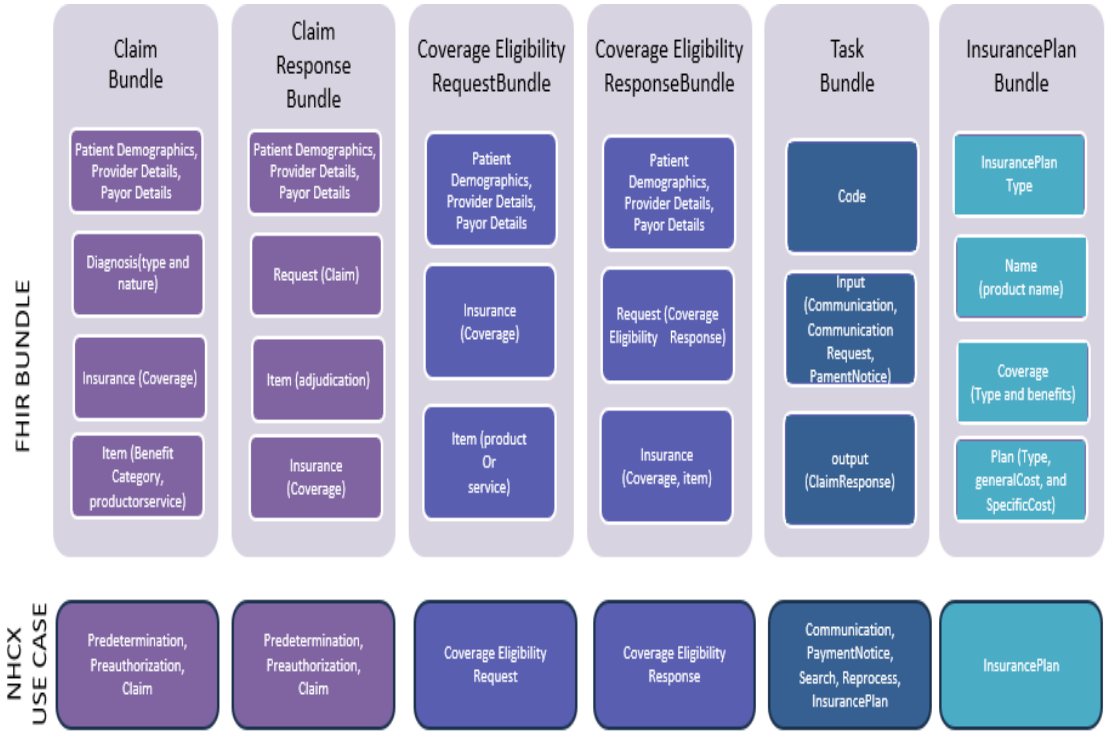


Refer: [ABDM Profiles - FHIR Implementation Guide for ABDM]

FHIR Profiles for NHCX

The NHCX artifacts are designed to facilitate standardized and efficient exchange of health claim-related information among payers, providers, beneficiaries, and other stakeholders. These artifacts support a range of processes related to health

claims, including eligibility checks, pre-authorization requests, claims submissions, and payment notifications. They ensure that data is exchanged in an interoperable, machine-readable, and auditable format, promoting accurate and timely processing of health claims. This includes 06 Health Claim Artifacts, 11 Core Profiles, 10 CodeSystem 17 ValueSets, and 51 examples.



Refer: [NHCX Profiles - FHIR Implementation Guide for ABDM]

IMPLEMENTING AND VALIDATING FHIR USING JAVA

This document provides a comprehensive guide for creating and validating FHIR resources programmatically using Java. We will use the ***HAPI FHIR library***, a popular choice for working with FHIR in Java.

HAPI FHIR Library

The ***HAPI FHIR library*** is an open-source Java framework that simplifies the process of working with FHIR (Fast Healthcare Interoperability Resources) resources. It provides comprehensive support for creating, manipulating, validating, serializing, and parsing FHIR resources.

Refer: HAPI FHIR - The Open Source FHIR API for Java

Key Features of HAPI FHIR Library

- **Complete FHIR Support**: HAPI FHIR offers full support for all FHIR resource types. This includes creating, validating, serializing, and parsing each resource, ensuring compliance with FHIR standards.
- **Built-in Validators**: The library includes a robust validation framework that checks FHIR resources against the FHIR specification. You can validate resources using predefined profiles, custom profiles, or specific rules.
- **Serialization & Parsing**: The library provides seamless serialization and deserialization capabilities for FHIR resources in both **JSON** and **XML** formats. The parsing functions convert the structured data into Java objects for easy manipulation.
- **Extensive Resource Coverage**: The library contains Java classes representing every resource in the FHIR specification, such as Patient, Practitioner

HAPI FHIR Dependencies

- **hapi-fhir-structures-r4**: This library provides Java classes and structures for all FHIR R4(4.0.1) resources, enabling the creation and manipulation of FHIR resources in Java applications.
- **hapi-fhir-validation**: This module performs validation of FHIR resources against standard and custom FHIR profiles, ensuring that the resources conform to the specified FHIR rules and constraints.
- **hapi-fhir-validation-resources-r4**: This library includes predefined resources and profiles required for validating specific FHIR R4 ensuring proper compliance with FHIR.

Creating FHIR Resources Programmatically

Prerequisites

Before you start, ensure you have:

- **Java Development Kit (JDK):** Ensure JDK 11 or higher is installed.
- **Integrated Development Environment (IDE):** Use an IDE like IntelliJ IDEA or Eclipse.

Setting Up the Development Environment

✓ **Add HAPI FHIR Dependency**

Add the HAPI FHIR dependency to your ***'pom.xml'*** file if you are using Maven

```
<!-- https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-structures-r4 -->
<dependency>
   <groupId>ca.uhn.hapi.fhir</groupId>
   <artifactId>hapi-fhir-structures-r4</artifactId>
   <version>6.4.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-validation -->
<dependency>
   <groupId>ca.uhn.hapi.fhir</groupId>
   <artifactId>hapi-fhir-validation</artifactId>
   <version>6.4.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-validation-
resources-r4-->
<dependency>
   <groupId>ca.uhn.hapi.fhir</groupId>
   <artifactId>hapi-fhir-validation-resources-r4</artifactId>
   <version>6.4.3</version>
</dependency>
```

Alternatively, if using Gradle, add the following to your build.gradle:

```
// https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-structures-r4
implementation group: 'ca.uhn.hapi.fhir', name: 'hapi-fhir-structures-r4', version: '6.4.3'

// https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-validation
implementation group: 'ca.uhn.hapi.fhir', name: 'hapi-fhir-validation', version: '6.4.3'

// https://mvnrepository.com/artifact/ca.uhn.hapi.fhir/hapi-fhir-validation-
resources-r4
implementation group: 'ca.uhn.hapi.fhir', name: 'hapi-fhir-validation-resources-r4',
version: '6.4.3'
```

✓ **Set Up the Project**

Create a new Java project in your IDE and configure it to include the HAPI

Creating Resource

This section outlines the basic steps involved in creating a FHIR resource using Java. The process involves defining, populating, and serializing the resource.

1. **Define the Resource:** This step involves instantiating the specific FHIR resource. Each FHIR resource (e.g., Patient, Bundle, Observation) represents a particular element in the healthcare data model.
    - Use the HAPI FHIR library to create a new instance of the required resource.

```
// Define a Patient resource
Patient patient = new Patient();
```

2. **Populate the Resource:** Once the resource is defined, populate it with appropriate values. This includes setting the attributes, identifiers, names, dates, and other elements in line with the FHIR specification.
    - Use setter methods provided by the library to populate attributes. Make sure to include mandatory fields and ensure data consistency.

```
// Populate patient resource with details
patient.addIdentifier()
    .setSystem("http://hospital.org/patients")
            .setValue("12345");
patient.addName().setText("ABC");
patient.setGender(AdministrativeGender.MALE);
patient.setBirthDateElement(new DateType("2024-01-01"));
```

3. **Serialize the Resource:** After populating the resource, the next step is to serialize it into a format that can be shared or stored. FHIR resources are typically serialized in either JSON or XML formats.
    - Use the HAPI FHIR library's serialization functionality to convert the resource into a desired format (JSON or XML).

```
// Serialize the resource to JSON format
FhirContext ctx = FhirContext.forR4();
String serializedResource = ctx.newJsonParser()
                    .setPrettyPrint(true)
                .encodeResourceToString(patient);
```

    - Also serialize the resource to XML if required:

```
// Serialize the resource to XML format
String serializedResourceXml = ctx.newXmlParser()
                    .setPrettyPrint(true)
                .encodeResourceToString(patient);
```

Patient Resource in Java and JSON Representation

The below image demonstrates how to define and populate a Patient resource in Java, along with its corresponding JSON output after serialization. The Java code example highlights the creation of the resource, while the JSON representation shows the final structured data format that adheres to the FHIR specification.

```
public static Patient populatePatientResource() {
Patient patient = new Patient();
patient.setId("b7bb7467-0f45-4a86-a1e0-
ca6a7c116f5f");
patient.getMeta()
       .addProfile("https://nrces.in/ndhm/fhir/r4/
StructureDefinition/Patient");
patient.addIdentifier()
      .setType(
      new CodeableConcept(
      new Coding(
        "https://nrces.in/ndhm/fhir/r4/CodeSystem
/ndhm-identifier-type-code",
         "ADN",
         "Adhaar number")))
      .setSystem("https://uidai.gov.in/")
      .setValue("7225-4829-5255");
patient.addName()
     .setText("ABC");
patient.setGender(AdministrativeGender.MALE)
     .setBirthDateElement(
       new DateType("1981-01-12"));
   return patient;
}
```

Patient Resource in Java

```
{
  "resourceType" : "Patient",
  "id" : "b7bb7467-0f45-4a86-a1e0-ca6a7c116f5f",
  "meta" : {
    "profile" :
["https://nrces.in/ndhm/fhir/r4/StructureDefiniti
on/Patient"]
  },
  "identifier" : [{
    "type" : {
      "coding" : [{
        "system" :
"https://nrces.in/ndhm/fhir/r4/CodeSystem/ndh
m-identifier-type-code",
        "code" : "ADN",
        "display" : "Adhaar number"
      }]
    },
    "system" : "https://uidai.gov.in/",
    "value" : "7225-4829-5255"
  }],
  "name" : [{
    "text" : "ABC"
  }],
  "gender" : "male",
  "birthDate" : "1981-01-12"
}
```

Patient Resource in Java

Creating FHIR Bundles Programmatically in ABDM and NHCX

FHIR Bundles can group resources together for specific workflows. In ABDM and NHCX, bundles are categorized based on their type, either Document (used for clinical workflows) or Collection (used for administrative workflows like claim processing). Below is a guide on how to create these types of bundles in Java, leveraging the HAPI FHIR library.

Creating a Document Bundle for ABDM

In the context of ABDM, FHIR Bundles of type *'Document'* are used to represent clinical documents. The Composition resource serves as the first entry and acts as the header for the document, Composition provides essential structure and context, including details such as the author, date of creation, and overall clinical narrative. This structure links all the resources within the document, creating a clear and meaningful representation of the healthcare data. For example, in a Diagnostic Report, the Composition organizes and presents the diagnostic findings along with related information.

- **Bundle Type**: Document
- **First Entry**: Always a Composition resource.

```
// Creating a Diagnostic Report Bundle for ABDM
Bundle diagnosticReportBundle = new Bundle();

// Set logical id of this artifact
diagnosticReportBundle.setId("DiagnosticReport-Imaging-DCM-example-01");

// Set metadata about the resource
Meta meta = diagnosticReportBundle.getMeta();
meta.addProfile("https://nrces.in/ndhm/fhir/r4/StructureDefinition/DocumentBundle");
diagnosticReportBundle.setMeta(meta);

// Set Bundle Type
diagnosticReportBundle.setType(BundleType.DOCUMENT);

// Adding Composition as the first entry
BundleEntryComponent compositionEntry = new BundleEntryComponent();
compositionEntry.setFullUrl("urn:uuid:df810c39-55e7-441c-8569-d6ab77aa1c66");
compositionEntry.setResource(populateDiagnosticReportRecordDCMCompositionResource());

// Adding additional entries (e.g., Patient, Practitioner, etc.)
BundleEntryComponent patientEntry = new BundleEntryComponent();
patientEntry.setFullUrl("urn:uuid:1efe03bf-9506-40ba-bc9a-80b0d5045afe");
patientEntry.setResource(populatePatientResource());

diagnosticReportBundle.addEntry(compositionEntry);
diagnosticReportBundle.addEntry(patientEntry);

// Adding more resources as necessary (Practitioner, ImagingStudy, etc.)
```

Creating a Collection Bundle for NHCX

In NHCX, bundles of type *'Collection'* are used for administrative purposes such as health claim processing, they group related resources (like Claim, Coverage, Patient) for processing workflows.

- **Bundle Type:** Collection.

```
// Creating a Claim Bundle for NHCX
Bundle claimBundle = new Bundle();

// set Id - Logical id of this artifact
claimBundle.setId("ClaimBundle-preauth-example-01");

// set Meta - Metadata about the resource
Meta meta = new Meta();
meta.addProfile("https://nrces.in/ndhm/fhir/r4/StructureDefinition/ClaimBundle");
claimBundle.setMeta(meta);

//set Identifier
claimBundle.setIdentifier(new Identifier(). setSystem("http://hip.in").
        setValue("bc3c6c57-2053-4d0e-ac40-139ccccff645"));

// set Type - collection
claimBundle.setType(BundleType.COLLECTION);

// Adding entries for claim-related resources
BundleEntryComponent claimEntry = new BundleEntryComponent();
claimEntry.setFullUrl("urn:uuid:4776dbdf-d596-4cd1-9966-9d44ae9dec0b");
claimEntry.setResource(ResourcePopulator.populateClaimSettlementResource());

BundleEntryComponent patientEntry = new BundleEntryComponent();
patientEntry.setFullUrl("urn:uuid:1efe03bf-9506-40ba-bc9a-80b0d5045afe");
patientEntry.setResource(ResourcePopulator.populatePatientResource());

claimBundle.addEntry(claimEntry);
claimBundle.addEntry(patientEntry);

// Adding more resources as necessary (Coverage, Organization, etc.)
```

Common Pitfalls in Creating FHIR Bundles

When implementing FHIR bundles, it's essential to adhere to specific rules and best practices to ensure the bundle is correctly structured and conforms to the required profiles. Below are common pitfalls to watch out for when creating bundles, especially in the context of ABDM and NHCX.

1. Incorrect Resource Format: Ensure that each resource within the bundle adheres strictly to the FHIR Specification Version R4, including the required structure, data types, and relationships. Use built-in FHIR validators to check that the resources are correctly formatted. For example, when creating a Patient resource, ensure that fields such as name and identifier are properly structured according to FHIR rules.

2. Missing Required Fields: In the ABDM and NHCX profiles, certain elements are mandatory. A common issue occurs when these required fields are missing, which can result in improper processing of the bundle. Ensure that all mandatory elements, as defined by the cardinality in FHIR profiles (e.g., 1..1 or 1..*), are included when creating the resource.

3. Invalid Data Types: Ensure that the data types used for each element conform to the FHIR Specification Version R4. Incorrect data types, such as using a string instead of an integer, can cause validation errors.

4. Profile Violations and Missing *'meta.profile'* Element: All resources should comply with specific FHIR profile constraints, including properly populating the *'meta.profile'* element with the appropriate canonical URL of the profile. This applies to every resource, including the bundle itself.

5. Use of *'urn:uuid'* in fullUrl: When referencing resources within a bundle, the fullUrl element should use the *'urn:uuid'* format, which ensures each resource is uniquely identified within the bundle. The correct format is urn:uuid:<unique-id>, where <unique-id> is the UUID of the resource.

6. Incorrect System URL in CodeableConcept: For any element of the data type CodeableConcept, ensure that the correct coding system URL is specified in the coding.system element. It is recommended to select the code and display values from the bound value sets to ensure consistency and avoid potential issues during validation or data exchange.

Validating FHIR Resources

Validation Tools

1. **HAPI FHIR Validator:** A built-in validation tool in HAPI FHIR library.
    - **URL:** HAPI FHIR Validator
2. **FHIR Validator CLI:** Command-line interface for FHIR validation.
    - **URL:** FHIR Validator CLI
3. **FHIR GUI Validator:**
    - **URL:** FHIR GUI Validator

Validation Aspects

When validating a FHIR resource, several key aspects are checked to ensure compliance with the FHIR specification:

1. **Structure:** Verifies that the resource conforms to the FHIR specification, with no extra or undefined elements present.
2. **Cardinality:** Ensures that properties adhere to their defined cardinality (minimum and maximum occurrences).
3. **Value Domains:** Confirms that property values match their data types and enumerated codes are valid.
4. **Coding/CodeableConcept Bindings:** Validates that Coding or CodeableConcept elements use correct system URL, codes and display values as per required valuesets.
5. **Invariant:** Checks that all constraints or co-occurrence rules are satisfied (e.g., if one field is present, another must also be present).
6. **Profile:** Ensures compliance with specific rules defined in FHIR profiles (including those listed in the Resource.meta.profile, or in CapabilityStatement, or in an ImplementationGuide, or otherwise required by context)
7. **Business Rules:** Includes additional checks like duplicate detection, reference resolution, and authorization validation.

These aspects ensure that FHIR resources are structurally sound and meet both clinical and business requirements.

Refer: Validation - FHIR

Validation Process

1. **HAPI FHIR Validator:** To validate a resource using the HAPI FHIR library, following are the steps:
   - **Step 1:** Load NPM Package
     - Download the package.tgz containing all the Structure Definitions, CodeSystems, and ValueSets from ABDM FHIR Implementation Guide.
     - Add package.tgz to the class path *("src/main/resource")*
     - ***"package.tgz"*** is essential for validating resources against ABDM and NHCX profiles and terminologies.

```
// Create NpmPackageValidationSupport instance
NpmPackageValidationSupport npmValidationSupport = new
NpmPackageValidationSupport(ctx);

// Load package from classpath
npmValidationSupport.loadPackageFromClasspath("classpath:package.tgz");
```

For more Information Refer: Instance Validator using package

- **Step 2:** Setup a validation support chain
  - Establish a validation support chain that incorporates the core FHIR structure definitions. This chain includes FHIR StructureDefinition and FHIR's built-in vocabulary (such as ValueSet and CodeSystem resources).
  - It involves an in-memory terminology service, module caching, and support for validating codes with CodeSystems that are not distributed as part of the FHIR specification.

```
// Create a chain that will hold our modules
ValidationSupportChain validationsupportchain = new ValidationSupportChain(
    npmValidationSupport,                                    // NPM package support
    new DefaultProfileValidationSupport(ctx),                // Default profile validation
    new InMemoryTerminologyServerValidationSupport(ctx), // Terminology validation
    new CommonCodeSystemsTerminologyService(ctx),            // Common code system
    new SnapshotGeneratingValidationSupport(ctx)             // Generate snapshots of
                                                                StructureDefinitions
);
// Add caching layer on top of the validation chain
CachingValidationSupport validationSupport = new
CachingValidationSupport(validationsupportchain);
```

Refer: Validation Support Modules

- **Step 3:** Register validator and validate resource

```
// Initialize FHIR context for R4
FhirContext ctx = FhirContext.forR4();

// Declare FhirValidator and FhirInstanceValidator
FhirValidator validator;
FhirInstanceValidator fhirInstanceValidator;

// Initialize the validator
validator = ctx.newValidator();

// Set up FhirInstanceValidator with the validation support chain
fhirInstanceValidator = new FhirInstanceValidator(validationSupport);
validator.registerValidatorModule(fhirInstanceValidator);

// Validate a FHIR resource
ValidationResult outcome = validator.validateWithResult(resource);

// Print the overall validation outcome
System.out.println(outcome);

// Loop through and print individual validation messages
for (SingleValidationMessage next : outcome.getMessages()) {
System.out.println(next.getSeverity() + " - " + next.getLocationString() + " - " +
next.getMessage());
}
```

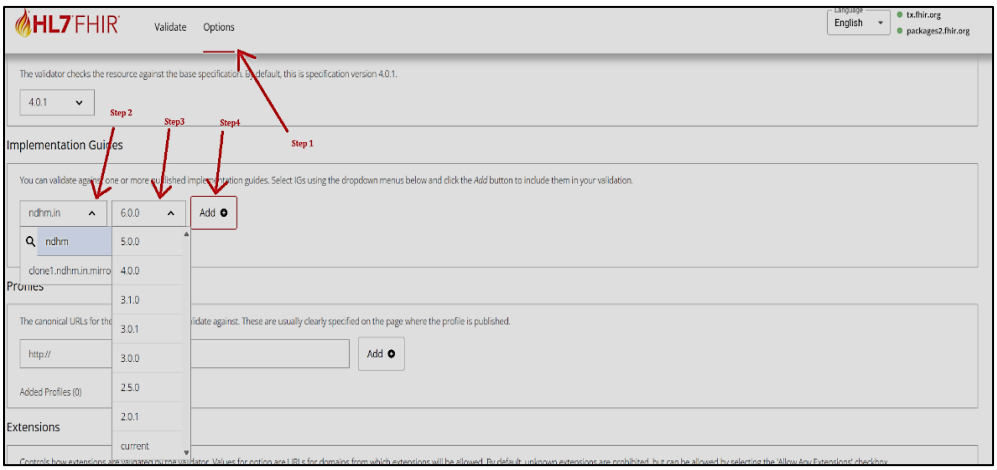
For more Information Refer: Instance Validator using package

2. **FHIR Validator CLI:** To validate a resource using the JAR file provided by HL7, use following command:

   "java -jar <path to validator_cli.jar> <file_name> -ig ndhm.in#<ig-version>"

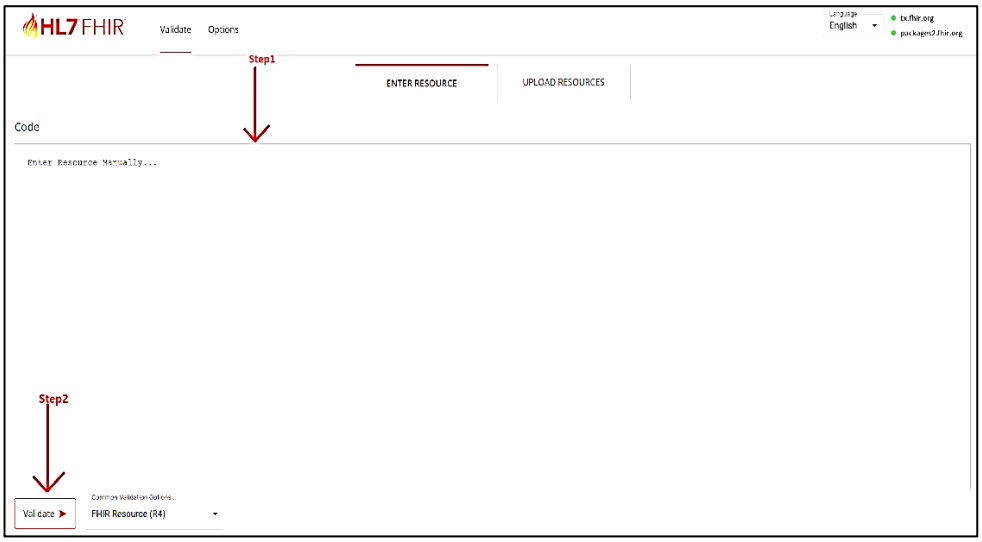
   For Documentation refer: [Using the FHIR Validator - FHIR - Confluence (hl7.org)]

3. **FHIR GUI Validator**
   - Navigate to [https://validator.fhir.org]
   - Add Implementation Guide



   - **Step1:** Click on the "Options" tab
   - **Step2:** Select the implementation guide "ndhm.in"
   - **Step3:** Choose the latest version from the dropdown.
   - **Step4:** Click on the "Add" button to add IG.

   - Validation process



- **Step1:** Paste the FHIR resource that need to be validated.
- **Step2:** Click on the "Validate" button to begin the validation process. The validator will check the resource against the selected implementation guide and provide feedback on any errors or warnings.

IMPLEMENTATION REFERENCE

- **Implementation Guide**
  - HL7 : Index - FHIR v4.0.1 (hl7.org)
  - ABDM : Home - FHIR Implementation Guide for ABDM
- **Implementation Libraries**
  - Java : HAPI FHIR - The Open Source FHIR API for Java
  - C# : Firely .NET SDK | The official .NET SDK for HL7 FHIR
  - JavaScript : fhir-kit-models-npm (npmjs.com)
  - Typescript : ts-fhir-types - npm (npmjs.com)
  - Additional : Open Source Implementations - FHIR - Confluence
- **Tool**
  - Validator cli : Using the FHIR Validator - FHIR - Confluence
- **Schema**
  - JSON : JSON Schema
- **Usage Sample**
  - Java : Usage Sample code – JAVA
  - .NET : Usage Sample code - DOTNET

ANNEXURE: FHIR Bundles utilized by each NHCX API based on specific Use Cases

S.N.	Use Case	API End Point	Flow	FHIR Bundle
1	Coverage Eligibility	/coverageeligibility/check	provider->NHCX->payer	CoverageEligibiltyRequestBundle
2	Coverage Eligibility	/coverageeligibility/on_check	payer->NHCX->provider	CoverageEligibiltyResponseBundle
3	Preauthorization	/preauth/submit	provider->NHCX->payer	ClaimBundle
4	Preauthorization	/preauth/on_submit	payer->NHCX->provider	ClaimResponseBundle
5	Predetermination	/predetermination/submit	provider->NHCX->payer	ClaimBundle
6	Predetermination	/predetermination/on_submit	payer->NHCX->provider	ClaimResponseBundle
7	Claim	/claim/submit	provider->NHCX->payer	ClaimBundle
8	Claim	/claim/on_submit	payer->NHCX->provider	ClaimResponseBundle
9	Request Additional Attachments	/communication/request	payer->NHCX->provider	TaskBundle
10	Send Attachments	/communication/on_request	provider->NHCX->payer	TaskBundle
11	Payment	/paymentnotice/request	payer->NHCX->provider	TaskBundle
12	Payment	/paymentnotice/on_request	provider->NHCX->payer	TaskBundle
13	Search	/search/submit	NHA->NHCX->Payer	TaskBundle
14	Search	/search/on_submit	payer->NHCX->NHA	TaskBundle
15	Reprocess	/task/submit	provider->NHCX->payer	TaskBundle
16	Reprocess	/task/on_submit	payer->NHCX->provider	TaskBundle
17	Status Check	/hcx/status	provider->NHCX,Payer->NHCX	NA
18	Status Check	/NHCX/on_status	provider->NHCX,Payer->NHCX	NA