Access 4 Learning Community

Powered by: SIF

Simple. Secure. Scalable. Standard.

SIF Infrastructure Specification 3.2.1: Functional Services

www.A4L.org

Version 3.2.1, March 2016

Introduction

Functional Services are conceptually simple, but the many application uses can make a general understanding of them elusive at the same time. Broadly speaking, a Functional Service coordinates one or more data exchanges such that they have a beginning, middle, and an end.

A Functional Service consists of:

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

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 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 request.
2. The Provider updates the Job Object, sometimes as a result of a Consumer request.
3. Either the Consumer or Provider may delete the Job Object, depending on the service definition.

XPath	Chars	Description	Type
/job/@id			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
/job/stateDescription	O	A descriptive message elaborating on the current state, e.g. if the current state is "FAILED" the stateDescription may be "Timeout occurred".	string
/job/created	O	The datetime this job was created.	dateTime
/job/lastModified	O	The datetime this job was last modified.	dateTime
/job/timeout	O	The amount of time after creation before	duration

		this job is automatically deleted.	
/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
/job/phases/phase/states/state/created	O	The datetime this job was created.	dateTime
/job/phases/phase/states/state/lastModified	O	The datetime this job 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 "FAILED" the stateDescription may be "Timeout occurred".	string
/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	string
/job/phases/phase/rights/right	M	The Access Right	string

/job/phases/phase/statesRights/right/@type	M	The type of the Access Right	string
/job/phases/phase/statesRights/right	M	The Access Right	string
/job/initialization/phaseName	O	Name of initialization phase the payload corresponds to.	token
/job/initialization/payload	M	Information matching 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

```
<job xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.sifassociation.org/infrastructure/3.2.1"
 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>  <!-- Other states omitted for brevity. -->
          <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>  <!-- Other states omitted for brevity. -->
          <type>COMPLETED</type>
          <created>2016-03-09T09:00:02</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>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 intent, status, and results of the Functional Service, 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. Importantly a functional service addresses a particular user story. All parties need to follow their defined role in order for interoperability to be accomplished.

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/3.2.1">
  <name>newIntake</name>
</job>
```

Response

The flushed out object similar to the example in the proceeding section is returned.

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 return, however this behavior **should** be configurable so a system health monitoring 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.

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

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 Consumer access to each phase and (when allowed by states rights) state to follow strong and meaningful naming conventions. The rights for each phase notify the Consumer what operations are available to it. The data model being used defines any payloads needed for a phase. For states updatable by the Consumer, it creates a new state much like it would a single object. The actual result (state or error) is included in the response. Each functional service should define the expectations for which Adaptor is responsible for updating the phase status.

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. 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 is specified in the binding document of each locale’s functional services.

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 and is a locale's data model object.
Create (POST)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	The payload is an object defined in the locale's data model.
Update (PUT)	`[services connector]/[job name]+s/[job id]/[phase name]`	Yes	The payload is an object defined in the locale's data model.
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.
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.

State URL Form

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

Example State URL

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

Example State Payload

```
<state xmlns="http://www.sifassociation.org/infrastructure/3.2.1">
  <type>INPROGRESS</type>
</state>
```


Example State Response

```
<state xmlns="http://www.sifassociation.org/infrastructure/3.2.1">
      <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 uniquely 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
	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`	Creating a state adds it to the corresponding list of states.
	Multi	Not supoprted	
Update (PUT)	Single	Not Supported	
	Multi	Not Supported	
Delete (DELETE)	Single	Not Supported	
	Multi	Not Supported	
Events (POST)	Multi	Not Supported	

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 a "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 available in the 'receipt' phase.

Example Error

```
<error id="cf9a1bcd-0152-1000-007f-00505686707f"
xmlns="http://www.sifassociation.org/infrastructure/3.2.1">
  <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 not let the receiver know 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.

Of course 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 would like 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 chose to occasionally read the Job Object after its creation instead.

```
<subscription xmlns="http://www.sifassociation.org/infrastructure/3.2.1">
  <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 Job Object may be changed only indirectly, through: Phase operations, 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 the 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 cleanup the specified Job.

Cleaning Up or Keeping a Record

Another great 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. Let your use case rule how you handle retention of Job Objects. Just make sure everyone involved understands so the outcome is clean.

