# Task transmission & Fulfilment request options

## Direct POST of request to fulfiller’s system

1. Placer system invokes a “create” action by POSTing a ‘request’ resource (e.g. MedicationOrder, DiagnosticOrder, ReferralRequest, etc.) to the appropriate RESTful resource endpoint (e.g. [base]/MedicationOrder) on the filler system and sets a “flag” on the resource that indicates the request is “actionable”[[1]](#footnote-1)
2. The filler synchronously responds with a “201” indicating that that they have received and stored (created) the resource on their system
3. At some later point, the filler POSTs an ‘event’ resource (e.g. MedicationDispense, DiagosticReport, Encounter, etc.) to the appropriate resource endpoint on the placer system, including a link to the ‘request’ resource that the action was performed in fulfillment of.
4. The placer system synchronously responds with a “201” indicating they’ve received and store (created) the resource on their system

**Benefit**

* Lowest amount of overhead. No need for Task. No need for polling or subscriptions
* Explicit acknowledgement that filler has received the request

**Limitations**

* Can only use when requesting fulfillment (can’t use to request status change or other updates)
* Placer and filler must be able to communicate directly (i.e. know each other’s respective endpoints) and must each have a FHIR server and must have “write” permissions to each other’s servers
	+ This could become unmanageable if there are a large (or dynamic) number of placers and fillers that need to communicate
* No indication of agreement to act on the request
* There’s no ability to negotiate fulfillment – no ability to say “no”

**Usage**

Use this when there’s no ability to have intermediaries and no support/need for complexity of Task, polling or pub/sub (and no need for negotiation or the ability for the filler to say “no”). This is a pseudo-messaging architecture that doesn’t actually use messaging architecture.

## POST of request to placer/intermediary system, receiver uses polling

1. Placer system invokes a “create” action by POSTing a ‘request’ resource (e.g. MedicationOrder, DiagnosticOrder, ReferralRequest, etc.) to the appropriate RESTful resource endpoint (e.g. [base]/MedicationOrder) on either its own system or a third party intermediary system and sets a “flag” on the resource that indicates the request is “actionable”. The request explicitly identifies the intended fullfiller
2. The filler system, at some agreed frequency, RESTfully queries the placer or intermediary system to see if there are any “new” requests that: are tagged as “actionable”, have the filler identified as the intended performer, and are a type of request “of interest” to the filler. (The queries could be initiated by the filler system automatically in the background or upon triggering by user of the filler system.)
3. At some later point, the filler POSTs an ‘event’ resource (e.g. MedicationDispense, DiagosticReport, Encounter, etc.) to the appropriate resource endpoint on either its own system, the same intermediary as the request was placed on or some alternate intermediary, including a link to the ‘request’ resource that the action was performed in fulfillment of.
4. The placer system, at some agreed frequency, RESTfully queries the filler or intermediary system to see if there are any “new” events that are tied to any outstanding orders the placer has initiated. (The queries could be initiated by the placer system automatically in the background or upon triggering by user of the placer system.)

**Benefit**

* Placer & Server don’t have to communicate directly (can act through intermediary). This can reduce the number of point-to-point interfaces that need to be supported.
* No need for Task.

**Limitations**

* Can only use when requesting fulfillment (can’t use to request status change or other updates)
* Placer and filler must regularly poll to see if there’s anything new. (Can represent significant communication overhead)
* Polling by the placer for “anything related to these 500 open orders” could be onerous, especially if some orders never get closed.
* Placer and fulfiller must know where to poll for content – this could be a large number of systems
* No indication of agreement to act on the request
* There’s no ability to negotiate fulfillment – no ability to say “no”
* Placer may not know when (or if) filler system has retrieved the request

**Usage**

Use this when there’s no ability to have intermediaries and no support/need for complexity of Task and where no Subscription infrastructure exists. This is a more typically RESTful approach where data resides on the server “owned” by the data creator and is accessed by other systems.

## POST of request to placer/intermediary system, receiver uses subscription

1. Same as B – placer posts to itself or to intermediary
2. Filler has set up a subscription to requests of interest (same criteria as for B)
3. When placer’s request is posted, filler receives a notification that a new, relevant request exists and queries to receive it
4. Same pattern follows for transmission of response from filler back to placer

**Benefit**

* Placer & Server don’t have to communicate directly (can act through intermediary). This can reduce the number of point-to-point interfaces that need to be supported.
* No need for Task.
* Lower communication overhead than polling

**Limitations**

* Can only use when requesting fulfillment (can’t use to request status change or other updates)
* Additional complexity of setting up and maintaining a subscription infrastructure
* Placer and fulfiller must know where to subscribe for content – this could be a large number of systems
* No indication of agreement to act on the request
* There’s no ability to negotiate fulfillment – no ability to say “no”
* Placer may not know when (or if) filler system has retrieved the request

**Usage**

Same a B, but in an environment where subscription capability does exist.

## POST of Task to filler system

1. Placer POSTs the request to its own system or to an intermediary system
2. Placer POSTs a Task resource to the filler system, pointing to the request resource and seeking fulfillment
3. Fulfiller system queries to retrieve the referenced request
4. Fulfiller Updates the Task to indicate acceptance of the task (and possibly interim progress notes)
5. Placer either polls the Task to note acceptance and changes or uses a subscription to determine the same
6. Fulfiller POSTs an event resource to its own system or to an intermediary system[[2]](#footnote-2)
7. Fulfiller Updates the Task resource to change its status to completed and to point to the event resource
8. Placer system becomes aware of the update via polling or subscription and retrieves the referenced event resource

**Benefit**

* Can use this approach for request other than just when requesting fulfillment (e.g. to request status change or other updates)
* There’s an ability to negotiate fulfillment – i.e. the ability to say “no”
* Explicit acknowledgement that filler has received and agreed to act on the request

**Limitations**

* Additional complexity of setting up and maintaining a subscription or polling infrastructure
* Additional complexity of using Task
* Placer and filler may need to be able to communicate directly (i.e. know each other’s respective endpoints) and have a FHIR server and must have “write” permissions to each other’s servers
	+ This could become unmanageable if there are a large (or dynamic) number of placers and fillers that need to communicate
	+ May not apply if there’s an intermediary
* Placer and fulfiller must know where to subscribe for content – this could be a large number of systems
* Placer may not know when (or if) filler system has retrieved the request

**Usage**

Why would anyone do this??

## POST of “open” Task to intermediary system

1. Placer POSTs the request to its own system or to an intermediary system
2. Placer POSTs a Task resource to its own system or an intermediary system, pointing to the request resource and seeking fulfillment
	1. Task does not have a specified “performer” (but may have performer type)
3. Fulfiller system uses either polling or pub/sub to become aware of the existence of the task
4. Fulfiller system queries to retrieve the referenced request
5. Fulfiller system updates the Task to indicate “ownership” and agreement to fulfill[[3]](#footnote-3)
6. Fulfiller may update the Task to indicate interim progress notes
7. Placer either polls the Task to note acceptance and changes or uses a subscription to determine the same
8. Fulfiller POSTs an event resource to its own system or to an intermediary system
9. Fulfiller Updates the Task resource to change its status to completed and to point to the event resource
10. Placer system becomes aware of the update via polling or subscription and retrieves the referenced event resource

**Benefit**

* No need for placer and fulfiller system to talk to each other directly
* Can use this approach for request other than just when requesting fulfillment (e.g. to request status change or other updates)
* There’s an ability to negotiate fulfillment – i.e. the ability to say “no”
* Explicit acknowledgement that filler has received and agreed to act on the request

**Limitations**

* Additional complexity of setting up and maintaining a subscription or polling infrastructure
* Additional complexity of using Task
* Placer and filler may need to be able to communicate directly (i.e. know each other’s respective endpoints) and have a FHIR server and must have “write” permissions to each other’s servers
	+ This could become unmanageable if there are a large (or dynamic) number of placers and fillers that need to communicate
	+ May not apply if there’s an intermediary
* Placer and fulfiller must know where to subscribe for content – this could be a large number of systems
* Placer may not know when (or if) filler system has retrieved the request

**Usage**

Used when requests are not (always) directed to a specific filler and where fillers may respond to tasks from multiple placers. This creates a sort of “market” where fillers choose what to take on and negotiate what work they perform.

## POST of “request” resource for filler system, response via Task

1. Placer system invokes a “create” action by POSTing a ‘request’ resource (e.g. MedicationOrder, DiagnosticOrder, ReferralRequest, etc.) to the appropriate RESTful resource endpoint (e.g. [base]/MedicationOrder) on the filler, placer or intermediary system and sets a “tag” on the resource that indicates the request is “actionable”
2. Filler POSTs a Task resource to its own system or an intermediary system, pointing to the request resource and indicating intent to fulfill or refusal to fulfill
3. Placer system uses either polling or pub/sub to become aware of the existence of the task and fulfillment intent
4. Fulfiller may update the Task to indicate interim progress notes
5. Placer either polls the Task to note acceptance and changes or uses a subscription to determine the same
6. Fulfiller POSTs an event resource to its own system or to an intermediary system
7. Fulfiller Updates the Task resource to change its status to completed and to point to the event resource
8. Placer system becomes aware of the update via polling or subscription and retrieves the referenced event resource

**Benefit**

* There’s an ability to negotiate fulfillment – i.e. the ability to say “no”
* Explicit acknowledgement that filler has received and agreed to act on the request (though no need for the placer to check)

**Limitations**

* Can only use when requesting fulfillment (can’t use to request status change or other updates)
* Additional complexity of setting up and maintaining a subscription or polling infrastructure
* Additional complexity of using Task
* Placer and filler may need to be able to communicate directly (i.e. know each other’s respective endpoints) and have a FHIR server and have “write” permissions to each other’s servers (if no intermediary is used)
* Placer and fulfiller must know where to subscribe for content – this could be a large number of systems

**Usage**

Use this when the filler needs to have complete ownership over the task and there’s no ability for both placer & filler to manipulate tasks on a common server

## Workflow broker

1. Placer POSTs the request to its own system or to an intermediary system
2. Broker detects that new un-assigned request (without a Task yet created and falling within the scope of the Broker to ensure fulfillment) via polling or subscription
3. Broker POSTs a Task resource to its own system or an intermediary system, pointing to the request resource and seeking fulfillment from a specific filler
	1. Task does not have a specified “performer” (but may have performer type)
4. If the Task is rejected by one potential recipient, the broker may create a new task to seek fulfillment from others
5. Continue as per E

**Benefit**

* Offloads responsibility for seeking fulfillment from the placer system, but more actively solicits fulfillment than a simple “post the task and see who takes it”. Also allows prioritized assignment of tasks (i.e. some fillers may be preferred over others)

**Limitations**

* Requires a broker to exist
* Broker must know all available fillers and their capabilities to allow appropriate assignment

**Usage**

* Appropriate in environments that have a workflow engine that takes on responsibility for ensuring fulfillment

## Messaging request from placer to filler & acknowledgment

1. Placer sends message to filler system including Request resource (and other relevant resources) along with a MessageHeader with an “event” code saying “please fulfill” and “data” element pointing to the Request resource as the item to fulfill. Message could potentially use Task instead of MessageHeader.event to convey desired action (ongoing discussion)
2. Filler system sends a response indicating receipt of the message and, optionally an indication of their intention to fulfill the request
3. Filler system may send incremental messages to the placer showing progress (e.g. specimen collected, preliminary results, final results)

**Benefit**

* Reduced number of communications
* All relevant data sent in one package
* Responses can be asynchronous and content may routed
* There’s an ability to negotiate fulfillment – i.e. the ability to say “no”
* Can request things other than just fulfillment (e.g. please suspend)
* Explicit acknowledgement that filler has received and agreed to act on the request (though no need for the placer to check)

**Limitations**

* Messaging is “heavy”
* Need to negotiate what allowed responses are and what data can be present in request and response messages
* Additional complexity of setting up and maintaining a subscription or polling infrastructure
* Additional complexity of using Task
* Need message delivery infrastructure in place

**Usage**

Existing messaging (likely v2) infrastructure and a need to stay consistent with that architecture

## Services request from placer to filler & acknowledgment

(This scenario needs work – not a lot of experience using services to manage the fulfillment process)

1. Placer may create and store a Request resource on their own system or an intermediary.
2. Placer invokes a service on the filler system saying “please fulfill this order”, including the content or a reference to the request resource and any other relevant data
3. Filler system responds (synchronously if using HTTP, but may be asynchronous if using SOAP or other transport mechanisms) with conformation of receipt and, optionally indication of intention to fulfill and/or results

**Benefit**

* ???

**Limitations**

* ???

**Usage**

* Not sure why anyone would do this?

# Additional scenarios

## Querying the status of a workflow using services

## Cancellation of a Task

1. Provenances for each of the creates could be created on the placer, filler, both, neither or a third party system [↑](#footnote-ref-1)
2. Some tasks may not have an associated event – they might just update the status to “completed”. E.g. a request to suspend a prescription might not result in an updated task that points to the revised prescription version. [↑](#footnote-ref-2)
3. Standard FHIR collision-detection mechanisms (i.e. eTag) are used to prevent multiple systems from trying to claim ownership [↑](#footnote-ref-3)