



FHIR Intermediate Course, Unit 4
SMART ON FHIR & CDS HOOKS
Reading Material

Contents

This unit's goals	4
Introduction to SMART on FHIR and CDS Hooks	4
SMART on FHIR	6
Introduction	6
Resources	7
The framework	7
Scope	7
Application Protection	8
Support for "public" and "confidential" applications	8
Register a SMART application with an EHR	9
Creating a Javascript-based application	9
Including the SMART-on-FHIR Java Script library	9
Example H-1: Including the Smart On FHIR Client	9
Description of the SMART authorization and FHIR access	10
Launching from EHR	10
Example H-2: Launching an app from an EHR	10
Standalone launch	11
Workflow of a SMART on FHIR application	12
Step 1: The application requests authorization	13
Example H-4: Script for launch.html	16
Step 2: Evaluate the authorization request	17
Step 3: Exchanging the authorization code for the access token	18
Example H-5: Authorization Request	19
Example H-6: Authorization Response	20
Step 4: Access to clinical data through the FHIR API	21
Example H-7: Requesting Resources with Bearer Token	21
Example H-8: Response From the Server	21
Step 5: Updating credentials	25
Example H-9: Updating credentials (refresh token request)	25
Example H-10: Updating credentials (refresh token response)	26
User interface applications	27
Background applications and services	27

Recommendations	27
CDS Hooks	29
What is a hook	29
How does it work?	29
Basic Components	30
Discovery of CDS services	31
Example H-11: Service Discovery	31
Calling a CDS service	33
Example H-12: Calling a CDS Service using patient-view hook	35
Example H-13: order-select hook from our project	36
Context Data	37
Prefetched Data	38
Example H-14: Prefetch Example	40
Prefetch template	40
Example H-15: Prefetch Template	40
Prefetch tokens	41
Example H-16a: A context with a FHIR patient resource	42
Example H-16b: A context with a FHIR patient resource and an encounterId	42
Example H-17: A CDS-Hooks card	46
Feedback	47
Suggestion accepted	48
Card ignored	48
Overridden guidance	48
Extensions	49
Example H-18: Supporting extensions	49
Security and Protection	49
Good Practices	50
Security	50
CDS Clients	50
CDS Services	50
Maturity Model of Hooks	51
Useful Links	52

This unit's goals

After Unit 4, you will be able to:

- Explain the need and use case for Smart-On-FHIR
- Follow the Smart-On-FHIR workflow
- Change the configuration of a Smart-On-FHIR connected App to reflect specific scope needs
- Describe the use cases and scope for CDS-Hooks
- Use a CDS Hook service and display cards.

Introduction to SMART on FHIR and CDS Hooks

In this unit we will look at two major projects that are categorized as "FHIR variants," SMART on FHIR and CDS Hooks, and we will work on a practical case combining both projects.

The idea is that by the end of the unit, you will know the fundamentals of each of these projects, their objectives, the tools they use and their current state, and you will end up with a functioning project.

References to the project we will work on will be placed within a box with an app icon, for easy identification.




To contextualize the project: This application initiates an EHR using SMART App Launcher. In this EHR we will work with the prescription module that will be implemented through a SMART on FHIR application.

This module will allow us to see the patient's prescriptions for medication (MedicationRequest) and for each medication will allow us to call a CDSS service using the CDS Hooks standard and check the medication price.

Once the context is established, we will focus on the details of each specification and we will refer to the project for better integration. We'll start with SMART on FHIR, since CDS-Hooks will be used in the SMART on FHIR application.

Throughout this unit you will see two distinct icons:



The **'example'** icon means that the following is a code example with actual values for the parameters it uses. Some examples are not in-line with the text due to their length. You will need to refer to an external file. Some of the long examples are also in github gist. When such an example is included, you will also see the github icon: 



The **"reference project"** icon signals that we are explaining how a certain feature of Smart-On-FHIR or CDS Hooks is applied in our example project (FHIR Server+CDS Hooks Service+Fake EHR+App)

Sometimes to illustrate a concept we use examples and also explain how the concept was used for our project.

SMART on FHIR

Introduction

When we are creating a healthcare-related application, it will need to access data on patients, results, orders, and medications, among other things. For a developer, one of the main difficulties in creating such an application is that obviously, health systems do not all use the same format or the same standards, etc. This raises the possibility of having to rewrite an application for each institution that wants to use it.

On the other hand, those who need an application find it difficult to try different ones at a reasonable cost because to do so, they would have to address integration issues and ensure that the new application worked with their system.

This makes it difficult for new applications to come into existence and for hospitals to test new applications. The architecture of SMART on FHIR tries to solve this problem with interchangeable medical applications based on reusable technologies that allow developers to generate only one application and health organizations to test solutions from several developers until they find what they need. The acronym SMART stands for Substitutable Medical Applications and Reusable Technologies, and "on FHIR" indicates, obviously, that it uses resources from this HL7 standard.

Some initial definitions:

- A SMART on FHIR system is a system that implements the SMART on FHIR specification, including FHIR, OAuth2 and OpenID Connect profiles, and which is capable of running SMART applications.
- A SMART on FHIR application runs against a SMART on FHIR system, extending its functionality through the use of clinical and contextual data.
- We assume that at this point you know OAuth2 and OpenID Connect is a web standard for authentication. It defines an OAuth2-based protocol that allows end users to log into applications using external identity providers.

In this way, the SMART on FHIR project

- provides a standard on
 - how systems and their applications are authenticated and integrated by standardizing these processes.
 - How server handle the authorization of a user who accesses the data of a patient.
- Maintains an open-source library for use by developers.
- Provides and maintains free "sandbox" API servers that can be used to test applications. These "sandboxes" are important because a SMART application cannot be launched simply by navigating to a URL.

SMART provides a "context" to the application when it is launched from the medical record that contains information about the user and the patient. The sandbox in this case simulates a system that launches the application with one of the many possible contexts.

Resources

- On the official site you can find an extensive gallery of apps <https://apps.smarthealthit.org/>
- Documentation that includes tutorials, software libraries, test environments, test sandbox and sample applications <http://docs.smarthealthit.org/>

The framework

The framework for launching a SMART application is responsible for connecting third-party applications to electronic medical records data, allowing applications to be launched inside or outside the user interface of an electronic medical records (EHR) system.

This framework:

- Supports applications for use by doctors, patients and others through a personal medical records portal (PHR) or any other FHIR system where a user can grant permissions to launch an application.
- It provides a secure and reliable authorization protocol for a variety of application architectures: applications that run on the device of an end user and applications that run on a secure server.
- It supports the four use cases defined for Phase 1 of the Argonaut Project:
 1. Patient applications that launch standalone
 2. Patient applications that launch from a portal
 3. Provider applications that launch standalone
 4. Provider applications that launch from a portal

Scope

This framework is intended for application developers who need to access FHIR resources by requesting access tokens to authorization servers that are compatible with OAuth 2.0.

It defines a method through which an application requests authorization to access a FHIR resource, and then uses that authorization to retrieve the resource. It is compatible with FHIR DSTU2 or higher.

OAuth 2.0 authorization servers are configured to mediate access based on a set of rules configured to enforce institutional policy, which may include the end-user authorization request.

Application Protection

The application is responsible for protecting itself against possible misconduct or malicious values entered in its redirection URL (for example, SQL injection) and for protecting authorization codes, access tokens and refresh tokens against unauthorized use and access (sent only through authenticated servers, via the HTTPS protocol).

The application developer should know about possible threats, such as malicious applications running on the same platform, counterfeit authorization servers and counterfeit resource servers, and implement countermeasures to help protect both the application itself and any confidential information it may contain. Tokens and other confidential data should only be kept in application-specific storage locations, and should not use recognizable locations throughout the system.

Support for "public" and "confidential" applications

This framework differentiates the two types of applications defined in the OAuth 2.0 specification: confidential and public.

The differentiation is based on whether the execution environment within which the application runs allows the application to protect the "secrets." Pure client-side applications (for example, HTML5/JS browser-based applications, iOS mobile applications or Windows desktop applications) may provide adequate security, but they may not be able to "keep a secret" in the OAuth2 sense. In other words, any "secret" key, code or string that is statically embedded in the application can be extracted by an end user or an attacker. Therefore, the security of these applications cannot depend on the "secrets" embedded at the time of installation.

- Use the **confidential app** profile if your application is able to protect a *client_secret*.
 - The application runs on a trusted server where *client_secret* is accessed only from the server side.
 - The application is a native app that uses additional technology (such as dynamic client registration and universal *redirect_uris*) to protect the *client_secret*
- Use the **public app** profile if your application is not able to protect a *client_secret*.
 - The application is an HTML5 or JS browser-based app that would expose the *client_secret* in the user space (such as our example project)
 - The application is a native app that can only distribute a *client_secret* statically

Register a SMART application with an EHR

Before a SMART application can run against an EHR, the application must be registered with that EHR's authorization service. SMART does not specify a standards-based registration process, but encourages EHR implementers to consider the OAuth 2.0 dynamic client registration protocol for a ready-to-use solution.

No matter how an application is registered with the EHR authorization service, at the time of registration, each SMART application must:

- Register zero or more fixed launch URLs.
- Register one or more fixed *redirect_uris*. In the case of native clients that follow the OAuth 2.0 specification for native applications, it may be appropriate to leave the port as a dynamic variable in a fixed redirect URI.

Creating a Javascript-based application

To create our application, the simplest thing is to include in our project the SMART on FHIR JavaScript library that will help you create browser-based SMART applications that interact with a FHIR REST API server. It will help you solve how to obtain authorization tokens, provide information about the user and patient record with context and make API calls to retrieve clinical data.

Including the SMART-on-FHIR Java Script library

The most recent code is always available in NPM CDN at <https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js>.

The inclusion of this script will create a global FHIR object.



Example H-1: Including the Smart On FHIR Client

```
<script src = "https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js"> </script>
```

Description of the SMART authorization and FHIR access

As already detailed, an application can be started from an existing EHR or patient portal session. This is known as EHR launch. Alternatively, it can be launched as a standalone launch application.

Launching from EHR

In the launch flow, a user has established a session in an EHR and then decides to launch an application. This could be an app for a single patient (which runs in the context of a particular patient record) or a user-level app (such as an appointment manager or a population dashboard). The EHR starts a "launch sequence" by opening a new browser instance (or iframe) that points to the registered launch URL of the application, sending the following parameters:

- **iss**: identifies the FHIR endpoint of the EHR. The application can use it to obtain additional details about the EHR, including its authorization URL.
- **launch**: opaque identifier for this specific launch, and any context of the EHR associated with it. *This parameter must be communicated again to the EHR at the time of authorization by passing along a **launch** parameter (explained below).*



Example H-2: Launching an app from an EHR


For example: `https://{app launch_uri}?launch=xyz123&iss=https://{fhir base url}`

Upon receiving the launch notification, the SMART application generates a request to the EHR FHIR server using the following addresses:

- GET `https://{fhir base url}/metadata` or
- GET `https://{fhir base url}/.well-known/smart-configuration`

That returns a JSON with, among other details, the URL of the OAuth2 authorization endpoint and the URL of the OAuth2 token endpoint of the EHR that will be used to request authorization to access FHIR resources. This is resolved by the library.

In our project: we use <https://launch.smarthealthit.org/>.


SMART Launcher

App Launch Options

Client Registration & Validation

Launch Type

Provider EHR Launch

Practitioner opens the app from within an EHR

FHIR Version

R4

Select what FHIR version your app should work with

Misc. Options
☐ Simulate launch within the EHR UI (launch within an iframe)

Simulated Error

None

Force the server to throw certain type of error (useful for manual testing).

Patient(s)

59bc5dbb-1957-47ef-b6b5-25952e6bfdab

Simulates the active patient in EHR when app is launched. If no Patient ID is entered or if multiple comma delimited IDs are specified, a patient picker will be displayed as part of the launch flow.

Provider(s)


e443ac58-8ece-4385-8d55-775c1b8f3a37

Simulates user who is launching the app. If no provider is selected, or if multiple comma delimited Practitioner IDs are specified, a login screen will be displayed as part of the launch flow.

Encounter

Select the most recent encounter if available

How to select the current Encounter


App's Launch URL

http://localhost/smartonfhir/launch.html

Launch

Launch Sample App

Full url of the page in your app that will initialize the SMART session (often the path to a launch.html file or endpoint)

http://localhost/smartonfhir/launch.html?iss=https%3A%2F%2Flaunch.smarthealthit.org%2Fv%2Fr4%2Ffhir&launch=WzAsIjU5YmM1ZGJiL TE5NTctNDdIZi1iNmI1LTl1OTUyZTZiZmRhYiIsImU0NDNhYzU4LThiY2UtNDM4NS04ZDU1LTc3NWMxYjhmM2EzNyIsIkFVVE8iLDAsMCwwLCliLClilClilClilClilClilClilClilLDAsMV0

Standalone launch

In a standalone launch, a user selects an application outside an EHR session. This app will be launched from its registered URL without a launch id.

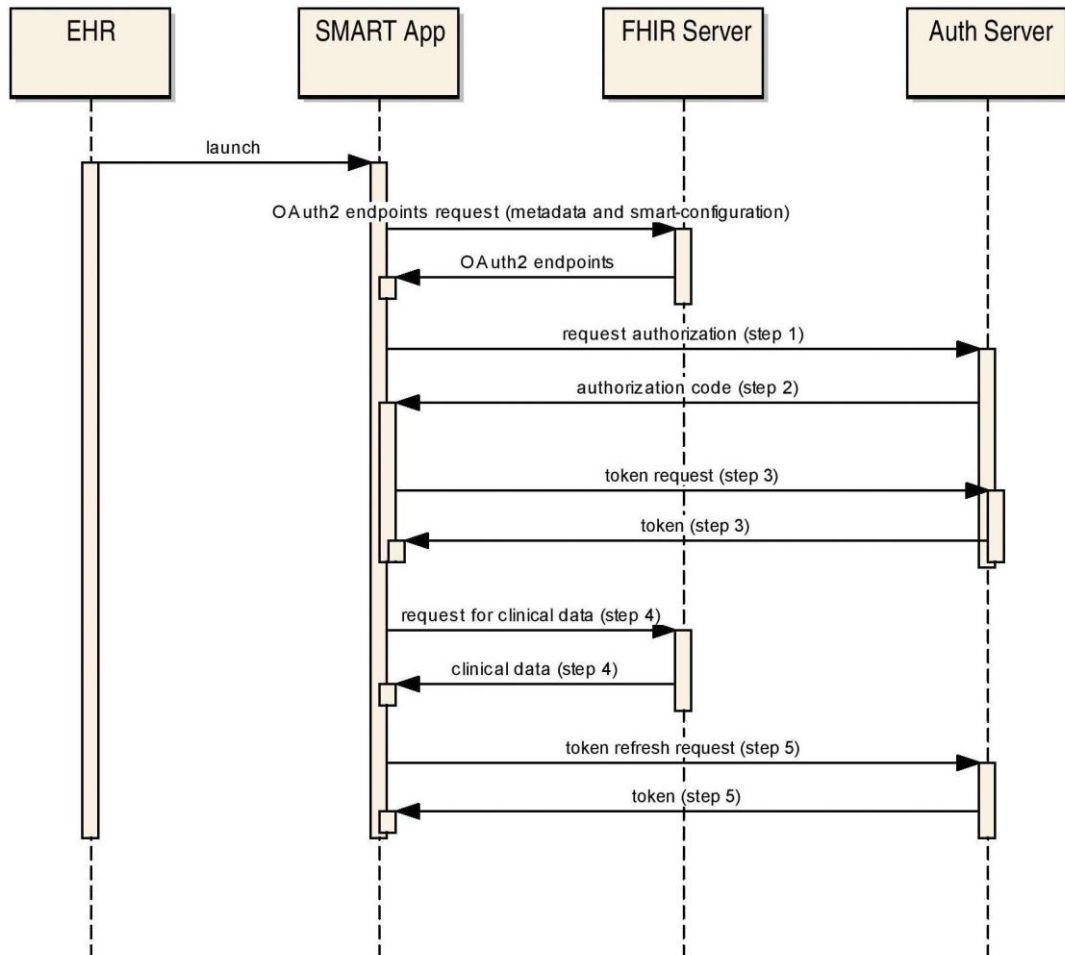
Once an application receives a launch request, to obtain a launch context and request authorization to access FHIR resources, the app generates a request to the EHR FHIR Server in the same way it does with "launch from EHR."

SMART application authorization against the EHR authorization server.

Once the JSON is obtained from the EHR FHIR server with the URL of the OAuth2 authorization endpoint and the URL of the OAuth2 token endpoint, the authorization sequence is initiated.

Workflow of a SMART on FHIR application

In the following diagram we can see all the steps that make up the launch and authorization of a SMART on FHIR application. In the following sections we will delve into all the steps:

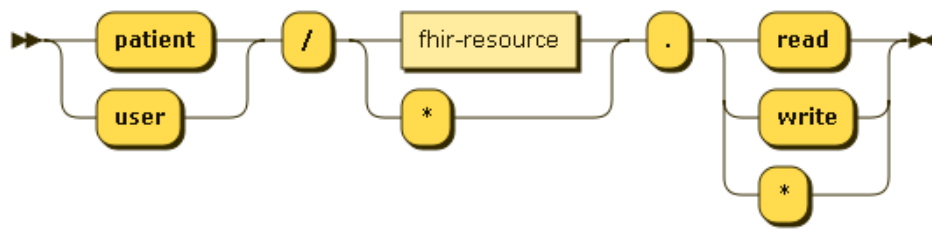


Step 1: The application requests authorization

At the time of launch, the application creates an authorization request by adding the following parameters to the query component of the endpoint URL used by the EHR to “authorize”:

- **response_type**: fixed value: code
- **client_id**: the client’s identifier.
- **redirect_uri**: must match one of the redirection URIs for the registered client.
- **launch**: when using the EHR launch flow, it must match the launch value received from the EHR. If we follow the example given above, it would be "xyz123". *This parameter is optional.*
- **scope**: must describe the access that the app needs, including scopes of clinical, context and identity data. Examples of the most used scopes (expressed using Smart-on-FHIR V1 and V2 syntax):
 - **patient/*.read or patient/.rs**: permission to read any resource related to the current patient.
 - **user/*.*** or **user/.cruds**: permission to read and write all the resources that the current user can access.
 - **openid fhirUser**: permission to retrieve information about the currently logged in user.
 - **launch**: permission to get the launch context when the app is launched from an EHR.
 - **launch/patient**: when the app launches outside the EHR and needs the context of the patient, the EHR authorization server can provide the end user with a list to select from.
 - **offline_access**: request a *refresh_token* that can be used to obtain a new access token to replace an expired one, even when the end user is no longer online after the access token expires.
 - **online_access**: request a *refresh_token* that can be used to obtain a new access token to replace an expired one, and that can be used while the end user remains online.

The V1 scope for clinical data is formed as follows:



1. use one of these if we are referring to the patient or the logged-in user
2. add a slash ("/")
3. then refer to any FHIR resource (*) or a specific FHIR resource (fhir-resource)
4. next, a dot (".")
5. then we refer to the operations that can be performed on the resources - either read (read), write (write) or any operation (*)

The V2 scope defines permissions based on "CRUDS" initials.

- c for create
- r for read
- u for update
- d for delete
- s for search

Valid suffixes are a subset of the in-order string **.cruds**.

For example, to convey support for creating and updating observations, use scope patient/Observation.cu. To convey support for reading and searching observations, use scope patient/Observation.rs.

For backwards compatibility with scopes defined in the SMART App Launch 1.0 specification, servers SHOULD advertise the permission-v1 capability in the .well-known/smart-configuration discovery document: they SHOULD return v1 scopes when v1 scopes are requested and granted, and SHOULD process v1 scopes with the following semantics in v2:

- v1 .read ⇒ v2 .rs
- v1 .write ⇒ v2 .cud
- v1 .* ⇒ v2 .cruds

Scope requests with undefined or out of order interactions MAY be ignored, replaced with server default scopes, or rejected. For example, a request of .dus is not a defined scope request. This policy is to prevent misinterpretation of scopes with other conventions (e.g., interpreting .read as .rd and granting extraneous delete permissions).

- **state**: an opaque value used by the client to maintain the state between the request and the callback. The authorization server includes this value when redirecting to the client. This parameter should be used to prevent counterfeit requests between sites or session fixation attacks.

The application must use an unpredictable value for this parameter with at least 122 entropy bits (for example, a correctly configured random uuid is appropriate). The app must validate the value of the parameter when returning to the redirect URL and must ensure that it is securely tied to the user's current session (for example, by relating the value of this parameter to a session identifier issued by the application). The app must limit the permissions, the scope and the requested period of time to the minimum necessary.

- **aud**: URL of the resource server of the EHR from which the app wishes to retrieve FHIR data. This parameter prevents a genuine bearer token from being leaked to a counterfeit resource server. In the case of a launch flow through the EHR, this value is the same as the **iss** value of the launch.

For example:

If an app needs demographic data and results (Observations) for a patient, and also wants information about the currently logged in user, the app can request in the parameter **scope**:

- patient/Patient.rs -> allows reading data for the current patient (Patient resource only)
- patient/Observation.rs -> allows reading the results for that patient
- openid fhirUser -> allows obtaining information for the user currently logged in

If the application is launched from an EHR: the app adds the value **launch** to the **scope** described above, and the launch id is placed in the **launch parameter** (**launch = {launch id}**), echoing the value it received from the EHR to be associated with the EHR context of this launch notification.

If the application is launched independently (standalone): it will not have a launch id at this time. These apps can declare launch context requirements by adding specific scopes to the authorization request: for example, **launch/patient** to indicate that the app needs a patient id. The EHR endpoint used to "authorize" will be responsible for

acquiring the necessary context (making it available to the application). For example, if the app needs patient context, the EHR can provide the end user with a patient selection list.

The most common type of SMART application is designed to run within the EHR, as in our case. This app must support the flow of executing the electronic medical record. To do that, we need to separate our logic into two pages:

- **launch.html**: the EHR will issue a call to initiate the authorization process.
- **index.html**: after a successful authorization, the EHR will redirect us there. This is where the actual application is initialized.

In launch.html we include the authorization.



Example H-4: Script for launch.html

```
<html>
<head>
  <meta charset="UTF-8" />
  <title>Launch My APP</title>
  <script src="https://cdn.jsdelivr.net/npm/fhirclient/build/fhir-client.js"></script>
</head>
<body>
  <script>

    FHIR.oauth2.authorize({

      // The client_id that you should have obtained after registering a client at
      // the EHR.
      clientId: "my_web_app",

      // The scopes that you request from the EHR. In this case we want to:
      // launch - Get the launch context
      // openid & fhirUser - Get the current user
      // patient/*.read - Read patient data
      scope: "launch openid fhirUser patient/*.read",

      // Typically, if your redirectUri points to the root of the current directory
      // (where the launchUri is), you can omit this option because the default value is
      // ".". However, some servers do not support directory indexes so "." and "/"
      // will not automatically map to the "index.html" file in that directory.
      redirectUri: "index.html"
    });
    debugger;
  </script>
</body>
</html>
```

Note that you cannot simply open this in the browser to launch the application. The EHR will open this page and pass some parameters. Without those parameters, the launch will not work.

In index.html, after the application launches, we will be redirected to the EHR, where we may have to authorize the launch and select a patient (all depending on the “scopes” we have requested). Once that is completed, the EHR will redirect to our site

again using the `redirectUri` option provided above. Once we are there, we have to complete the authorization flow (which is actually resolved by the JS library). From that point, we can start making requests to the FHIR server and implement our business logic. In our project we select the patient and retrieve the `medicationRequest` resources for this patient.

Step 2: Evaluate the authorization request

The EHR evaluates the authorization request, requesting an input from the end user

The authorization decision depends on the EHR's authorization server, which can request authorization from the end user. The EHR authorization server will impose access rules based on local policies and, optionally, the direct input from the end user.

The EHR decides whether to grant or deny access based on the `client_id`, the user currently logged into the EHR, the configured policy and perhaps the user's direct input. This decision is communicated to the application when the EHR authorization server returns an authorization code (or, if access is denied, an error response).

Authorization codes are short-lived, usually expiring in about a minute. The code is sent when the EHR authorization server causes the browser to navigate to the application's `redirect_uri`, with the following parameters in the URL:

- **code:** The authorization code generated by the authorization server. The code must expire shortly after it is issued to mitigate the risk of leaks.
- **state:** The exact value received from the app.

For example:

`https://app-after-auth?code=123abc&state=98wrghuwuogerg97`

Step 3: Exchanging the authorization code for the access token

At this point, the application exchanges the authorization code for the access token.

After obtaining an authorization code, the app exchanges this code for an access token through an HTTP POST call to the endpoint token URL of the EHR authorization server, using content-type *application/x-www-form-urlencoded*.

For public apps, authentication is not possible (and therefore not necessary), since a client with no secret cannot prove its identity when it issues a call. (The system can still be secure from end to end because the client comes from a specific endpoint, known and protected by the https protocol and enforced by the redirect URL.)

For confidential apps, an Authorization header is required that uses HTTP Basic authentication, where the username is the application's **client_id** and the password is the app's **client_secret**.

For example: If the **client_id** is "my-app" and the **client_secret** is "my-app-secret-123," the header uses the value *B64Encode ("my-app: my-app-secret-123")*, which is converted to *bXktYXBwOm15LWFwcC1zZWNyZXQtMTIz*. This gives the app the authorization token for "Basic Auth":

GET Header:

Authorization: Basic bXktYXBwOm15LWFwcC1zZWNyZXQtMTIz

The parameters to be used when the application exchanges the authorization code it was given for an access token:

- **grant_type**: fixed value: *authorization_code*
- **code**: code that the app received from the authorization server
- **redirect_uri**: The same *redirect_uri* used in the initial authorization request
- **client_id**: required for public applications. Skip for confidential apps.

Example H-5: Authorization Request

```
POST /token HTTP/1.1
Host: ehr
Authorization: Basic bXktYXBwOm15LWFwcC1zZWNyZXQtMTIz
Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code&
code=123abc&
redirect_uri=https://app-after-auth
```

The EHR authorization server must return a JSON object that includes an access token or a message indicating that the authorization request has been denied. The JSON structure includes the following parameters:

- **access_token**: the access token issued by the authorization server
- **token_type**: fixed value: bearer
- **expires_in**: useful life of the access token expressed in seconds; after that time the token must not be accepted by the FHIR resource server.
- **scope**: scope of authorized access. Keep in mind that this may be different from the scopes requested by the application.
- **id_token**: authenticated patient identity and user details, if requested. Optional parameter
- **refresh_token**: token that can be used to obtain a new access token, using the same or a subset of the original authorization permissions. Optional parameter

In addition, if the application was launched from the patient's context, parameters can be included to communicate the context values. For example, a parameter such as "patient": "123" would indicate the FHIR resource that is obtained through the FHIR API call `https://{fhir base url}/Patient/123`.

The parameters are included in the body of the HTTP response.

The access token is a string of characters. It is essentially a private message that the authorization server passes to the FHIR resource server, telling the FHIR server that the "message bearer" has been authorized to access the specified resources. The definition of the format and content of the access token depends on the organization that issues it and retains the requested resource.

The response from the authorization server must include in the HTTP fields of response header "Cache-Control" with a value of "no-store," as well as "Pragma" with a value of "no-cache."

The EHR authorization server decides what value of **expires_in** to assign to an access token and whether a refresh token must be issued along with the access token.

If the application receives a refresh token along with the access token, it can exchange this refresh token for a new access token when the current access token expires. A refresh token must be linked to the same `client_id` and must contain the same set or a subset of authorized claims for the access token with which it is associated.

Applications must store tokens only in application-specific storage locations, not in locations that are recognizable throughout the system. Access tokens must have a valid useful life of no more than one hour. Confidential clients can receive tokens of longer duration than public clients.

A wide range of threats to access tokens can be mitigated by signing the token digitally or using a Message Authentication Code (MAC) instead. Alternatively, an access token may contain a reference to the authorization information, instead of encoding the information directly in the token.

To be effective, such references must be unfeasible for an attacker to guess. The use of a reference may require additional interaction between the resource server and the authorization server.

Example H-6: Authorization Response

```
▼ Client 1
  ▶ encounter: {read: f}
  ▶ environment: BrowserAdapter {_url: URL, _storage: Storage, security: {...}, options: {...}}
  ▶ patient: {read: f, request: f}
  ▼ state:
    authorizeUri: "https://launch.smarthealthit.org/v/r4/auth/authorize"
    clientId: "my_web_app"
    codeChallenge: "_u8sU93km1MmNKIvhhUAoaxpuxfcTfMen7ANhV4ok0Y"
    ▶ codeChallengeMethods: ['S256']
    codeVerifier: "_wih6P34AhsDZxGz6UUNRvGTMpQdY0JAH5oYwG4s9PKa6qft73TmTy6CcuV11ANpNKuLDmLmYXIJqs7KQTnLN8VFYIyFTE5Nw5wkLVVUbPcJc"
    expiresAt: 1678973725
    key: "kUSuRdmvGGjfZBE7"
    redirectUri: "http://localhost/smartonfhir/index.html"
    registrationUri: ""
    scope: "launch openid fhirUser patient/*.read"
    serverUrl: "https://launch.smarthealthit.org/v/r4/fhir"
  ▼ tokenResponse:
    access_token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzY29wZSI6ImxhdW5jaCBvcGVuakQgZmhpclVzZXIgcGF0akVudC8qLnJlYWQilCjB2"
    encounter: "0f3e95c6-d47c-4fc7-8db6-660344e9c764"
    expires_in: 3600
    id_token: "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJwcm9maWx1IjoiaUJhY3RpdGlvdmVzL2U0NDNhYzU4LTlY2U0tNDM4NS04ZDU1LTc3NWpYj"
    need_patient_banner: true
    patient: "59bc5dbb-1957-47ef-b6b5-25952e6bfdab"
    scope: "launch openid fhirUser patient/*.read"
    smart_style_url: "https://launch.smarthealthit.org/smart-style.json"
    token_type: "Bearer"
    ▶ [[Prototype]]: Object
    tokenUri: "https://launch.smarthealthit.org/v/r4/auth/token"
```

At this point, the authorization flow is complete.



Step 4: Access to clinical data through the FHIR API

The application accesses clinical data through the FHIR API.

With a valid access token, the application can access protected EHR data through a FHIR API call to the endpoint of the EHR's FHIR server. The request includes an authorization header that presents the **access_token** as a "Bearer" token:

Authorization: Bearer {{access_token}}

Where {{access_token}} is replaced with the current value of the token that was obtained in the previous step.

For example:

Example H-7: Requesting Resources with Bearer Token

GET https://{fhir base url}/Patient/ 59bc5dbb-1957-47ef-b6b5-25952e6bfdab

Authorization: Bearer i8hweunweunweofiwwwoijewiwe

Example H-8: Response From the Server

```
{
  "resourceType": "Patient",
  "id": "59bc5dbb-1957-47ef-b6b5-25952e6bfdab",
  "meta": {
    "versionId": "3",
    "lastUpdated": "2021-04-06T03:01:44.046-04:00",
    "tag": [
      {
        "system": "https://smarthealthit.org/tags",
        "code": "synthea-5-2019"
      }
    ]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns='http://www.w3.org/1999/xhtml'>Generated by <a href='\"https://github.com/syntheahealth/synthea\">Synthea</a>.Version identifier: v2.4.0-100-g26a4b936\\n . Person seed: -3180320877229169863 Population seed: 1559319163074</div>"
  },
  "extension": [
    {
      "url": "http://hl7.org/fhir/us/core/StructureDefinition/us-core-race",
      "extension": [
        {
          "url": "ombCategory",
          "valueCoding": {
            "system": "urn:oid:2.16.840.1.113883.6.238",
            "code": "2106-3",
            "display": "White"
          }
        }
      ]
    },
    {
      "url": "text",
      "valueString": "White"
    }
  ]
},
{
  "url": "http://hl7.org/fhir/us/core/StructureDefinition/us-core-ethnicity",
  "extension": [
```

```

    {
      "url": "ombCategory",
      "valueCoding": {
        "system": "urn:oid:2.16.840.1.113883.6.238",
        "code": "2186-5",
        "display": "Not Hispanic or Latino"
      }
    },
    {
      "url": "text",
      "valueString": "Not Hispanic or Latino"
    }
  ]
},
{
  "url": "http://hl7.org/fhir/StructureDefinition/patient-mothersMaidenName",
  "valueString": "Sherise Bernier"
},
{
  "url": "http://hl7.org/fhir/us/core/StructureDefinition/us-core-birthsex",
  "valueCode": "M"
},
{
  "url": "http://hl7.org/fhir/StructureDefinition/patient-birthPlace",
  "valueAddress": {
    "city": "Turin",
    "state": "Piedmont",
    "country": "IT"
  }
},
{
  "url": "http://synthetichealth.github.io/synthea/disability-adjusted-life-years",
  "valueDecimal": 0
},
{
  "url": "http://synthetichealth.github.io/synthea/quality-adjusted-life-years",
  "valueDecimal": 17
}
],
"identifier": [
  {
    "system": "https://github.com/synthetichealth/synthea",
    "value": "dc9335d0-bbdd-4120-8ae9-baa6604343b6"
  },
  {
    "type": {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
          "code": "MR",
          "display": "Medical Record Number"
        }
      ]
    },
    "text": "Medical Record Number"
  },
  {
    "system": "http://hospital.smarthealthit.org",
    "value": "dc9335d0-bbdd-4120-8ae9-baa6604343b6"
  },
  {
    "type": {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
          "code": "SS",
          "display": "Social Security Number"
        }
      ]
    },
    "text": "Social Security Number"
  }
]

```

```

    },
    "system": "http://hl7.org/fhir/sid/us-ssn",
    "value": "999-87-6016"
  },
  {
    "type": {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
          "code": "DL",
          "display": "Driver's License"
        }
      ],
      "text": "Driver's License"
    },
    "system": "urn:oid:2.16.840.1.113883.4.3.25",
    "value": "S99971521"
  }
],
"name": [
  {
    "use": "official",
    "family": "Armstrong",
    "given": [
      "Alberto"
    ],
    "prefix": [
      "Mr."
    ]
  }
],
"telecom": [
  {
    "system": "phone",
    "value": "555-393-2687",
    "use": "home"
  }
],
"gender": "male",
"birthDate": "2002-12-16",
"address": [
  {
    "extension": [
      {
        "url": "http://hl7.org/fhir/StructureDefinition/geolocation",
        "extension": [
          {
            "url": "latitude",
            "valueDecimal": 42.673909
          },
          {
            "url": "longitude",
            "valueDecimal": -71.091334
          }
        ]
      }
    ]
  }
],
"line": [
  "201 Lueilwitz Manor Unit 62"
],
"city": "North Andover",
"state": "Massachusetts",
"postalCode": "01845",
"country": "US"
}
],
"maritalStatus": {
  "coding": [

```

```

    {
      "system": "http://terminology.hl7.org/CodeSystem/v3-MaritalStatus",
      "code": "S",
      "display": "Never Married"
    }
  ],
  "text": "Never Married"
},
"multipleBirthBoolean": false,
"communication": [
  {
    "language": {
      "coding": [
        {
          "system": "urn:ietf:bcp:47",
          "code": "it",
          "display": "Italian"
        }
      ],
      "text": "Italian"
    }
  }
]
}

```

With this response, the application knows which patient is in context and has an OAuth2 Bearer access token that can be used to fetch clinical data.

The resource server must validate the access token and ensure that it has not expired and that its scope covers the requested resource. It also validates that the parameter **aud** associated with the authorization matches the endpoint of the FHIR resource server.

Occasionally, an application may receive a FHIR resource that contains a "reference" to a resource hosted on a different resource server. The application should not blindly follow such references and send it along with its access token, since the token may be subject to possible theft. The application must ignore the reference or initiate a new request for access to that resource.

From this point on, the functionality is controlled by the SMART application, which will have all the logic of what is expected of it, whether that is graphing percentiles, showing a trend graph, displaying a vascular risk calculator or, as in our case, providing a prescription list.

Step 5: Updating credentials

Later ... The application uses a refresh token to obtain a new access token.

Refresh tokens are issued to allow sessions to last longer than the validity period of an access token. The application can use the **expires_in** field of the token response to determine when its access token will expire. An application with "online access" can continue to obtain new access tokens as long as the end user remains online. Applications with "offline access" can continue to obtain new access tokens without the user engaging interactively in cases where an application must have long-term access that extends beyond the time a user is still interacting with the client.

The application requests a refresh token in its authorization request through the scope **online_access** or **offline_access**. A server can decide what types of clients (public or confidential) are eligible for offline access and can receive a refresh token. If granted, the EHR provides a **refresh_token** in the token response. After the current access token expires, the application requests a new access token by providing its refresh token to the EHR token endpoint.

An HTTP POST transaction is made to the URL of the EHR authorization server token, with content-type *application/x-www-form-urlencoded*. The decision on how long the refresh token lasts is determined by a mechanism chosen by the server. For clients with online access, the goal is to ensure that the user is still online.

The following request parameters are defined:

- **grant_type**: fixed value: **refresh_token**
- **refresh_token**: the refresh token of a prior authorization response
- **scope**: the requested access scopes. If present, this value must be a strict subset of the scopes granted in the original release (no new permissions can be obtained at the time of update). A missing value indicates a request for the same scopes granted in the original release.

Example H-9: Updating credentials (refresh token request)

```
POST /token HTTP/1.1
Host: ehr
Authorization: Basic bXktYXBwOm15LWFwcC1zZWNYZXQtMTIz
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
refresh_token=a47txjiipgxkvohibvsm
```

The answer is a JSON object that contains a new access token, with the following fields:

- **access_token**: new access token issued by the authorization server
- **token_type**: fixed value: bearer
- **expires_in**: the access token's useful life, expressed in seconds. For example, the value 3600 denotes that the access token will expire in one hour from the moment the response was generated.
- **scope**: scope of authorized access. Keep in mind that this will be the same as the scope of the original access token, and may be different from the areas requested by the application.
- **refresh_token**: the refresh token issued by the authorization server. If present, the application must discard any previous refresh_token associated with this release, replacing it with this new value.

Example H-10: Updating credentials (refresh token response)

```
{
  "access_token": "m7rt6i7s9nuxkji8vsx",
  "token_type": "bearer",
  "expires_in": 3600,
  "scope": "patient/Observation.read patient/Patient.read",
  "refresh_token": "tGzv3JOkF0XG5Qx2TIKWIA"
}
```

User interface applications

Our focus is on user-oriented SMART on FHIR applications. We create these test applications as HTML5 web applications that run on any browser or mobile device.

The platform also supports native applications on iOS, Android and desktop. Web applications can be integrated into EHR systems in different ways: for web-based EHR, by adding an iframe to the web interface; for native client EHR, adding a browser widget or launching an external browser; and as we saw, when applications are launched from an existing EHR session, SMART on FHIR defines a protocol to communicate the EHR context (i.e., the patient, the encounter and the identity of the user, for example).

Background applications and services

Background applications and services can access the same FHIR APIs, obtaining authorization tokens through fully automated OAuth2 without the need for an end user. This functionality uses the OAuth2 client credentials.

Some examples of uses may be quality metrics, event reminders or alerts for critical values in laboratory results.

Recommendations

Everything sounds quite reasonable from a technical point of view. However, the problem is with the resource's content. **FHIR resources must be constrained.**

The FHIR base implementation is generic and impose very few constraints on resources. It makes almost all data optional and mostly 'example' vocabulary binding.

For example, the base MedicationRequest resource leaves all fields optional, including prescription identification, date and basic medication data. In addition, FHIR does not specify the coding of medications, diseases, laboratory results, procedures or allergies. This work must be established by defining profiles for each of the resources used by the app.

As a recommended good practice, we must be very careful when creating new profiles and validate resources against the profile to make sure they are conformant. One risk is that various organizations can generate incompatible profiles that lead to fragmented semantics and prevent applications from being interchangeable.

Thus, most implementations of SMART-on-FHIR leverage US CORE profiles to constrain resource content.

CDS Hooks

The CDS Hooks specification describes a RESTful API and interactions to integrate a clinical decision support system (CDS) with CDS Clients (usually electronic health records (EHR) or other health information systems).

In CDS Hooks, a CDS Service is a service that provides specific recommendations for the context where it is called. All data exchanged through the RESTful API is sent and received in JSON format, using the HTTPS protocol.

Today the standard (version 2.0) is in continuous development and the specification is sure to change in subsequent versions; however, the approach is highly interesting and deserves a look.

What is a hook

In this context, a hook is responsible for expanding the functionality of a CDS Client (for example, an electronic medical record). It intercepts the events that occur in the Client's workflow to call the corresponding CDS Services.

The CDS Service API using CDS Hooks supports synchronous calls, which return information and suggestions. Also, it supports the response of the CDS Service to launch a user-oriented SMART on FHIR application when the CDS requires additional interaction.

How does it work?

User activity within the CDS Client activates the hooks that invoke the corresponding CDS Services in real time. Some hooks that are already defined by the specification are:

- patient-view: Activated when opening a new patient record.
- appointment-book: This hook is invoked when the user is scheduling one or more future encounters/visits for the patient.
- encounter-discharge: This hook is invoked when the user is performing the discharge process for an encounter where the notion of 'discharge' is relevant - typically an inpatient encounter.
- encounter-start: This hook is invoked when the user is initiating a new encounter.
- order-select: The order-select hook fires when a clinician selects one or more orders to place for a patient, (including orders for medications, procedures, labs and other orders).
- order-sign: The order-sign hook fires when a clinician is ready to sign one or more orders for a patient.

The set of defined hooks is not a closed set; anyone can define new hooks adapted to their use cases and propose them to the community to be included in a new version of the specification.

The name of the hook must clearly describe the activity or event that it represents within the workflow of the CDS Client. Names are unique, so when creating a new one, care must be taken to ensure that it does not conflict with existing ones. It is necessary to include as much detail as possible in your specification to minimize any ambiguity or

confusion among implementers. The name of the hook must start with the subject (noun) of the hook and then the activity (verb), as we can see above in the hooks defined by the specification.



In our project, the hook used is “order-select,” which is used when a practitioner selects one medication to check the price.

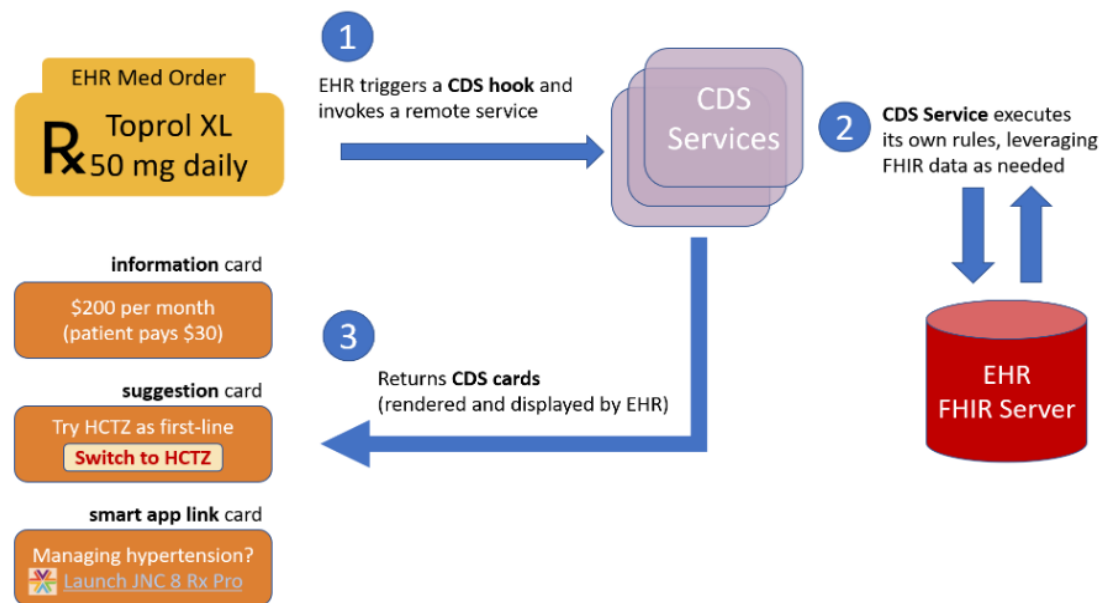
When a triggering activity occurs, the CDS Client notifies each CDS Service registered for the activity. For each activity, one or more services can be called.

Each service gets details about the context (*context* parameter of the hook) of the CDS Client in addition to specific data (via the parameter *prefetch*) that the service requires to make decisions. The CDS Service must provide almost real-time feedback (on the order of 500 ms) on the event that triggered it. Sensitive and confidential information is exchanged between the Client and the CDS Services, so security is extremely important and should be kept in mind at all times.

Basic Components

The basic components of CDS Hooks are:

- **CDS Client** (or EHR): An electronic health record or other clinical information system that consumes support for the clinical decision in the form of services. This Client may provide the CDS Service with the authorization to connect to its FHIR resource server. (*Point 1 in the graphic*)
- **Hook**: A defined point within the CDS Client workflow with well-known contextual information that is provided as part of the request to the CDS Service. (*Point 1 in the graphic*)
- **CDS Service**: A clinical decision-making support service (CDS) that accepts requests containing patient information and provides responses (information, suggestions or the possibility of launching a SMART on FHIR app) so that the clinician can make a decision. (*Point 2 in the graphic*)
- **Card**: The CDS Service returns recommendations or suggestions in the form of cards. These cards in general are presented by the CDS Client to the end user. (*Point 3 in the graphic*)



Discovery of CDS services

An endpoint is the interface through which external systems can send and receive messages. The developers of CDS Services must provide a stable endpoint to allow the CDS Client to discover the available services, including information on the purpose of each one, when it should be invoked and the data it needs.

When invoking the following URL, the list of available services must be obtained:

GET {URLbase}/cds-services.

For example, if the base URL is <https://example.com>, the Client can invoke <https://example.com/cds-services> and discover all available services.

Each CDS Service is detailed by the following attributes:

- **hook**: The hook used to invoke the service. *string*. **REQUIRED**.
- **title**: The friendly name of the service. *string*. **RECOMMENDED**.
- **description**: The description of the service. *string*. **REQUIRED**.
- **id**: The identifier of the service used to invoke it using the URL {URLbase} / cds-services / {ServiceID}. *string*. **REQUIRED**.
- **prefetch**: An object containing key/value pairs of FHIR queries that the Service requests from the Client (that is invoking it) to obtain its result in advance and provide it in each call to this service. The key is a string that describes the type of data requested and the value is a string that represents a FHIR query. *object*. **OPTIONAL**.
- **usageRequirements**: Human-friendly description of any preconditions for the use of this CDS Service. (string **OPTIONAL**)

Example H-11: Service Discovery

```

{
  "services": [
    {
      "hook": "patient-view",
      "title": "CDS Service Example",
    }
  ]
}
  
```

```

    "description": "An example of a CDS Service that returns a set of cards",
    "id": "static-patient-greeter",
    "prefetch": {
      "patientToGreet": "Patient/{{context.patientId}}"
    }
  },
  {
    "hook": "medication-prescribe",
    "title": "Medication Echo CDS Service",
    "description": "An example of a CDS Service that simply echos the medication being prescribed",
    "id": "medication-echo",
    "prefetch": {
      "patient": "Patient/{{context.patientId}}",
      "medications": "MedicationRequest?patient={{context.patientId}}"
    }
  }
]
}

```



If we consult the CDSS service used by our SMART application, doing a GET <https://fhir-org-cds-services.appspot.com/cds-services> we will obtain:

```

{
  "services": [
    {
      "id": "cms-price-check",
      "title": "CMS Pricing Service",
      "description": "Determine if an authored prescription has a cheaper alternative to switch to and display pricing",
      "hook": "order-select"
    },
    {
      "id": "pama-imaging",
      "title": "Pama Imaging Connectathon 2019 Scenarios",
      "description": "Produce an appropriateness score according to scenario inputs.",
      "hook": "order-select"
    },
    {
      "id": "patient-greeting",
      "title": "Patient greeting",
      "description": "Display which patient the user is currently working with",
      "hook": "patient-view",
      "prefetch": {
        "patient": "Patient/{{context.patientId}}"
      }
    }
  ]
}

```


A hook can encompass several CDS Services, which are individually identified by the service identifier (id).



The hook that we use in the project, “order-select”, can group several relevant services. In this case the service informs the medication price and gives some recommendations in case a cheaper option is found. By discovering the available CDS Services, a business analyst will determine where they can be called based on the workflow determined by the CDS Client.

Calling a CDS service

The Client calls a CDS Service by sending the data required by the service to the URL that identifies it. The URL is constructed, as mentioned above, using the base URL and the particular identifier of the service. For example:

POST {baseUrl}/cds-services/{service.id}

The call includes the following fields:

- **hook:** The hook that activated the call to this CDS Service. *string*. REQUIRED.
- **hookInstance:** A UUID that represents this particular call. When working on a Client, a user can perform multiple actions, one after the other or simultaneously. The UUID allows the CDS Service to unequivocally identify each invocation of the hook. The call to the hook is globally unique and should contain enough entropy not to be guessed. *string*. REQUIRED.
- **fhirServer:** If a FHIR Server is used, its base URL must be sent. The Server will always be accessed using the HTTPS protocol. *URL* OPTIONAL. *(If information is provided in the fhirAuthorization attribute, this field becomes REQUIRED.)*
- **fhirAuthorization:** A structure that contains an OAuth 2.0 access token that gives the CDS Service access to the Client's FHIR Resource Server, along with other complementary information related to the token. This token will be sent with each call made to the FHIR Server in search of resources. The CDS Client has control of creating the access token associated with the specific CDS Service, the user and the context of the invocation. That is, the CDS Service runs on behalf of a user; therefore access to the resources of the FHIR Server to which this Service can have access must be limited with the same restrictions and authorizations granted to the current user. The access token must have scope for the invoked CDS Service and the current user. *object*. OPTIONAL. This field contains:
 - **access_token:** the OAuth 2.0 access token that provides access to the FHIR server. *string*. REQUIRED.
 - **token_type:** fixed value “Bearer”. *string*. REQUIRED.
 - **expires_in:** represents the life of the token expressed in seconds. This life must be very short, since the access token must be treated as a transitory value by the CDS Service. *whole*. REQUIRED.

- **scope:** the scope provided by the access token to the CDS Service. This field follows the SMART on FHIR specification. *string*. REQUIRED.
 - **subject:** the OAuth 2.0 client identifier of the CDS Service as it is registered on the authorization server of the CDS Client. *string*. REQUIRED.
- **context:** FHIR contextual data that the CDS Service will need to make decisions. *object*. REQUIRED.
- **prefetch:** FHIR data that the Client had to recover prior to the Service call. The Client knows what data it has to recover by setting the **prefetch** field of the corresponding Service when the discovery is made. *object*. OPTIONAL.

Example H-12: Calling a CDS Service using patient-view hook

```
{
  "hookInstance" : "d1577c69-df6e-44ad-ba6d-3e05e953b2ea",
  "fhirServer" : "http://hooks.smarthealthit.org:9080",
  "hook" : "patient-view",
  "fhirAuthorization" : {
    "access_token" : "some-opaque-fhir-access-token",
    "token_type" : "Bearer",
    "expires_in" : 300,
    "scope" : "patient/Patient.rs patient/Observation.rs",
    "subject" : "cds-service4"
  },
  "context" : {
    "userId" : "Practitioner/example",
    "patientId" : "1288992",
    "encounterId" : "89284"
  },
  "prefetch" : {
    "patientToGreet" : {
      "resourceType" : "Patient",
      "gender" : "male",
      "birthDate" : "1925-12-23",
      "id" : "1288992",
      "active": true
    }
  }
}
```

The context data is relevant to all CDS services that are subscribed to the same hook. On the other hand, prefetched data are exclusive to each service and complement the context data.



In the case of our project, when prescribing a new medication to the patient, the maximum dose of opioids per day that the patient can take will be checked. To do this, a POST is done at <http://localhost:8084/cdss/cds-services/drug-max-dose> with the following data

Example H-13: order-select hook from our project

```
{
  "hookInstance": "6011fae5-225a-48c2-9fc6-5023424d8684",
  "hook": "order-select",
  "fhirserver": "https://launch.smarthealthit.org/v/r4/fhir",
  "context": {
    "userId": "Practitioner/e443ac58-8ece-4385-8d55-775c1b8f3a37",
    "patientId": "59bc5dbb-1957-47ef-b6b5-25952e6bfdab",
    "selections": ["MedicationRequest/f56be270-0ea1-4b05-af4d-b7cf2b151a82"],
    "draftOrders": {
      "resourceType": "Bundle",
      "entry": [{
        "resource": {
          "resourceType": "MedicationRequest",
          "id": "f56be270-0ea1-4b05-af4d-b7cf2b151a82",
          "meta": {
            "versionId": "3",
            "lastUpdated": "2021-04-06T03:11:26.109-04:00",
            "tag": [{
              "system": "https://smarthealthit.org/tags",
              "code": "synthea-5-2019"
            }]
          },
          "status": "stopped",
          "intent": "order",
          "medicationCodeableConcept": {
            "coding": [{
              "system": "http://www.nlm.nih.gov/research/umls/rxnorm",
              "code": "834061",
              "display": "Penicillin V Potassium 250 MG Oral Tablet"
            }],
            "text": "Penicillin V Potassium 250 MG Oral Tablet"
          },
          "subject": {
            "reference": "Patient/59bc5dbb-1957-47ef-b6b5-25952e6bfdab"
          },
          "encounter": {
            "reference": "Encounter/f9cfda88-0c08-405d-aec3-cb4381176c18"
          },
          "authoredOn": "2016-03-28T07:00:48+00:00",
          "requester": {
            "reference": "Practitioner/a1b28730-eedd-4efa-870e-a70af74fec46"
          },
          "reasonReference": [{
            "reference": "Condition/8d214a0e-73f7-492c-86cc-b09e8ffa4c4d"
          }]
        }
      }]
    }
  }
}
```



In this case the “order-select” hook specification is followed <https://cds-hooks.org/hooks/order-select/>

“hookInstance”: generates a UUID for this particular call to the CDSS service.

“fhirServer”: is the same URL that SMART uses.

“fhirAuthorization”: would be the access token to the FHIR server that the authorization server returns to us in step 3 of the SMART specification. In this case it is simulated, since we have not implemented the OAuth2 process for the SMART application for reasons of simplicity.

In the next sections we will address the fields "context" and "prefetch."

Context Data

Any workflow or user action within the CDS Client will naturally include contextual information (**context**) such as the current user and the patient. CDS Hooks allows each hook to define the information that is available in the context.

Because this standard is designed to be used within any CDS Client, this context may contain mandatory and optional data.

Only the data necessarily associated with the purpose of the hook should be represented in the context. All fields defined by the context must be described by the following attributes:

- **field:** the name of the field in the JSON object of the context. Hook authors should name their context fields to be consistent with other existing hooks when they refer to the same field.
- **optionality:** a string that indicates REQUIRED or OPTIONAL.
- **prefetch token:** a string that indicates YES or NO, referring to whether this field can be used in the templates for prefetch of data by the Client.
- **type:** the type of the field in the context JSON object, expressed as a JSON type or the name of a FHIR resource type. Valid types for this attribute are: Boolean, string, number, object, array or the name of a FHIR resource type. When a field can be of many types, their names must be separated with | (pipes).
- **description:** a functional description of the context value. If this value can change according to the version of FHIR that is used, this field should describe the value of each compatible version of FHIR.

Following the example of the order-select hook, the context specification would look like this:

userId	REQUIRED	Yes	<i>string</i>	The id of the current user. For this hook, the user is expected to be of type <u>Practitioner</u> or <u>PractitionerRole</u> . For example, PractitionerRole/123 or Practitioner/abc.
patientId	REQUIRED	Yes	<i>string</i>	The FHIR Patient.id of the current patient in context
encounterId	OPTIONAL	Yes	<i>string</i>	The FHIR Encounter.id of the current encounter in context
selections	REQUIRED	No	<i>array</i>	The FHIR id of the newly selected order(s). The selections field references FHIR resources in the draftOrders Bundle. For example, MedicationRequest/103.
draftOrders	REQUIRED	No	<i>object</i>	DSTU2 - FHIR Bundle of MedicationOrder, DiagnosticOrder, DeviceUseRequest, ReferralRequest, ProcedureRequest, NutritionOrder, VisionPrescription with <i>draft</i> status STU3 - FHIR Bundle of MedicationRequest, ReferralRequest, ProcedureRequest, NutritionOrder, VisionPrescription with <i>draft</i> status R4 - FHIR Bundle of DeviceRequest, MedicationRequest, NutritionOrder, ServiceRequest, VisionPrescription with <i>draft</i> status

The fields of the context that may contain multiple FHIR resources should be defined as a FHIR Bundle, rather than an array of FHIR resources.

Many times, the context is completed with data in progress or in memory that may not be available from the client FHIR server.

All context FHIR resources must be based on the same version of FHIR.

The context fields are defined in the “order-select” specification. The mandatory fields are: userId (in this case the logged-in doctor), patientId, selections (the orders that have just been created) and draftOrders



In this project, we send the MedicationRequest and draftsOrders (a Bundle with the MedicationRequest). The encounterId is optional and in our example we are not sending it.

The userId and patientId values are part of the context of the application. That is, they are provided by the Authorization server to the SMART application in step 3 of the specification along with the access token.

Prefetched Data

Each CDS Service will require specific FHIR resources to provide the recommendations requested by the CDS Client. If real-world performance were not a problem, a CDS Client could invoke a CDS Service by sending only context data (such as current user and patient identifiers), and the CDS Service could request authorization to access resources through the Client FHIR server when necessary. Since CDS Services have to respond quickly (on the order of 500 ms), this specification

defines a process to allow a CDS Service to request and obtain FHIR resources efficiently.

Two optional methods are provided:

- In the first, the CDS Service can obtain the FHIR resources through the data prefetched by the CDS Client. The FHIR resources requested in the CDS Service description are passed as key-value pairs, where each key matches a key described in the Service description and each value is a FHIR resource. In the case of searches, this resource can be a Bundle. If the data must be prefetched, the CDS Service registers a set of "prefetch templates" with the CDS Client.
- The second method allows the CDS Service to retrieve FHIR resources for itself, but more efficiently than if it needed to request and obtain its own authorization to operate with the Client's FHIR Server. If the CDS Client decides that the CDS Service will fetch its own FHIR resources, the Client obtains and sends a token directly to the CDS Service so that it can execute FHIR API calls against its FHIR server and obtain the necessary resources. Some CDS Clients may pass along prefetched data, along with a token for the CDS Service to use if it requires additional resources.

Each CDS Client has to decide which approach it will use. A combination of both approaches can also be selected, always taking into account the performance and security risks associated with each approach. In addition, each CDS Client will decide which FHIR resources to authorize and prefetch, based on the request for "pre-capture" in the description of the CDS Service and the evaluation of the Service provider as "minimum necessary" for the proper functioning of the support system.

When context data is related to a FHIR resource, it is important not to combine context and previously retrieved data (prefetch).

For example, in the hook described above (patient-view), the identification of the patient whose medical history is being opened is included, not the patient's full FHIR resource. In this case, the patient's FHIR identifier is appropriate since the CDS Services may not be interested in the details of the patient resource, but in other data related to this patient. Or, in other scenarios a CDS Service might need the complete patient resource. Therefore, including the entire patient resource in the context would be unnecessary.

Consider another hook for when a new patient is being registered. In this case, it is likely that the context contains the complete Patient FHIR resource for the patient being registered, since it is possible that the patient is not yet registered in the FHIR Server of the CDS Client and CDS Services using this hook would be interested primarily in the details of the patient being registered.

Example H-14: Prefetch Example

```
"prefetch" : {  
  "patientToGreet" : "Patient / {{context.patientId}}"  
}
```

Prefetch template

This template is a FHIR search or read request and describes relevant data needed by the CDS Service. For example, the following template can be used to obtain results (Observation) of hemoglobin A1c for a particular Patient:

Observation?patient={{context.patientId}}&code=4548-4&_count=1&sort:desc=date

These templates may include references to context using "prefetch tokens," for example **{{context.patientId}}**

The prefetch field of a CDS Service description defines a set of templates for that Service, providing a "prefetch key" for each.

Example H-15: Prefetch Template

```
{  
  "prefetch": {  
    "patient": "Patient/{{context.patientId}}",  
    "hemoglobin-a1c": "Observation?patient={{context.patientId}}&code=4548-4&_count=1&sort:desc=date",  
    "user": "{{context.userId}}"  
  }  
}
```

The prefetch key for the template is **"hemoglobin-a1c"**.

A CDS Client may choose to honor some or all of the desired prefetch templates, and is free to choose the most appropriate source for these data. For example:

- The CDS Client may have some of the desired data previously in memory, thus eliminating the need for any network call.
- The CDS Client may process an efficient set of templates from multiple CDS Services, thus minimizing the number of calls.
- The CDS Client may satisfy some of the desired prefetched templates through some internal service or even its own FHIR Server.

The CDS Client must deny access to the requested resource if the user is outside the authorized scope.

As part of preparing the request, a CDS Client processes each template it intends to satisfy by replacing the prefetch tokens to build a relative request URL to the FHIR Server.

Regardless of how the CDS Client satisfies the templates (if at all), the prefetched data provided to the CDS Service must be equivalent to the data that the CDS Service would receive if it made its own call to the Client's FHIR Server using the parameterized template.

The CDS Services should receive only the data they requested to be previously recovered by the Client and for which they are authorized recipients.

Prefetch tokens

A prefetch token marks a place ({{{}}}) in a template that is replaced with a value from the context of the hook to build the FHIR URL used to request the data.

Only fields at the root level (first level) with a primitive value (string, number or Boolean) within the context field are eligible to be used as prefetch tokens. For example, `{{context.medication.id}}` is not a valid token because it attempts to access the identifier of the medication field (which is not a field at the root level).

For example: a particular CDS Service could recommend a guide based on the patient's conditions when the electronic medical record is opened. The FHIR query to retrieve these conditions could be `Condition?patient = 123`. To express this as a template, the CDS Service must express the patient's FHIR identifier as a token so that the CDS Client can replace the token with the appropriate value.

```
"context" : {  
  "patientId": "123"  
}
```

The name of the token is `{{context.patientId}}`. The template in this case would be `Condition?patient={{context.patientId}}`.

A prefetch template may include any of the following prefetch tokens:

Token	Description
<code>{{userPractitionerId}}</code>	FHIR id of the Practitioner resource corresponding to the current user.
<code>{{userPractitionerRoleId}}</code>	FHIR id of the PractitionerRole resource corresponding to the current user.
<code>{{userPatientId}}</code>	FHIR id of the Patient resource corresponding to the current user.
<code>{{userRelatedPersonId}}</code>	FHIR id of the RelatedPerson resource corresponding to the current user.

Example H-16a: A context with a FHIR patient resource

```
"context" : {  
  "encounterId": "456",  
  "patient": {  
    "resourceType": "Patient",  
    "id": "123",  
    "active": true,  
    "name": [  
      {  
        "use": "official",  
        "family": "Watts",  
        "given": [  
          "Wade"  
        ]  
      }  
    ],  
    "gender": "male",  
    "birthDate": "2024-08-12"  
  }  
}
```

Only **encounterId** is eligible to be a token since it is a first level field and also has a primitive data type (character string) that can be placed into a FHIR query. If we wanted to use the patient identifier as a token, then we should make a change so that this identifier is in the first level. Then it would look like this

Example H-16b: A context with a FHIR patient resource and an encounterId

```
"context" : {  
  "patientId": "123",  
  "encounterId": "456",  
  "patient": {  
    "resourceType": "Patient",  
    "id": "123",  
    "active": true,  
    "name": [  
      {  
        "use": "official",  
        "family": "Watts",  
        "given": [  
          "Wade"  
        ]  
      }  
    ],  
    "gender": "male",  
    "birthDate": "2024-08-12"  
  }  
}
```

CDS Service Response

The response of each CDS Service occurs in the form of cards. Each card contains decision support provided by the CDS Service.

The cards can provide a combination of information (text for the user to read), suggested actions (which will be applied automatically if the user selects them) and links (to launch a SMART on FHIR application where the user can provide details or do anything else necessary to reach a decision).

An end user must be able to see these cards, one or more of each type, in the Client and must be able to interact with them.

HTTP Status Codes

Code	Description
200 OK	A successful response.
412 Precondition Failed	The CDS Service is unable to retrieve the necessary FHIR data to execute its decision support, either through a prefetch request or directly calling the FHIR server.

CDS Services MAY return other HTTP statuses, specifically 4xx and 5xx HTTP error codes.

HTTP Response

Field	Optionality	Type	Description
cards	REQUIRED	array of Cards	An array of Cards . Cards can provide a combination of information (for reading), suggested actions (to be applied if a user selects them), and links (to launch an app if the user selects them). The CDS Client decides how to display cards, but this specification recommends displaying suggestions using buttons, and links using underlined text.
systemActions	OPTIONAL	array of Actions	An array of Actions that the CDS Service proposes to auto-apply. Each action follows the schema of a card-based suggestion.action . The CDS Client decides whether to auto-apply actions.

The CDS Client has the power to decide how to display the cards, but using buttons for suggestions and underlined text for links is recommended. If there is no support for user decision-making, the CDS Service must return an empty list of cards. Each card is composed of the following attributes:

- **uuid**: Unique identifier of the card. MAY be used for auditing and logging cards and SHALL be included in any subsequent calls to the CDS service's feedback endpoint. String. OPTIONAL
- **summary**: A phrase (<140 characters) that summarizes what is inside the card to the user. *string*. **REQUIRED**.
- **detail**: Optionally you can include more detailed information for the user about the card's content. *string*. OPTIONAL.
- **indicator**: An indicator of the urgency/importance that the card conveys. Possible values are: info (information), warning and critical. The CDS Client can

use this field to help make display decisions such as order or color. *string*. **REQUIRED**.

- **source:** The source of the information displayed on the card, which guides the decision support. *object*. **REQUIRED**. The source contains the following attributes:
 - **label:** A short and readable label to show the source of the information displayed on the card. If a URL is also specified, this tag can be the text for the hyperlink. *string*. **REQUIRED**.
 - **url:** Optionally an absolute URL is added so that the user can click on the link and obtain more information about the organization or the data set that provided the information on the card. *URL*. **OPTIONAL**.
 - **icon:** An absolute URL can also be added to an icon that identifies the source of information on the card. The icon returned by the URL must be in PNG format, must be 100 x 100 pixels in size and must not include any transparent regions. *URL*. **OPTIONAL**.
 - **topic:** A topic describes the content of the card by providing a high-level categorization that can be useful for filtering, searching or ordered display of related cards in the CDS client's UI. This specification does not prescribe a standard set of topics. *Coding*. **OPTIONAL**
- **suggestions:** If the card suggests actions, a list of them should be included. The CDS Service may suggest a set of changes in the context of the current activity of the CDS Client. *array of suggestions*. **OPTIONAL**. Each suggestion is described with the following attributes:
 - **label:** A readable label to show the suggestion. The Client can use this label as the text of the button linked to this suggestion. *string*. **REQUIRED**.
 - **uuid:** A UUID is included that is a unique identifier for this suggestion. It will help us to monitor what happens with it. *string*. **OPTIONAL**.
 - **isRecommended:** When there are multiple suggestions, allows a service to indicate that a specific suggestion is recommended from all the available suggestions on the card. *Boolean*. **OPTIONAL**
 - **actions:** A list of actions will also be included. If a user selects a suggestion, all actions within it will be selected. *array*. **OPTIONAL**. Each action is described by the following attributes
 - **type:** The type of action being performed. Allowed values are *create, update, delete*. *string*. **REQUIRED**.
 - **description:** A readable description of the suggested action. It can be presented to the end user. *string*. **REQUIRED**.
 - **resource:** A resource. Depending on the type attribute, this field will be a new resource or the id of one that already exists. For a type *create*, this attribute contains a new FHIR resource. To

delete, this field will contain the identifier of the resource that will be deleted. For *update*, this field contains the updated resource in its entirety and not just the modified fields. *object*.
CONDITIONAL.

- **resourceId**: A relative reference to the relevant resource. SHOULD be provided when the type attribute is delete. *String*.
CONDITIONAL
- **selectionBehavior**: Describes the expected selection behavior of the suggestions contained in the card. Currently, the only value allowed is *at-most-one*, which indicates that the user can choose none or at most one of the suggestions. In future versions of the specification this behavior can be expanded. *string*.
CONDITIONAL.
- **overrideReasons**: Override reasons can be selected by the end user when overriding a card without taking the suggested recommendations. The CDS service MAY return a list of override reasons to the CDS client. If override reasons are present, the CDS Service MUST populate a display value for each reason's Coding. *Array of coding*.
OPTIONAL
- **links**: If the card suggests links, a list of them must be included. The CDS Service allows you to suggest a link to an application that the user might want to run to obtain additional information or to help guide a decision. *arrangement of links*.
OPTIONAL. Each link is described by the following attributes:
 - **label**: A readable tag to display this link. The CDS Client can render this tag as the underlined text of a clickable link. *string*. **REQUIRED**.
 - **url**: A URL to load when the user clicks on this link. *URL*. **REQUIRED**.
 - **type**: The type of the given URL. There are two possible values for this field: an absolute type and a smart type. If the type is smart, it is indicating that the URL launches a SMART on FHIR application and the CDS Client must ensure that all appropriate parameters are provided in the launch URL. *string*. **REQUIRED**.
 - **appContext**: Allows the CDS Service to share card information with a SMART on FHIR application that will be launched later. This attribute should only be utilized if the type of link is smart. The field and the value of this attribute will be sent to the SMART application as part of the response of the OAuth 2.0 access token, along with other parameters for launching the SMART application. This field could be escaped JSON, base64 encoded XML or even a simple string, as long as the SMART application can recognize it. *string*.
OPTIONAL

Example H-17: A CDS-Hooks card

```
"cards": [
  {
    "uuid": "db79a756-85b1-4b73-8375-69f42f6d6a0c",
    "summary": "Cost: $500.41. Save $426.47 with a generic medication.",
    "source": {
      "label": "CMS Public Use Files"
    },
    "indicator": "info",
    "suggestions": [
      {
        "label": "Change to generic",
        "uuid": "06ced259-3ce8-4f34-877e-11a1bc512fb4",
        "actions": [
          {
            "type": "create",
            "description": "Create a resource with the newly suggested medication",
            "resource": {
              "resourceType": "MedicationRequest",
              "id": "request-123",
              "status": "draft",
              "subject": {
                "reference": "Patient/b218cee9-019d-47a4-b161-e97c0fd6f736"
              },
              "authoredOn": "2023-03-16",
              "dosageInstruction": [
                {
                  "timing": {
                    "repeat": {
                      "frequency": 1,
                      "period": 1,
                      "periodUnit": "d"
                    }
                  },
                  "doseAndRate": [
                    {
                      "doseQuantity": {
                        "value": 1,
                        "system": "http://unitsofmeasure.org",
                        "code": "{pill}"
                      }
                    ]
                  }
                ]
              }
            },
            "medicationCodeableConcept": {
              "text": "12 HR Oxycodone Hydrochloride 80 MG Extended Release Oral Tablet",
              "coding": [
                {
                  "code": "1049599",
                  "system": "http://www.nlm.nih.gov/research/umls/rxnorm",
                  "display": "12 HR Oxycodone Hydrochloride 80 MG Extended Release Oral Tablet"
                }
              ]
            },
            "reasonCode": {
              "coding": [
                {
                  "system": "http://snomed.info/sct",
                  "code": "271737000",
                  "display": "Anemia (disorder)"
                }
              ],
              "text": "Anemia (disorder)"
            }
          }
        ]
      }
    ]
  }
]
```

```

    ]
  },
  "overrideReasons": [
    {
      "code": "patient-requested-brand",
      "system": "http://terminology.cds-hooks.org/CodeSystem/OverrideReasons",
      "display": "Patient Requested Brand Product"
    },
    {
      "code": "generic-drug-unavailable",
      "system": "http://terminology.cds-hooks.org/CodeSystem/OverrideReasons",
      "display": "Generic Drug Out of Stock or Unavailable"
    }
  ]
}
]
}
}

```



If there is no price, then the CDSS service will return an empty card array:

```

{
  "cards": []
}

```

Feedback

Once a CDS Hooks Service responds to a hook by returning a card, the service has no further interaction with the CDS Client. The acceptance of a suggestion or rejection of a card is valuable information to enable a service to improve its behavior towards the goal of the end-user having a positive and meaningful experience with the CDS. A feedback endpoint enables suggestion tracking & analytics. A CDS Service MAY support a feedback endpoint; a CDS Client SHOULD be capable of sending feedback.

Upon receiving a card, a user may accept its suggestions, ignore it entirely, or dismiss it with or without an override reason.

Typically, an end user may only accept (a suggestion), or override a card once; however, a card once ignored could later be acted upon. CDS Hooks does not specify the UI behavior of CDS Clients, including the persistence of cards. CDS Clients should faithfully report each of these distinct end-user interactions as feedback.

A CDS Client provides feedback by POSTing a JSON document. The feedback endpoint can be constructed from the CDS Service endpoint and a path segment of "feedback" as {baseUrl}/cds-services/{service.id}/feedback. The request to the feedback endpoint SHALL be an object containing an array.

Each Feedback SHALL be described by the following attributes.

- **card** : The card.uuid from the CDS Hooks response. Uniquely identifies the card. *String*. REQUIRED
- **outcome**: A value of accepted or overridden. *String*. REQUIRED

- **acceptedSuggestions** : An array of json objects identifying one or more of the user's AcceptedSuggestions. Required for accepted outcomes. Array. CONDITIONAL
- **overrideReason**: A json object capturing the override reason as a Coding as well as any comments entered by the user. OverrideReason. OPTIONAL
 - **reason** : The Coding object representing the override reason selected by the end user. Required if user selected an override reason from the list of reasons provided in the Card (instead of only leaving a userComment). Coding. CONDITIONAL
 - **userComment**: The CDS Client may enable the clinician to further explain why the card was rejected with free text. That user comment may be communicated to the CDS Service as a userComment *string* OPTIONAL
- **outcomeTimestamp**: ISO8601 representation of the date and time in Coordinated Universal Time (UTC) when action was taken on the card. *string* REQUIRED

Suggestion accepted

The CDS Client can inform the service when one or more suggestions were accepted by POSTing a simple JSON object. The CDS Client authenticates to the CDS service as described in [Trusting CDS Clients](#).

Upon the user accepting a suggestion (perhaps when she clicks a displayed label (e.g., button) from a "suggestion" card), the CDS Client informs the service by posting the card and suggestion uuids to the CDS Service's feedback endpoint with an outcome of accepted.

Card ignored

If the end-user doesn't interact with the CDS Service's card at all, the card is ignored. In this case, the CDS Client does not inform the CDS Service of the rejected guidance.

Overridden guidance

A CDS Client may enable the end user to override guidance without providing an explicit reason for doing so. The CDS Client can inform the service when a card was dismissed by specifying an outcome of overridden without providing an overrideReason. This may occur, for example, when the end user viewed the card and dismissed it without providing a reason why.

Extensions

The specification is not prescriptive about support for extensions. However, to support extensions, the specification reserves the name *extension* and will never define an element with that name, allowing implementations to use it to provide personalized information and behavior. The value of an extension element must be a pre-coordinated JSON object.

Example H-18: Supporting extensions

```
{
  "hookInstance" : "d1577c69-df6e-44ad-ba6d-3e05e953b2ea",
  "fhirServer" : "http://fhir.example.org:9080",
  "hook" : "patient-view",
  "context" : {
    "userId" : "Practitioner/example"
  },
  "extension" : {
    "com.example.timestamp": "2017-11-27T22:13:25Z",
    "myextension-practitionerspecialty" : "gastroenterology"
  }
}
```

Security and Protection

Security risks associated with the CDS Hooks API include:

1. The risk of an attacker intercepting confidential information and privileged authorizations transmitted between a CDS Client and a CDS Service;
2. The risk that an attacker posing as a legitimate CDS Service could receive confidential information or privileged authorizations from a CDS Client, or could provide a CDS Client with decision support recommendations that could be detrimental to a patient;
3. The risk that an attacker posing as a CDS Client subscribing to a legitimate service (that is, an intermediary) could intercept and possibly alter the data exchanged between the two parties.
4. The risk that a CDS Service could incorporate dangerous suggestions or links to dangerous apps in the cards returned to a CDS Client.
5. The risk that a browser-based implementation of CDS Hooks could be the victim of a cross-origin resource sharing (CORS) attack.
6. The risk that a CDS Service could return a decision based on outdated patient data, resulting in a safety risk to the patient.

CDS Hooks defines a security model that addresses these risks by ensuring that the identities of both the CDS Service and the CDS Client are authenticated to each other (using JWT), protecting confidential information and privileged authorizations shared between them.

Good Practices

Security

The security model of CDS Hooks requires consideration by the implementers. Security issues are never “binary”, in general they are complex, and robust implementations must take into account concepts such as risk and usability. The implementers must address their development paying close attention to security.

CDS Clients

Each CDS Client is individually responsible for determining the suitability, security and integrity of the CDS Services it uses, based on the organization's own risk management strategy. Clients must maintain a whitelist of CDS Services that they can invoke. These Services may only be invoked guaranteeing that they are Trusted Services. This is especially important since CDS Clients can send an authorization token that allows the CDS Service temporary access to the FHIR Server.

Clients must issue secure tokens for access to the FHIR Server. Each token must be unique for each CDS Service, it must be ephemeral and it must only provide access for the necessary data.

CDS Services

Each interaction between a CDS Client and a CDS Service is initiated when the Client sends a request to the Service, protected via the HTTPS protocol. JWT must be used to ensure security between parties (CDS Client - CDS Service).

CDS Services should never store, share or log the access token to the FHIR Server provided by the CDS Client. The access token must be treated as extremely sensitive and transitory data.

Maturity Model of Hooks

Like FHIR, hooks have versions and mature according to the maturity model of hooks. The intention of this model is to achieve broad community participation and consensus, before a hook is labeled mature. Also, it is assessed that a hook is necessary, implementable and valuable for CDS Services and CDS Clients who are reasonably expected to use it. The feedback from the implementer should boost the maturity of the new hooks.

The maturity levels are:

Level	Maturity Title
0	(Draft)
1	(Submitted)
2	(Tested)
3	(Considered)
4	(Documented)
5	(Mature)
6	(Normative)

Useful Links

- To get involved in the CDS Hooks community, you can go to this link for discussion forums, chat, etc.: <https://cds-hooks.org/community/>.
- You can try CDS Hooks in your sandbox: <http://sandbox.cds-hooks.org/>
- The CDS Hooks REST API specification can be found at this link: [http://editor.swagger.io/?url= https://raw.githubusercontent.com/cds-hooks/api/master/cds-hooks.yaml](http://editor.swagger.io/?url=https://raw.githubusercontent.com/cds-hooks/api/master/cds-hooks.yaml)