Draft - STU2



**FHIRcast**

**Overview**

The FHIRcast specification describes the APIs used to synchronize disparate healthcare applications' user interfaces in real time, allowing them to show the same clinical content to a user (or group of users).

In order to connect to a session, the subscribing app needs information on the session. Once that [information is retrieved](https://fhircast.org/specification/STU2/%22%20%5Cl%20%22session-discovery), the app [subscribes](https://fhircast.org/specification/STU2/%22%20%5Cl%20%22subscribing-and-unsubscribing) to specific workflow-related events for the given session. The app is then [notified](https://fhircast.org/specification/STU2/#event-notification) when those workflow-related events occur; for example, when the clinician is opening a patient's chart. The subscribing app can also [initiate context changes](https://fhircast.org/specification/STU2/#request-context-change), e.g. closing the patient's chart, by accessing the APIs specified in this specification. When the application no longer needs to receive these events, it [deletes its subscription](https://fhircast.org/specification/STU2/#unsubscribe). The notification messages describing the workflow event are defined as a simple json wrapper around one or more FHIR resources.

FHIRcast recommends the [HL7 SMART on FHIR launch protocol](http://www.hl7.org/fhir/smart-app-launch) for both session discovery and API authentication. FHIRcast enables a subscriber to receive notifications either through a WebHook or over a WebSocket connection. This protocol is modeled on the [W3C WebSub RFC](https://www.w3.org/TR/websub/), and uses GET vs POST interactions to a Hub for managing subscriptions. The Hub exposes APIs for subscribing and unsubscribing, requesting context changes and also distribute event notifications. The flow diagram presented below illustrates the series of interactions specified by FHIRcast, their origination and their outcome.



All data exchanged through the HTTP APIs SHALL be formatted, sent and received as [JSON](https://tools.ietf.org/html/rfc8259) structures, and SHALL be transmitted over channels secured using the Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS), also known as HTTPS which is defined in [RFC2818](https://tools.ietf.org/html/rfc2818). For WebSockets, data SHALL be transmitted over Secure Web Sockets (WSS) as defined in [RFC6455](https://tools.ietf.org/html/rfc6455).

**Events**

FHIRcast describes an workflow event subscription and notification scheme with the goal of improving a clinician's workflow across multiple disparate applications. The set of events defined in this specification is not a closed set; companies can define their own w events to fit specific use cases and are encouraged to propose those events to the community for standardization.

New events are proposed in a prescribed format using the [documentation template](https://fhircast.org/events/template) by submitting a [pull request](https://github.com/fhircast/docs/tree/master/docs/events). FHIRcast events are versioned, and mature according to the [Event Maturity Model](https://fhircast.org/specification/STU2/#event-maturity-model).

FHIRcast events are stateless. For a given event, opens and closes are a complete replacement of previous communicated events, not "deltas". Understanding an event SHALL not require receiving a previous or future event.

**Event Definition Format**

Each event definition specifies a single event name, a description of the workflow in which the event occurs, and the contextual information associated with the event. FHIR is the interoperable data model used by FHIRcast. The context information associated with an event is communicated as subsets of FHIR resources. Event notifications SHALL include the elements of the FHIR resources defined in the context from the event definition. Event notifications MAY include other elements of these resources. The source of these resources is the application's context or the FHIR server. The Hub SHALL return FHIR resources from the application's context. If the resource is not part of the application's context, it SHALL read them from the FHIR server.

For example, when the [ImagingStudy-open](https://fhircast.org/events/imagingstudy-open) event occurs, the notification sent to a subscriber SHALL include an ImagingStudy FHIR resource, which includes at least the elements defined in the *\_elements* query parameter, indicated in the event definition. For ImagingStudy this is defined as ImagingStudy/{id}?\_elements=identifier,accession. The element based query is defined in the [FHIR specification](https://www.hl7.org/fhir/search.html%22%20%5Cl%20%22elements).

A Hub SHALL at least send the required elements; a subscriber SHALL gracefully handle receiving a full FHIR resource in the context of a notification.

Each defined event in the standard event catalog SHALL be defined in the following format.

**Event Definition Format: hook-name**

Most FHIRcast events conform to an extensible syntax based upon FHIR resources. In the rare case where the FHIR data model doesn't describe content in the session, FHIRcast events MAY be named differently. For example, FHIR doesn't cleanly contain the concept of a user or user's session.

Patterned after the SMART on FHIR scope syntax and expressed in EBNF notation, the FHIRcast syntax for workflow related events is:

hub.events ::= ( fhir-resource | '\*' ) '-' ( 'open' | 'close' | '\*' )



FHIRcast events SHOULD conform to this extensible syntax.

Event names are unique and case-insensitive. Implementers may define their own events. Such proprietary events, SHALL be named with reverse domain notation (e.g. org.example.patient\_transmogrify). Reverse domain notation SHALL not be used by a standard event catalog. Event names SHALL not contain a dash ("-").

**Event Definition Format: Workflow**

Describe the workflow in which the event occurs. Event creators SHOULD include as much detail and clarity as possible to minimize any ambiguity or confusion amongst implementers.

**Event Definition Format: Context**

Describe the set of contextual data associated with this event. Only data logically and necessarily associated with the purpose of this workflow related event should be represented in context. An event SHALL contain all required data fields, MAY contain optional data fields and SHALL not contain any additional fields.

All fields available within an event's context SHALL be defined in a table where each field is described by the following attributes:

* **Key**: The name of the field in the context JSON object. Event authors SHOULD name their context fields to be consistent with other existing events when referring to the same context field.
* **Optionality**: A string value of either REQUIRED, OPTIONAL or Required, if exists
* **FHIR operation to generate context**: A FHIR read or search string illustrating the intended content of the event.
* **Description**: A functional description of the context value. If this value can change according to the FHIR version in use, the description SHOULD describe the value for each supported FHIR version.

**Session Discovery**

A session is an abstract concept representing a shared workspace, such as user's login session over multiple applications or a shared view of one application distributed to multiple users. FHIRcast requires a session to have a unique, unguessable and opaque identifier. This identifier is exchanged as the value of the hub.topic parameter. Before establishing a subscription, an app must not only know the hub.topic, but also the hub.url which contains the base url of the Hub.

Systems SHOULD use SMART on FHIR to authorize, authenticate and exchange initial shared context. If using SMART, following a [SMART on FHIR EHR launch](http://www.hl7.org/fhir/smart-app-launch#ehr-launch-sequence) or [SMART on FHIR standalone launch](http://www.hl7.org/fhir/smart-app-launch/#standalone-launch-sequence), the app SHALL request and, if authorized, SHALL be granted one or more fhircast OAuth 2.0 scopes. Accompanying this scope grant, the authorization server SHALL supply the hub.url and hub.topic SMART launch parameters alongside the access token and other parameters appropriate to establish initial shared context. Per SMART, when the openid scope is granted, the authorization server additionally sends the current user's identity in an id\_token.

Although FHIRcast works best with the SMART on FHIR launch and authorization process, implementation-specific launch, authentication, and authorization protocols may be possible. If not using SMART on FHIR, the mechanism enabling the app to discover the hub.url and hub.topic is not defined in FHIRcast. See [other launch scenarios](https://fhircast.org/launch-scenarios) for guidance.

**FHIRcast Authorization & SMART scopes**

FHIRcast defines OAuth 2.0 access scopes that correspond directly to [FHIRcast events](https://fhircast.org/specification/STU2/#events). Our scopes associate read or write permissions to an event. Apps that need to receive workflow related events should ask for read scopes. Apps that request context changes should ask for write scopes. Hubs may decide what specific interactions and operations will be enabled by these scopes.

Expressed in [Extended Backus-Naur Form](https://www.iso.org/obp/ui/#iso:std:iso-iec:14977:ed-1:v1:en) (EBNF) notation, the FHIRcast syntax for workflow related events is:

scope ::= ( 'fhircast' ) '/' ( FHIRcast-event | '\*' ) '.' ( 'read' | 'write' | '\*' )



The FHIRcast event name is also a [computable syntax](https://fhircast.org/specification/STU2/#event-definition-format-hook-name), the complete syntax for FHIRcast scopes is:

scope ::= 'fhircast' '/' fhir-resource '-' ( 'open' | 'close' | '\*' ) '.' ( 'read' | 'write' | '\*' )



For example, a requested scope of fhircast/patient-open.read would authorize the subscribing application to receive a notification when the patient in context changed. Similarly, a scope of fhircast/patient-open.write authorizes the subscribed app to [request a context change](https://fhircast.org/specification/STU2/#request-context-change).

**SMART Launch Example**

An example of the SmartOnFhir message is presented below.

{

 "access\_token": "i8hweunweunweofiwweoijewiwe",

 "token\_type": "bearer",

 "patient": "123",

 "expires\_in": 3600,

 "encounter": "456",

 "imagingstudy": "789",

 "hub.url" : "https://hub.example.com",

 "hub.topic": "fdb2f928-5546-4f52-87a0-0648e9ded065",

}

**Subscribing and Unsubscribing**

Subscribing and unsubscribing is how applications establish their connection and determine to which events they will be notified. Hubs SHALL support WebSockets and MAY support webhooks. If the Hub does not support webhooks then they should deny any subscription requests with webhook as the channel type.

Subscribing consists of two exchanges:

* Subscriber requests a subscription at the hub.url url.
* The hub confirms the subscription that was requested by the subscriber. This exchange can be implemented in two ways depending on the channel type.
	+ For webhook subscriptions, the Hub confirms the subscription was actually requested by the subscriber by contacting the hub.callback url.
	+ For websocket subscriptions, the Hub returns a wss url and subscriber establishes WebSocket connection.

Unsubscribing works in the same way, using the same message format. The hub.mode field is set to unsubscribe instead of subscribe. The Hub will not validate unsubscription requests with the subscriber.

**Subscription Request**

To create a subscription, the subscribing app SHALL perform an HTTP POST to the Hub's base url (as specified in hub.url) with the parameters in the table below.

This request SHALL have a Content-Type header of application/x-www-form-urlencoded and SHALL use the following parameters in its body, formatted accordingly:

| **Field** | **Optionality** | **Channel** | **Type** | **Description** |
| --- | --- | --- | --- | --- |
| hub.channel.type | REQUIRED | All | *string* | The subscriber SHALL specify a channel type of websocket or webhook. Subscription requests without this field SHOULD be rejected by the Hub. |
| hub.mode | REQUIRED | All | *string* | The literal string "subscribe" or "unsubscribe", depending on the goal of the request. |
| hub.topic | REQUIRED | All | *string* | The identifier of the session that the subscriber wishes to subscribe to or unsubscribe from. MAY be a Universally Unique Identifier ([UUID](https://tools.ietf.org/html/rfc4122)). |
| hub.events | CONDITIONAL | All | *string* | Required for subscriptions, SHALL not be present during unsubscriptions. Comma-separated list of event types from the Event Catalog for which the Subscriber wants to subscribe. Partial unsubscriptions are not supported. |
| hub.lease\_seconds | OPTIONAL | All | *number* | Number of seconds for which the subscriber would like to have the subscription active, given as a positive decimal integer. Hubs MAY choose to respect this value or not, depending on their own policies, and MAY set a default value if the subscriber omits the parameter. If using OAuth 2.0, the Hub SHALL limit the subscription lease seconds to be less than or equal to the access token's expiration. |
| hub.callback | REQUIRED | webhook | *string* | The Subscriber's callback URL where notifications should be delivered. The callback URL SHOULD be an unguessable URL that is unique per subscription. |
| hub.secret | OPTIONAL | webhook | *string* | A subscriber-provided cryptographically random unique secret string that SHALL be used to compute an [HMAC digest](https://www.w3.org/TR/websub/#bib-RFC6151) delivered in each notification. This parameter SHALL be less than 200 bytes in length. |
| hub.channel.endpoint | REQUIRED | websocket | *string* | The wss url identifying an existing WebSocket subscription. |

If OAuth 2.0 authentication is used, this POST request SHALL contain the Bearer access token in the HTTP Authorization header.

Hubs SHALL allow subscribers to re-request subscriptions that are already activated. Each subsequent and verified request to a Hub to subscribe or unsubscribe SHALL override the previous subscription state for a specific hub.topic, hub.callback / hub.channel.endpoint url, and hub.events combination. For example, a subscriber MAY modify its subscription by sending a subscription request (`hub.mode=subscribe`) with a different `hub.events` value with the same topic and callback/endpoint url. The Hub SHALL replace the subscription’s previous `hub.events` with the newly provided list of events.

For webhook subscriptions, the callback URL MAY contain arbitrary query string parameters (e.g., ?foo=bar&red=fish). Hubs SHALL preserve the query string during subscription verification by appending new, Hub-defined, parameters to the end of the list using the & (ampersand) character to join. When sending the event notifications, the Hub SHALL make a POST request to the callback URL including any query string parameters in the URL portion of the request, not as POST body parameters.

The client that creates the subscription may not be the same system as the server hosting the callback url or connecting to the wss url. (For example, some type of federated authorization model could possibly exist between these two systems.) However, in FHIRcast, the Hub assumes that the same authorization and access rights apply to both the subscribing client and the system receiving notifications.

**Subscription Response**

If the Hub URL successfully received the subscription or unsubscription requests, the Hub SHALL respond to a subscription request with an HTTP 202 "Accepted" response. This indicates that the request was received and will now be verified by the Hub. When using WebSockets, the HTTP body of the response SHALL contain an element named "hub.channel.endpoint" that contains the wss url as json. If using WebHook, the Hub SHOULD perform the verification of intent as soon as possible.

If a Hub refuses the request or finds any errors in the subscription request, an appropriate HTTP error response code (4xx or 5xx) SHALL be returned. In the event of an error, the Hub SHOULD return a description of the error in the response body as plain text, used to assist the client developer in understanding the error. This is not meant to be shown to the end user. Hubs MAY decide to reject some subscription requests based on their own policies.

**webhook vs websocket**

A hub is REQUIRED to support WebSockets and MAY support WebHook based subscriptions. Implementer feedback is solicited around the preference and desired optionality of webhooks.

**webhook Subscription Request Example**

In this example, the app asks to be notified of the patient-open and patient-close events.

POST https://hub.example.com

Host: hub.example.com

Authorization: Bearer i8hweunweunweofiwweoijewiwe

Content-Type: application/**x**-www-form-urlencoded

hub.channel.type=webhook&hub.callback=https%3A%2F%2Fapp.example.com%2Fsession%2Fcallback%2Fv7tfwuk17a&hub.mode=subscribe&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065&hub.secret=shhh-this-is-a-secret&hub.events=patient-**open**,patient-**close**

**webhook Subscription Response Example**

HTTP/1.1 202 Accepted

**websocket Initial Subscription Request Example**

In this example, the app creates an initial subscription and asks to be notified of the patient-open and patient-close events.

POST https://hub.example.com

**Host: hub.example.com**

**Authorization: Bearer i8hweunweunweofiwweoijewiwe**

**Content-Type: application/x-www-form-urlencoded**

hub.channel.type=websocket&hub.mode=subscribe&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065&hub.events=patient-open,patient-close

**websocket Subscription Response Example**

HTTP/1.1 202 Accepted

{

 "hub.channel.endpoint": wss://hub.example.com/ee30d3b9-1558-464f-a299-cbad6f8135de

}

**Subscription Denial**

If (and when) the subscription is denied, the Hub SHALL inform the subscriber. This can occur when the subscription is requested for a variety of reasons, or it can occur after the subscription had already been accepted because the Hub no longer supports that subscription (e.g. it has expired). The communication mechanism for a subscription denial varies per hub.channel.type, but the information communicated from the Hub to the subscriber does not.

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.mode | REQUIRED | *string* | The literal string "denied". |
| hub.topic | REQUIRED | *string* | The topic given in the corresponding subscription request. MAY be a UUID. |
| hub.events | REQUIRED | *string* | A comma-separated list of events from the Event Catalog corresponding to the events string given in the corresponding subscription request, which are being denied. |
| hub.reason | OPTIONAL | *string* | The Hub may include a reason. The subscription MAY be denied by the Hub at any point (even if it was previously accepted). The Subscriber SHOULD then consider that the subscription is not possible anymore. |

The below webhook flow diagram and WebSocket flow diagram and examples illustrate the subscription denial sequence and message details.

**webhook Subscription Denial**

To deny a webhook subscription, the Hub sends an HTTP GET request to the subscriber's callback URL as given in the subscription request. This request appends the fields as query string arguments. The subscriber SHALL respond with an HTTP success (2xx) code.

**webhook Subscription Denial Sequence**



**webhook Subscription Denial Example**

GET https://app.example.com/session/callback/v7tfwuk17a?hub.mode=denied&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065hub.events=patient-open,patient-close&hub.reason=session+unexpectedly+stopped HTTP 1.1

Host: subscriber

**websocket Subscription Denial**

To deny a websocket subscription, the Hub sends a json object to the subscriber through the established WebSocket connection.

**websocketSubscription Denial Sequence**



**websocket Subscription Denial Example**

{

 "hub.mode": "denied",

 "hub.topic":" "fba7b1e2-53e9-40aa-883a-2af57ab4e2c",

 "hub.events": "patient-**open**,patient-**close**",

 "hub.reason": "session unexpectedly stopped"

}

**Subscription Confirmation**

If a subscription or unsubscription is not denied, the Hub SHALL confirm the subscription. The subscription confirmation step informs the subscriber of the details of Hub's recently created subscription. For webhook subscriptions, the confirmation also verifies the intent of the subscriber and ensures that the subscriber actually controls the callback url.

**webhook Intent Verification Request**

In order to prevent an attacker from creating unwanted subscriptions on behalf of a subscriber, a Hub must ensure that a webhook subscriber did indeed send the subscription request. The Hub SHALL verify a subscription request by sending an HTTPS GET request to the subscriber's callback URL as given in the subscription request. This request SHALL have the following query string arguments appended.

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.mode | REQUIRED | *string* | The literal string "subscribe" or "unsubscribe", which matches the original request to the Hub from the subscriber. |
| hub.topic | REQUIRED | *string* | The session topic given in the corresponding subscription request. MAY be a UUID. |
| hub.events | REQUIRED | *string* | A comma-separated list of events from the Event Catalog corresponding to the events string given in the corresponding subscription request. |
| hub.challenge | REQUIRED | *string* | A Hub-generated, random string that SHALL be echoed by the subscriber to verify the subscription. |
| hub.lease\_seconds | REQUIRED | *number* | The Hub-determined number of seconds that the subscription will stay active before expiring, measured from the time the verification request was made from the Hub to the subscriber. If provided to the client, the Hub SHALL unsubscribe the client once lease\_seconds has expired, close the websocket connection if used, and MAY send a subscription denial. If the subscriber wishes to continue the subscription it MAY resubscribe. |

**webhook Intent Verification Request Example**

GET https://app.example.com/session/callback/v7tfwuk17a?hub.mode=subscribe&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065&hub.events=patient-open,patient-close&hub.challenge=meu3we944ix80ox&hub.lease\_seconds=7200 HTTP 1.1

Host: subscriber

**webhook Intent Verification Response**

If the hub.topic of the Intent Verification Request corresponds to a pending subscription or unsubscription that the subscriber wishes to carry out it SHALL respond with an HTTP success (2xx) code, a header of Content-Type: text/html, and a response body equal to the hub.challenge parameter. If the subscriber does not agree with the action, the subscriber SHALL respond with a 404 "Not Found" response.

The Hub SHALL consider other server response codes (3xx, 4xx, 5xx) to mean that the verification request has failed. If the subscriber returns an HTTP success (2xx) but the content body does not match the hub.challenge parameter, the Hub SHALL also consider verification to have failed.

The below flow diagram and example illustrate the successful subscription sequence and message details.

**webhook Successful Subscription Sequence**



**webhook Intent Verification Response Example**

HTTP/1.1 200 OK

Content-Type: text/html

meu3we944ix80ox

NOTE The spec uses GET vs POST to differentiate between the confirmation/denial of the subscription request and delivering the content. While this is not considered "best practice" from a web architecture perspective, it does make implementation of the callback URL simpler. Since the POST body of the content distribution request may be any arbitrary content type and only includes the actual content of the document, using the GET vs POST distinction to switch between handling these two modes makes implementations simpler.

**websocket Subscription Confirmation**

To confirm a subscription request, upon the subscriber establishing a WebSocket connection to the hub.channel.endpoint wss url, the Hub SHALL send a confirmation. This confirmation includes the following elements:

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.mode | Required | *string* | The literal string "subscribe". |
| hub.topic | Required | *string* | The session topic given in the corresponding subscription request. |
| hub.events | Required | *string* | A comma-separated list of events from the Event Catalog corresponding to the events string given in the corresponding subscription request. |
| hub.lease\_seconds | Required | *number* | The Hub-determined number of seconds that the subscription will stay active before expiring, measured from the time the verification request was made from the Hub to the subscriber. If provided to the client, the Hub SHALL unsubscribe the client once lease\_seconds has expired, close the websocket connection if used, and MAY send a subscription denial. If the subscriber wishes to continue the subscription it MAY resubscribe. |

**websocket Subscription Confirmation Example**

{

 "hub.mode": "subscribe",

 "hub.topic": "fdb2f928-5546-4f52-87a0-0648e9ded065",

 "hub.events": "patient-open,patient-close",

 "hub.lease\_seconds": 7200

}

**websocket Successful Subscription Sequence**



**Unsubscribe**

Once a subscribing app no longer wants to receive event notifications, it SHALL unsubscribe from the session. An unsubscribe cannot alter an existing subscription, only cancel it. Note that the unsubscribe request is performed over HTTP(s), even for subscriptions using WebSockets. websocket unsubscribes will destroy the websocket which cannot be reused. A subsequent subscription SHALL be done over a newly created and communicated WebSocket endpoint.

| **Field** | **Optionality** | **Channel** | **Type** | **Description** |
| --- | --- | --- | --- | --- |
| hub.channel.type | Required | All | *string* | The subscriber SHALL specify a channel type of websocket or webhook. Subscription requests without this field SHOULD be rejected by the Hub. |
| hub.mode | Required | All | *string* | The literal string "unsubscribe". |
| hub.topic | Required | All | *string* | The identifier of the session that the subscriber wishes to subscribe to or unsubscribe from. MAY be a UUID. |
| hub.lease\_seconds | Optional | All | *number* | This parameter MAY be present for unsubscription requests and MUST be ignored by the hub in that case. |
| hub.callback | Required | webhook | *string* | The Subscriber's callback URL. |
| hub.secret | Optional | webhook | *string* | A subscriber-provided cryptographically random unique secret string that SHALL be used to compute an [HMAC digest](https://www.w3.org/TR/websub/#bib-RFC6151) delivered in each notification. This parameter SHALL be less than 200 bytes in length. |
| hub.channel.endpoint | Conditional | websocket | *string* | Required for websocket re-subscribes and unsubscribes. The wss url identifying an existing WebSocket subscription. |

**webhook Unsubscribe Request Example**

POST https://hub.example.com

Host: hub

Authorization: Bearer i8hweunweunweofiwweoijewiwe

Content-Type: application/x-www-form-urlencoded

hub.channel.type=webhook&hub.callback=https%3A%2F%2Fapp.example.com%2Fsession%2Fcallback%2Fv7tfwuk17a&hub.mode=unsubscribe&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065&hub.secret=shhh-this-is-a-secret&hub.challenge=meu3we944ix80ox

**websocket Unsubscribe Request Example**

POST https://hub.example.com

Host: hub

Authorization: Bearer i8hweunweunweofiwweoijewiwe

Content-Type: application/x-www-form-urlencoded

hub.channel.type=websocket&hub.channel.endpoint=wss%3A%2F%2Fhub.example.com%2Fee30d3b9-1558-464f-a299-cbad6f8135de%0A&hub.mode=unsubscribe&hub.topic=fdb2f928-5546-4f52-87a0-0648e9ded065

**webhook and websocket Unsubscription Sequence**



**Event Notification**

The Hub SHALL notify subscribed apps of workflow-related events to which the app is subscribed. The notification is a JSON object communicated over the webhook or websocket channel.

**Event Notification Request**

The HTTP request notification interaction to the subscriber SHALL include a description of the subscribed event that just occurred, an ISO 8601-2 formatted timestamp in UTC and an event identifier that is universally unique for the Hub. The timestamp SHOULD be used by subscribers to establish message affinity (message ordering) through the use of a message queue. The event identifier MAY be used to differentiate retried messages from user actions.

**Event Notification Request Details**

The notification's hub.event and context fields inform the subscriber of the current state of the user's session. The hub.event is a user workflow event, from the Event Catalog (or an organization-specific event in reverse-domain name notation). The context is an array of named FHIR resources (similar to [CDS Hooks's context](https://cds-hooks.hl7.org/1.0/#http-request_1) field) that describe the current content of the user's session. Each event in the Event Catalog defines what context is included in the notification. The context contains zero, one, or more FHIR resources. Hubs SHOULD use the [FHIR \_elements parameter](https://www.hl7.org/fhir/search.html#elements) to limit the size of the data being passed while also including additional, local identifiers that are likely already in use in production implementations. Subscribers SHALL accept a full FHIR resource or the [\_elements](https://www.hl7.org/fhir/search.html#elements)-limited resource as defined in the Event Catalog.

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| timestamp | REQUIRED | *string* | ISO 8601-2 timestamp in UTC describing the time at which the event occurred. |
| id | REQUIRED | *string* | Event identifier used to recognize retried notifications. This id SHALL be unique for the Hub, for example a UUID. |
| event | REQUIRED | *object* | A json object describing the event. See below. |

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.topic | REQUIRED | string | The session topic given in the subscription request. MAY be a UUID. |
| hub.event | REQUIRED | string | The event that triggered this notification, taken from the list of events from the subscription request. |
| context | REQUIRED | array | An array of named FHIR objects corresponding to the user's context after the given event has occurred. Common FHIR resources are: Patient, Encounter, and ImagingStudy. The Hub SHALL only return FHIR resources that the subscriber is authorized to receive with the existing OAuth 2.0 access\_token's granted fhircast/ scopes. |

**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 custom behavior and information. The value of an extension element MUST be a pre-coordinated JSON object. For example, an extension on a notification could look like this:

{

 "context": [{

 "key": "patient",

 "resource": {

 "resourceType": "Patient",

 "id": "ewUbXT9RWEbSj5wPEdgRaBw3"

 }

 },

 {

 "key": "extension",

 "data": {

 "user-timezone": "+1:00"

 }

 }

 ]

}

**webhook Event Notification Request Details**

For webhook subscriptions, using the hub.secret from the subscription request, the Hub SHALL generate an HMAC signature of the payload and include that signature in the request headers of the notification. The X-Hub-Signature header's value SHALL be in the form *method=signature* where method is one of the recognized algorithm names and signature is the hexadecimal representation of the signature. The signature SHALL be computed using the HMAC algorithm ([RFC6151](https://www.w3.org/TR/websub/#bib-RFC6151)) with the request body as the data and the hub.secret as the key.

**POST** https://app.example.com/session/callback/v7tfwuk17a HTTP/1.1

Host: subscriber

X-Hub-Signature: sha256=dce85dc8dfde2426079063ad413268ac72dcf845f9f923193285e693be6ff3ae

**<json object>**

**Event Notification Request Example**

For both webhook and websocket subscriptions, the event notification content is the same.

{

 "timestamp": "2018-01-08T01:37:05.14",

 "id": "q9v3jubddqt63n1",

 "event": {

 "hub.topic": "fdb2f928-5546-4f52-87a0-0648e9ded065",

 "hub.event": "patient-open",

 "context": [

 {

 "key": "patient",

 "resource": {

 "resourceType": "Patient",

 "id": "ewUbXT9RWEbSj5wPEdgRaBw3",

 "identifier": [

 {

 "type": {

 "coding": [

 {

 "system": "http://terminology.hl7.org/CodeSystem/v2-0203",

 "value": "MR",

 "display": "Medication Record Number"

 }

 "text": "MRN"

 ]

 }

 }

 ]

 }

 }

 ]

 }

}

**Event Notification Response**

The subscriber SHALL respond to the event notification with an appropriate HTTP status code. In the case of a successful notification, the subscriber SHALL respond with an HTTP 200 (OK) or 202 (Accepted) response code to indicate a success; otherwise, the subscriber SHALL respond with an HTTP error status code. The Hub MAY use these statuses to track synchronization state.

In the case of a successful notification, if the subscriber is able to implement the context change, an HTTP 200 (OK) is the appropriate code; if the subscriber has successfully received the event notification, but has not yet taken action: an HTTP 202 (Accepted).

**webhook Event Notification Response Example**

For webhook subscriptions, the HTTP status code is communicated in the HTTP response, as expected.

HTTP/1.1 200 OK

**websocket Event Notification Response Example**

For websocket subscriptions, the id of the event notification and the HTTP status code is communicated from the client to Hub through the existing WebSocket channel, wrapped in a json object. Since the WebSocket channel does not have a synchronous request/response, this id is necessary for the Hub to correlate the response to the correct notification.

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| id | Required | *string* | Event identifier from the event notification to which this response corresponds. |
| status | Required | *numeric HTTP status code* | Numeric HTTP response code to indicate success or failure of the event notification within the subscribing app. Any 2xx code indicates success, any other code indicates failure. |

{

 "id": "q9v3jubddqt63n1",

 "status": "200"

}

**webhook and websocket Event Notification Sequence**



**Event Notification Errors**

All standard events are defined outside of the base FHIRcast specification in the Event Catalog with the single exception of the infrastructural syncerror event.

If the subscriber cannot follow the context of the event, for instance due to an error or a deliberate choice to not follow a context, the subscriber SHOULD communicate the error to the Hub in one of two ways.

* Responding to the event notification with an HTTP error status code as described in [Event Notification Response](https://fhircast.org/specification/STU2/#event-notification-response).
* Responding to the event notification with an HTTP 202 (Accepted) as described above, then, once experiencing the error, send a syncerror event to the Hub.

If the Hub receives an error notification from a subscriber, it SHOULD generate a syncerror event to the other subscribers of that topic. syncerror events are like other events in that they need to be subscribed to in order for an app to receive the notifications and they have the same structure as other events, the context being a single FHIR OperationOutcome resource.

**Event Notification Error Request**

**Request Context Change Parameters**

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| timestamp | REQUIRED | *string* | ISO 8601-2 timestamp in UTC describing the time at which the syncerror event occurred. |
| id | REQUIRED | *string* | Event identifier, which MAY be used to recognize retried notifications. This id SHALL be unique and could be a UUID. |
| event | REQUIRED | *object* | A json object describing the event. See [below](https://fhircast.org/specification/STU2/#event-notification-error-event-object-parameters). |

**Event Notification Error Event Object Parameters**

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.topic | REQUIRED | string | The session topic given in the subscription request. |
| hub.event | REQUIRED | string | Shall be the string syncerror. |
| context | REQUIRED | array | An array containing a single FHIR OperationOutcome. The OperationOutcome SHALL use a code of processing. The OperationOutcome's details SHALL contain the id of the event that this error is related to as a code with the system value of "https://fhircast.hl7.org/events/syncerror/eventid" and the name of the relevant event with a system value of "https://fhircast.hl7.org/events/syncerror/eventname". Other coding values can be included with different system values so as to include extra information about the syncerror. |

**Event Notification Error Example**

**POST** https://hub.example.com/7jaa86kgdudewiaq0wtu HTTP/1.1

Host: hub

Authorization: Bearer i8hweunweunweofiwweoijewiwe

Content-Type: application/json

{

 "timestamp": "2018-01-08T01:37:05.14",

 "id": "q9v3jubddqt63n1",

 "event": {

 "hub.topic": "7544fe65-ea26-44b5-835d-14287e46390b",

 "hub.event": "syncerror",

 "context": [

 {

 "key": "operationoutcome",

 "resource": {

 "resourceType": "OperationOutcome",

 "issue": [

 {

 "severity": "warning",

 "code": "processing",

 "diagnostics": "AppId3456 failed to follow context",

 "details": {

 "coding": [

 {

 "system": "https://fhircast.hl7.org/events/syncerror/eventid",

 "code": "fdb2f928-5546-4f52-87a0-0648e9ded065"

 },

 {

 "system": "https://fhircast.hl7.org/events/syncerror/eventname",

 "code": "patient-open"

 }

 ]

 }

 }

 ]

 }

 }

 ]

 }

}

**webhook and websocket Event Notification Error Sequence**



**Request Context Change**

Similar to the Hub's notifications to the subscriber, the subscriber MAY request context changes with an HTTP POST to the hub.url. The Hub SHALL either accept this context change by responding with any successful HTTP status or reject it by responding with any 4xx or 5xx HTTP status. Similar to event notifications, described above, the Hub could also respond with a 202 (Accepted) status, process the request, then later respond with a syncerror event in order to reject the request. In this case the syncerror would only be sent to the requestor. The subscriber SHALL be capable of gracefully handling a rejected context request.

Once a requested context change is accepted, the Hub SHALL broadcast the context notification to all subscribers, including the original requestor. The requestor can use the broadcasted notification as confirmation of their request. The Hub reusing the request's id is further confirmation that the event is a result of their request.



**Request Context Change Request**

**Request Context Change Parameters**

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| timestamp | REQUIRED | *string* | ISO 8601-2 timestamp in UTC describing the time at which the event occurred. |
| id | REQUIRED | *string* | Event identifier, which MAY be used to recognize retried notifications. This id SHALL be uniquely generated by the subscriber and could be a UUID. Following an accepted context change request, the Hub MAY re-use this value in the broadcasted event notifications. |
| event | REQUIRED | *object* | A json object describing the event. See [below](https://fhircast.org/specification/STU2/#request-context-change-event-object-parameters). |

**Request Context Change Event Object Parameters**

| **Field** | **Optionality** | **Type** | **Description** |
| --- | --- | --- | --- |
| hub.topic | REQUIRED | string | The session topic given in the subscription request. |
| hub.event | REQUIRED | string | The event that triggered this request for the subscriber, taken from the list of events from the subscription request. |
| context | REQUIRED | array | An array of named FHIR objects corresponding to the user's context after the given event has occurred. Common FHIR resources are: Patient, Encounter, ImagingStudy and List. |

**POST** https://hub.example.com/7jaa86kgdudewiaq0wtu HTTP/1.1

Host: hub

Authorization: Bearer i8hweunweunweofiwweoijewiwe

Content-Type: application/json

{

 "timestamp": "2018-01-08T01:40:05.14",

 "id": "wYXStHqxFQyHFELh",

 "event": {

 "hub.topic": "fdb2f928-5546-4f52-87a0-0648e9ded065",

 "hub.event": "close-patient-chart",

 "context": [

 {

 "key": "patient",

 "resource": {

 "resourceType": "Patient",

 "id": "798E4MyMcpCWHab9",

 "identifier": [

 {

 "type": {

 "coding": [

 {

 "system": "http://terminology.hl7.org/CodeSystem/v2-0203",

 "value": "MR",

 "display": "Medication Record Number"

 }

 "text": "MRN"

 ]

 }

 }

 ]

 }

 }

 ]

 }

}

**Conformance**

The FHIRcast specification can be described as a set of capabilities and any specific FHIRcast Hub may implement a subset of these capabilities. A FHIRcast Hub declares support for FHIRcast and specific capabilities by exposing an extension on its FHIR server's CapabilityStatement as described below.

**Declaring support for FHIRcast**

A FHIRserver SHALL declare support for FHIRcast using one ore more FHIRcast extensions on its FHIR CapabilityStatement's rest element. The FHIRcast extension has the following internal components:

| **Component** | **Cardinality** | **Type** | **Description** |
| --- | --- | --- | --- |
| eventsSupported | 1..\* | string | Space-delimited list of FHIRcast events supported by the Hub. |
| hub.url | 0..1 | url | The url at which an app subscribes. May not be supported by client-side Hubs. |
| websocketSupport | 1..1 | boolean | The static value: true, indicating support for websockets. |
| webhookSupport | 0..1 | boolean | true or false indicating support for webhooks. |
| fhircastVersion | 0..1 | string | STU1 or STU2 indicating support for a specific version of FHIRcast. |

**CapabilityStatement Extension Example**

{

 "resourceType": "CapabilityStatement",

...

 "rest": [{

 ...

 "extension": [

 {

 "url": "http://fhircast.hl7.org/StructureDefinition/fhircast-configuration",

 "extension": [

 {

 "url": "eventsSupported",

 "valueUri": "patient-open patient-close com.example.researchstudy-transmogrify"

 },

 {

 "url": "hub.url",

 "valueUri": "https://hub.example.com/fhircast/hub.v2"

 },

 {

 "url": "websocketSupport",

 "valueBoolean": true

 },

 {

 "url": "webhookSupport",

 "valueBoolean": false

 },

 {

 "url": "fhircastVersion",

 "valueString": "STU2"

 }

 ]

 ] ...

**Change Management and Versioning**

**Event Maturity Model**

The intent of the FHIRcast Event Maturity Model is to attain broad community engagement and consensus, before an event is labeled as mature, and to ensure that the event is necessary, implementable, and worthwhile to the systems that would reasonably be expected to use it. Implementer feedback should drive the maturity of new events. Diverse participation in open developer forums and events, such as HL7 FHIR Connectathons, is necessary to achieve significant implementer feedback. The below criteria will be evaluated with these goals in mind.

| **Maturity Level** | **Maturity title** | **Requirements** |
| --- | --- | --- |
| 0 | Draft | Event is correctly named and defined per the [FHIRcast event template](https://fhircast.org/events/template). |
| 1 | Submitted | *The above, and …* Event definition is written up as a pull request using the [Event template](https://fhircast.org/events/template/) and community feedback is solicited from the community (e.g. the zulip FHIRcast stream](https://chat.fhir.org/#narrow/stream/179271-FHIRcast)). |
| 2 | Tested | *The above, and …* The event has been tested and successfully supports interoperability among at least one Hub and two independent subscribing apps using semi-realistic data and scenarios (e.g. at a FHIR Connectathon). The github pull request defining the event is approved and published. |
| 3 | Considered | *The above, and …* At least 3 distinct organizations recorded ten distinct implementer comments (including a github issue, tracker item, or comment on the event definition page), including at least two Hubs and three subscribing apps. The event has been tested at two connectathons. |
| 4 | Documented | *The above, and …* The author agrees that the artifact is sufficiently stable to require implementer consultation for subsequent non-backward compatible changes. The event is implemented in the standard FHIRcast reference implementation and multiple prototype projects. The Event specification SHALL:Identify a broad set of example contexts in which the event may be used with a minimum of three, but as many as 10.Clearly differentiate the event from similar events or other standards to help an implementer determine if the event is correct for their scenario.Explicitly document example scenarios when the event should not be used. |
| 5 | Mature | *The above, and ...* The event has been implemented in production in at least two Hubs and three independent subscribing apps. An HL7 working group ballots the event and the event has passed HL7 STU ballot. |
| 6 | Normative | *The above, and ...* the responsible HL7 working group and the sponsoring working group agree the material is ready to lock down and the event has passed HL7 normative ballot |

**Event Maturity**

As each event progresses through a process of being defined, tested, implemented, used in production environments, and balloted, the event's formal maturity level increases. Each event has its own maturity level, which SHALL be defined in the event's definition and correspond to the [Event Maturity Model](https://fhircast.org/specification/STU2/#event-maturity-model).

**Change Log**

Changes made to an event's definition SHALL be documented in a change log to ensure event consumers can track what has been changed over the life of an event. The change log SHALL contain the following elements:

* Version: The version of the change
* Description: A description of the change and its impact

For example:

| **Version** | **Description** |
| --- | --- |
| 1.1 | Added new context FHIR object |
| 1.0.1 | Clarified workflow description |
| 1.0 | Initial Release |
| --- |  |

**Glossary**

* session: an abstract concept representing a shared workspace, such as a user's login session across multiple applications or a shared view of one application distributed to multiple users. A session results from a user logging into an application and can encompass one or more workflows.
* topic: an identifier of a session
* client: subscribes to and requests or receives session events
* current context: data associated with a session at a given time and communicated between clients that share a session
* session event: a user initiated workflow event, communicated to clients, containing the current context

**Revision History**

All changes to the FHIRcast specification are tracked in the [specification's HL7 github repository](https://github.com/HL7/fhircast-docs/commits/master). Further, issues may be submitted and are tracked in [jira](https://jira.hl7.org/browse/FHIR-25651?filter=12642) or (historically as) [github issues](https://github.com/HL7/fhircast-docs/issues). For the reader's convenience, the below table additionally lists significant changes to the specification.

**20200715 Significant changes as part of the STU2 publication included:**

* Introduction of WebSockets as the preferred communication mechanism over webhooks.
* Creation of a FHIR CapabilityStatement extension to support Hub capability discovery.
* Additional, required information on syncerror OperationOutcome (namely communication of the error'd event's id and event name).
* Websocket wss url communicated in HTTP body, instead of Content-Location HTTP header.
* Subscribers should differentiate between immediately applied context changes and mere successfully received notifications with HTTP code responses of 200 and 202, respectively.