Access 4 Learning Community

Powered by: SIF

Simple. Secure. Scalable. Standard.

SIF Infrastructure Specification 3.3: Functional Services

www.A4L.org

Version 3.3, May 2019

Introduction

A Functional Service coordinates one or more data exchanges such that they have a beginning, middle, and an end. Importantly, a functional service addresses a particular user story.

A Functional Service consists of:

- a 'functional service' definition describing the service
- a 'Job Object' format for communicating the intent, status, and results of the Job
- one or more 'Phases' - which accomplish the Job step-by-step
- one or more `States', which the Job transitions within a Phase

Functional Services, in a SIF 3 Environment, perform in a similar manner to Data Object Services. The SIF 3 infrastructure routes the Function Service message, and the Data Model defines a 'use case driven' Job Object, in order to coordinate the multiple phases necessary to accomplish the desired result.

Examples of functional services include:

- student enrollment transaction
    - In a transaction, if the entire process is not completed successfully, any previously successful steps are backed out.
- assessment grading service
    - The purpose of a service is often adding a way to know when a long-running task has been completed and final results can be understood.

Job Object

All functional services **must** use this object design to track state.

1. The Consumer creates the Job Object by a request to the Provider.
2. The Consumer includes a payload in the Job Object, to be used by the Provider as input into the initial Phase of the Job.
3. The Provider updates the Job Object, sometimes as a result of a Consumer request (e.g. phase transitions), sometimes through information it is privy to (e.g. the access rights to phases and states for the Consumer).

4. The Consumer consults the Provider for the latest version of the Job Object during Job execution.
5. The Consumer may issue a request to the Provider for a particular phase of the job to trigger an action of that phase which may result in a transition to a new Phase, or the Provider may make the transition automatically. The Consumer may include a payload in their phase request to the Provider, to be used by the Provider as input into that phase of the Job.
6. Either the Consumer or Provider may delete the Job Object, depending on the service definition.

XPath	Chars	Description	Type
/job/@id		Unique identifier of the job.	uuidType
/job/name	M	The name of the job, e.g. "grading" or "sre".	token
/job/description	O	A description of the job, e.g. "Bowers Elementary School Final Marks"	string
/job/state	O	The current enumerable state of the job.	jobStateType. Legal values are NOTAPPLICABLE, NOTSTARTED, PENDING, SKIPPED, INPROGRESS, COMPLETED, FAILED. These are detailed below.
/job/stateDescription	O	A descriptive message elaborating on the current state, e.g. if the current state is "FAILED" the stateDescription	string

		may be "Timeout occurred".	
/job/created	O	The datetime this job was created.	dateTime
/job/lastModified	O	The datetime this job description was last modified.	dateTime
/job/timeout	O	The amount of time after creation before this job is automatically deleted.	duration
/job/phases/phase/name	M	The name of the Phase unique within the context of the owning job.	token
/job/phases/phase/states/state/type	M	The type of this State object.	phaseStateType. Legal values are NOTAPPLICABLE, NOTSTARTED, PENDING, SKIPPED, INPROGRESS, COMPLETED, FAILED. These are detailed below.
/job/phases/phase/states/state/created	O	The datetime this job state was created.	dateTime
/job/phases/phase/states/state/lastModified	O	The datetime this job state was last modified.	dateTime
/job/phases/phase/states/state/description	O	A descriptive message elaborating the condition of this state, e.g. if the state is	string

		"FAILED" the stateDescription may be "Timeout occurred".	
/job/phases/phase/required	M	Whether or not this phase is required for the job to complete successfully.	boolean
/job/phases/phase/rights/right/@type	M	The type of the Access Right relating to job phases	string
/job/phases/phase/rights/right	M	The Access Right relating to job phases	string
/job/phases/phase/statesRights/right/@type	M	The type of the Access Right relating to job states	string
/job/phases/phase/statesRights/right	M	The Access Right relating to job states	string
/job/initialization/phaseName	O	Name of initialization phase the payload corresponds to.	token
/job/initialization/payload	M	Information that needs to be provided for the initialization phase, **should** result in corresponding phase having an INPROGRESS, COMPLETED, or FAILED state.	any

Each functional service should define the expectations of how management of the job is managed for both the Consumer and Provider. For instance, certain optional fields may need to be included in order to successfully cause a job to be created.

Example

The following job illustrates these phase and state transitions:

- Phase: Read Permission
  - State: Not Started
  - State: In Progress (interaction with external agents)
  - State: Completed (move to next state)
- Phase: Create Learners
  - State: Not Started
  - State: Pending (automated, no consumer interaction)
  - State: Completed (move to next state)
- Phase: Read Receipt
  - State: Skipped (Consumer elected not to go through this phase)

```
<job xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.sifassociation.org/infrastructure/3.3"
 id="e03bf3f0-0152-1000-007f-00505686707f">
  <name>newIntake</name>
  <description>Bowers Elementary School</description>
  <state>INPROGRESS</state>
  <stateDescription>Completed 2/3 Tasks</stateDescription>
  <created>2016-03-09T09:00:00</created>
  <lastModified>2016-03-09T10:00:00</lastModified>
  <timeout>P3D</timeout>
  <phases>
    <phase>
      <name>readPermission</name>
      <states>
        <state>
          <type>NOTSTARTED</type>
          <created>2016-03-07T09:00:00</created>
          <lastModified>2016-03-07T09:00:01</lastModified>
          <description>Job started</description>
        </state>
        <state>
          <type>INPROGRESS</type>
          <created>2016-03-08T09:00:00</created>
          <lastModified>2016-03-08T09:00:01</lastModified>
          <description>Requesting Permissions</description>
```

```
      </state>
      <state>
        <type>COMPLETED</type>
        <created>2016-03-09T09:00:00</created>
        <lastModified>2016-03-09T09:00:01</lastModified>
        <description>Requested Permissions</description>
      </state>
    </states>
    <required>true</required>
    <rights>
      <right type="QUERY">APPROVED</right>
    </rights>
    <statesRights>
      <right type="CREATE">APPROVED</right>
    </statesRights>
  </phase>
  <phase>
    <name>createLearners</name>
    <states>
      <state>
        <type>NOTSTARTED</type>
        <created>2016-03-09T09:00:01</created>
        <lastModified>2016-03-09T09:00:01</lastModified>
        <description>Phase started</description>
      </state>
      <state>
        <type>PENDING</type>
        <created>2016-03-09T09:00:02</created>
        <lastModified>2016-03-09T09:00:02</lastModified>
        <description>Transmission underway</description>
      </state>
      <state>
        <type>COMPLETED</type>
        <created>2016-03-09T09:59:58</created>
        <lastModified>2016-03-09T09:59:58</lastModified>
        <description>Sent</description>
      </state>
    </states>
    <required>true</required>
    <rights>
      <right type="CREATE">APPROVED</right>
    </rights>
    <statesRights>
      <right type="CREATE">APPROVED</right>
    </statesRights>
```

```
    </phase>
    <phase>
      <name>readReceipt</name>
      <states>
        <state>
          <type>SKIPPED</type>
          <created>2016-03-09T09:59:59</created>
          <lastModified>2016-03-09T10:00:00</lastModified>
          <description>Consumer didn't Care</description>
        </state>
      </states>
      <required>false</required>
      <rights>
        <right type="QUERY">APPROVED</right>
      </rights>
      <statesRights>
        <right type="CREATE">APPROVED</right>
      </statesRights>
    </phase>
  </phases>
  <initialization>
    <phaseName>initial-parameters</phaseName>
    <payload>
      <parameters>
        <parameter>
          <key>timetableRefid</key><value>45c22938-7217-4d99-832d-0e61b836ba9e</value>
        </parameter>
        <parameter>
          <key>changeType</key><value>daily</value>
        </parameter>
        <parameter>
          <key>variationDate</key><value>2016-12-12</value>
        </parameter>
      </parameters>
    </payload>
  </initialization>
</job>
```

Creating the Job

Since the Job Object communicates the intent, status, and results of the Functional Service, and it is updated by both the Consumer and the Provider, it needs to live somewhere accessible to all parties involved who will require this particular service. Generally, the client (who could be either the data provider or consumer) kicks things off by creating an instance of the Job Object on the server. All parties need to follow their defined role in order for interoperability to be accomplished.

Both Job Creation requests, and subsequent requests by the Consumer to update the Job Object, may be either synchronous or asynchronous. Requests involving a single Job are always 'immediate'; requests involving multiple Jobs may be 'delayed'. 'Delayed' support is mandatory for brokered environments.

URL Form

```
[services connector]/[job name]+s/[job name]
```

Example URL

https://example.a4l.org/jobs/newIntakes/newIntake

Example Payload

```
<job xmlns="http://www.sifassociation.org/infrastructure/3.3">
  <name>newIntake</name>
</job>
```

Response

A flushed-out object similar to the example in the preceding section is returned. It will not yet contain populated values for the phases and states that the job has actually traversed; but it will contain rights statements for the phrases and states, which are given by the Provider (whether the Provider or the Consumer has created the Job Object).

Supported Job Requests

Request (HTTP)	Single or Multi Object	Sample URL	Delayed Allowed	Comments
Query (GET)	Single	`[services connector]/[job name]+s/[job ID]`	No	
	List	`[services connector]/[job name]+s`	Yes	Generally only job objects matching the provided fingerprint **should** be returned, however this behavior **should** be configurable so a system health monitor can get telemetry on all jobs
Create (POST)	Single	`[services connector]/[job name]+s/[job name]`	No	
	Multi	`[services connector]/[job name]+s`	Yes	The response to this request looks like the response to a Multi-Create of a standard Object Service.
Update (PUT)	Single	Not Supported	NA	
	Multi	Not Supported	NA	
Delete (DELETE)	Single	`[services connector]/[job name]+s/[job ID]`	No	
	Multi	`[services connector]/[job name]+s`	Yes	Note that this must be conveyed as a **HTTP PUT** with the HTTP **Header methodOverride set to DELETE,** as with standard Object services.

Working through the Phases

One of the great advantages of choosing SIF to build a Functional Service is you do not have to start from scratch, the same operations used throughout SIF 3 **should** be leveraged. This allows the Consumer access to each phase and (when permitted) state to follow strong and meaningful naming conventions. The rights listed for each phase in the Job Object notify the Consumer what operations are available to it. The data model being used defines any payloads needed for a phase; this data model is specified in the Binding Document for the Functional Service.

If a phase state can be updated by the Consumer, posting to the Phase endpoint creates a new state, much like it would a single object. The actual result (state or error) is included in the object issued in response; the Job Object on the provider is also updated accordingly. Each functional service should define in its Binding Document the expectations under which an Adaptor is responsible for updating the phase status: in some case the transition to a state needs to be triggered by the Consumer, in others there needs to be an automated transition made by the Provider.

URL Form

```
[services connector]/[job name]+s/[job id]/[phase name]
```

Example URL

```
https://example.a4l.org/jobs/newIntakes/e03bf3f0-0152-1000-007f-00505686707f/
createLearners
```

Supported Phase Operations

Note that “Phase Operations” really relate to the payload that is conveyed in the phase request. The allowed operations are specified in the “rights” element of each phase within the Job Object. Further the concept of “Single” or “Multiple” object operations do not really apply, as a phase object/payload is specified in the binding document of each locale’s functional services. Each operation may have a payload in the request (except HTTP GET) as well as in the response. The binding document may also define the HTTP status codes returned for each operation, however the default HTTP status code for success is provided in the table below.

All phase requests may be ’immediate’ or ’delayed’, even though they only involve a single job. Again, where allowed, ’delayed’ support is mandatory for brokered environments.

Request (HTTP)	Sample URL	Delayed Allowed	Comments
Query (GET)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	This will return the payload that is defined in the binding document for the given phase. Default success HTTP Status Code of Response: 200 – OK: Data is returned 204 – No Content: No data is returned
Create (POST)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	The payload of the request and response is an object defined in the binding document applicable for the functional service. Default success HTTP Status Code of Response: 201 – Created
Update (PUT)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	The payload of the request and response is an object defined in the binding document applicable for the functional service. Default success HTTP Status Code of Response: 200 – OK
Delete (DELETE)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	Note that this must be conveyed as a **HTTP PUT** with the **HTTP Header methodOverride set to DELETE** as with standard object services. The payload of the request and response is an object defined in the binding document applicable for the functional service Default success HTTP Status Code of Response: 204 – No Content
Events (POST)	Not Supported	NA	Events on the payload of a phase object is not supported.

* Where allowed, delayed support is mandatory for brokered environments.

States of Phases

Just as a Consumer can issue phase requests, to query the current phase and to initiate phase transitions, a Consumer can also state requests, to query the state(s) within a phase, and to initiate state transitions within a phrase. Note that a Consumer can only modify phase states, but not the overall Job state. For example, a Consumer can determine that the current Phase is complete; but it is up to the Provider to determine whether that means the overall Job is complete, or whether other Phases or conditions need to be satisfied.

Unlike phase requests, there is no support for rich payloads in state requests: the only payload defined is the identity of the state that the Consumer wishes the Job to transition to. All requests are 'immediate'; however, that does not mean that the POST request to change state, in particular, will be actioned immediately. The Consumer will need to keep polling the Provider about what state it is currently in or subscribe to the Events feed for the associated Job Object.

State URL Form

```
[services connector]/[job name]+s/[job id]/[phase name]/states/state
```

Example State URL (create a state in a particular phase)

```
https://example.a4l.org/jobs/newIntake/e03bf3f0-0152-1000-007f-00505686707f/
createLearners/states/state
```

Example State Payload

```
<state xmlns="http://www.sifassociation.org/infrastructure/3.3">
  <type>INPROGRESS</type>
</state>
```

Example State Response

```
<state xmlns="http://www.sifassociation.org/infrastructure/3.3">
      <type>INPROGRESS</type>
      <created>2016-05-29T12:45:10+01:00</created>
</state>
```

Since the newest state in "states" is the current state of the corresponding phase, keeping track of when each one was created is important.

Status Codes

Code	Meaning
NOTAPPLICABLE	The Provider will not be doing anything with this phase. If the Consumer likes, it can safely be SKIPPED in this instance.
NOTSTARTED	Based on the functional service definition, the appropriate Adaptor **should** start this phase at (by creating a status of INPROGRESS) at the appropriate time.
PENDING	This status indicates one Adaptor is waiting on the other, before the phase can be continued or COMPLETED.
SKIPPED	Indicates that one Adaptor did *not* need a particular step performed.
INPROGRESS	At least one Adaptor is actively fulfilling the requirements of a phase.
COMPLETED	The phase ended successfully.
FAILED	While processing the phase at least one Adaptor encountered an unrecoverable error.

Supported State Operations

Request (HTTP)	Single or Multi Object	Sample URL	Comments
Query (GET)	Single	Not supported	Since a phase's state has no unique identifier, all queries must be for multiples. Since only the states for a single phase of a single job will be returned, there is no reason to support delayed or batched query operations Default success HTTP Status Code of Response: 200 – OK: Data is returned 204 – No Content: No data is returned
	List	`[services connector]/[job name]+s/[job id]/[phase name]/states`	
Create (POST)	Single	`[services connector]/[job name]+s/[job id]/[phase name]/states/state`	This constitutes a request by the Consumer for the Provider to add and transition the current phase state to the state nominated in the payload.

			Default success HTTP Status Code of Response: 201 – Created. The state has been added/updated.
	Multi	Not supported	
Update (PUT)	Single	Not Supported	
	Multi	Not Supported	
Delete (DELETE)	Single	Not Supported	
	Multi	Not Supported	
Events (POST)	Multi	Not Supported	

Permission Chain

There are several ACLs applicable for functional services.

The first level is the ACL that describes the ACL for a specific functional service and is maintained in the environment of an adapter. The environment provider (e.g. a SIF3 Broker) only knows about these ACLs and can verify if a particular consumer has appropriate rights.

The next level of ACLs applies to phases of a functional service. The provider of the functional service will maintain these and return appropriate phase ACLs to the consumer as part of the "Create" operation of the functional service/job. It is the responsibility of the functional service Provider to check access to phases according to the ACLs of each phase. The environment provider is NOT aware of phase ACLs; hence the environment provider cannot check ACLs for phase operations. However, the environment provider can check the access to a functional service since that is controlled at the environment.

Further, a Consumer has only access rights to phases of jobs it has created. In other words, a consumer can only access phases if it also has the "CREATE" permission for the applicable functional service. If a phase operation is requested by a Consumer that does not have "CREATE" permission for a functional service, the environment provider must assume that the consumer cannot access any phase operations, hence should reject the phase request.

The final level of ACLs applies to states within phases of a functional service. Everything that applies to phase operation permissions is also true for phase state operation permissions.

Understanding the Results

After any exchange, but particularly the data exchanges, an error may be received. One of the advantages of a functional service is it can be defined such that it uses transactional behavior. That is, failure triggers the undoing of everything done under the umbrella of the job by the one or more recipients of data. Of course, other behaviors can also be defined, such as "continue on error."

If a non-SIF payload is exchanged, the error would be scoped to only cover the exchange and not any secondary processing.

If the phase processing is asynchronous and later fails, the status of the phase in the Job object would be set to 'Failed' and a payload related error message would be made available in the 'receipt' phase.

Example Error

```
<error id="cf9a1bcd-0152-1000-007f-00505686707f"
xmlns="http://www.sifassociation.org/infrastructure/3.3">
  <code>409</code>
  <scope>createLearners</scope>
  <message>Duplicate learner detected.</message>
</error>
```

Knowing the Phase and/or Job is Done

One of the surprising inherent issues of sending data is letting the receiver know it has all the data. Sometimes the desire is to let the receiver know not that it has all the current data, but rather that it has all the data in its final/official form. Fortunately, since a Job Object keeps track of both job and phase statuses, both of these issues are inherently covered.

For this to work, the system sending the data has to signal completion step-by-step. Depending on the functional service definition, by moving onto the next phase or creating a new state for

the corresponding phase, this can be accomplished. If this happens unexpectedly, one or more errors may need to be returned.

Monitoring the Job

If the creating Consumer wishes to receive Events about the state of the Functional Service it MUST subscribe to the functional service, before creating the job object (or some events may be missed). However, it may choose to occasionally read the Job Object after its creation instead.

```
<subscription xmlns="http://www.sifassociation.org/infrastructure/3.3">
  <zoneId>BSD_Current</zoneId>
  <serviceType>FUNCTIONAL</serviceType>
  <serviceName>xSRE</serviceName>
  <queueId>7976dee0-9717-11e6-ae22-56b6b6499611</queueId>
</subscription>
```

In a Brokered Architecture the Broker will add the source environment's fingerprint header to all job creation requests. The service provider will then include that fingerprint in any related service event intended for the owner of the service. Events published without specifying a fingerprint will be delivered to all subscribed consumers.

Supported Job Events

Generally, Object Services have only complete objects to operate on. With Requests for Functional Services that is very different: the Object is populated incrementally, as the Job proceeds through various states and phases. The Job Object may be changed only indirectly, through updates triggered through Consumer Phase operations, Consumer State creation requests, and the Provider's own logic. However, events are only supported for the Job Object! It is important to note the end-point of the Job Events is the standard event connector, using HTTP POST, and not the service connector.

Event Type	Usage/Constraints
CREATE	Sent upon job creation. Typically, in response to servicing a job creation request successfully.

UPDATE	Sent upon change in the Job Object. Generally used to track the progress of a Job, sometimes causing the receiving Consumer to act.
DELETE	Send upon Job deletion. Adaptors **should** use this to clean up the specified Job.

Cleaning Up or Keeping a Record

Another built-in piece of SIF Functional Services is that job objects (and their attached data) can be kept, archived, or deleted by the client, or even held for a time then removed by the server. Implementers should let their use case dictate how to handle retention of Job Objects and specify behavior explicitly in the service Binding Document. Just make sure everyone involved understands so the outcome is clean.

