



## **SIF 3.0.1 Infrastructure Services**

## Table of Contents

<b>1.</b>	<b>Introduction .....</b>	<b>4</b>
1.1.	Required Reading.....	4
1.2.	Infrastructure Functionality Summary.....	4
1.3.	Infrastructure Service “uniqueness” .....	6
1.3.1	Infrastructure Service Operation Invocation.....	6
1.3.2	Infrastructure Service Query Requests .....	7
1.3.3	Other Infrastructure Service Requests .....	7
1.3.4	Infrastructure Service Events.....	7
1.3.5	Infrastructure Service Scoping.....	7
<b>2.</b>	<b>Message Header Differences .....</b>	<b>8</b>
2.1	Universal Header Elements .....	8
2.2	Request Header Elements .....	8
2.3	Response Header Elements .....	9
<b>3.</b>	<b>The Consumer and its Environment.....</b>	<b>10</b>
3.1	Environment Scalability .....	10
3.2	Environment Initialization .....	11
<b>4.</b>	<b>Environments Service.....</b>	<b>13</b>
4.1	Consumer Authentication.....	14
4.1.1	Authentication Exchange Sequence .....	15
4.1.2	Authentication Elements .....	15
4.1.3	Authentication Methods.....	16
4.1.4	Authentication Validation: BASIC .....	17
4.1.5	Authentication Validation: SIF_HMACSHA256.....	18
4.1.6	Client Certificates: Best Practices .....	20
4.2	Session Creation .....	20
4.2.1	Generating subsequent Authorization Tokens.....	21
4.3	Overview of Basic Operations.....	21
4.3.1	Create Environment .....	22
4.3.2	Create Environment Response.....	26
<b>5.</b>	<b>Consumer <i>Environment</i> .....</b>	<b>27</b>
5.1	Overview of Basic Operations.....	27
5.2	Query .....	27

5.2.1	Global Elements .....	28
5.2.2	Application-specific Information .....	30
5.2.3	Infrastructure Service Assignment.....	31
5.2.4	Service Provisioning .....	33
5.3	Delete.....	36
<b>6.</b>	<b>Provision Requests .....</b>	<b>38</b>
6.1	Data Elements .....	38
6.2	Overview of Basic Operations.....	40
6.2.1	provisionRequests.....	40
6.2.2	provisionRequest.....	41
6.3	Lazy Authorization .....	41
<b>7.</b>	<b>Requests Connector .....</b>	<b>44</b>
7.1	Overview of Basic Operations.....	44
7.2	Identification of Service Provider .....	44
7.3	Secure Request / Response Routing .....	45
7.3.1	Consumer to Requests Connector .....	45
7.3.2	Internal to Requests Connector .....	45
7.3.3	Requests Connector to Provider Queue .....	46
7.3.4	Provider Response .....	46
7.3.5	Requests Connector to Consumer .....	46
7.4	Data Structure.....	47
<b>8.</b>	<b>Events Connector.....</b>	<b>48</b>
8.1	Overview of Basic Operations.....	48
8.2	Security, Replication and Delivery to Subscribing Consumers .....	48
8.3	Data Structure.....	49
<b>9.</b>	<b>Queues, Queue Instance and Queue Messages .....</b>	<b>50</b>
9.1	Service Operations .....	50
9.2	Queue Instance Object.....	52
9.3	Message Service .....	55
9.3.1	“Get Next and Pop” (Query with deleteMessageld).....	55
9.3.2	Long Polling .....	56
9.3.3	Support for multiple concurrent Messages Service Connections (optional) .....	56
9.3.4	Support for “Wake Up” .....	58

<b>10. Subscriptions</b>	<b>59</b>
10.1 Supported Operations	59
10.2 Subscription Data Elements	59
10.3 XML Examples	60
<b>Appendix A: The Role of the Environment Administrator</b>	<b>61</b>
A.1 Setting up the Environments	61
A.2 Administrating the Environments	61
<b>Appendix B: Connection Topologies</b>	<b>62</b>
<b>Appendix C: HTTP Codes and Header Fields</b>	<b>65</b>
C.1 HTTP Success Codes	65
C.2 HTTP Error Codes	65
C.3 HTTP Header Fields	66
C.4 URL Matrix Parameters:	68
C.5 URL Query Parameters:	68
<b>Appendix D: Common Types</b>	<b>70</b>
D.1 ErrorType	70
D.2 UUIDType	70
D.3 VersionType	71
D.4 PropertyType	71
D.5 ProtocolType	71
D.6 ProductIdentityType	72
D.7 InfrastructureServiceNameType	72
D.8 UtilityServiceNamesType	73
<b>Appendix E: Acknowledgements</b>	<b>74</b>

## 1. Introduction

All Service Consumers and Service Providers in a SIF 3.0 Environment interact only through and with the Environments Provider Interface. The implementation of this interface provides the infrastructure underlying the SIF 3.0 Environment, and is defined at three levels:

- The “Direct” Environments Provider API comprises the minimum set of functionality that must be offered to Service Consumers in environments where they connect directly to the implementation providing the Data Object Services.
- The “Brokered” Environments Provider API represents the minimum set of functionality that must be offered to Service Consumers and Service Providers in a more featureful “environment where Object Service Consumers and Providers are all connected through a central “Message Broker” middleware component such as an Enterprise Service Bus (ESB).
- The “Administrative” Environments Provider API includes an additional set of functionality that is a natural extension to the Brokered Environment API and which could be useful to Administrative applications monitoring and controlling the set of Consumers and Providers interoperating together to provide a SIF 3.0 solution.

While outlined here, this third level will not be explicitly documented as in many cases the specific details are internal to the Environments Provider’s implementation rather than its external interface.

### 1.1. Required Reading

Please refer to the SIF 3.0 *Basic Architecture* document for an understanding of the terminology, concepts global XML element definitions, Service types, operation descriptions and exchange choreographies that will be referred to here.

A working knowledge of that document is assumed as a prerequisite, as the Infrastructure Services will utilize the Service framework described there.

### 1.2. Infrastructure Functionality Summary

All Infrastructure Services are directly connected to every Service Consumer, rather than accessed through the Requests Connector Service. They follow the standard Data Object Service API in that they support some or all of the defined Object Service Provider interface methods (Query, Create, Update, Delete).

The generic functionality provided by each Infrastructure Service in supporting the Environments Provider API is shown in the following table. Each will be described in detail in the sections that follow.

Infrastructure Service	Functionality
<b>environments (the initial URL address)</b>	The starting point of an application’s interactions with the Environments Provider. It authenticates the registering application using a preset list of “authorized” Service Consumers. When successful, it provides the application with a (possibly customized)

	<p>consumer Environment. Specifically it:</p> <ul style="list-style-type: none"> <li>• Validates support for application interoperability requirements (Transport, Locale, DM and Infrastructure Versions) and Application key.</li> <li>• Creates and returns a customized “Environment” for the application</li> <li>• Returns an Authentication Token to be used on all subsequent Service Requests in that Environment.</li> </ul> <p>This is accomplished through a single operation (<i>create Environment</i>) which is invoked when an application wishes to Register as a Consumer.</p>
<b><i>environment</i></b>	<p>The Consumer’s gateway to all Services (Infrastructure, Utility, Data Object and Functional).</p> <p>The Environment Service provides the set of (possibly customized) information the Consumer needs to interoperate successfully, including:</p> <ul style="list-style-type: none"> <li>• Environment ID and type (ex: “Production” or “Test”)</li> <li>• URLs to the full set of Infrastructure Services supporting this Environment (the remaining rows of this table).</li> <li>• The ID of the application’s default Zone in the Environment.</li> </ul> <p>The main section also contains the complete set of authorization (provisioning) rights to the currently approved set of operations (qualified by Zone, Context and Service name) it can issue.</p> <p>If the Consumer is further authorized to be a Provider of one or more Services (Brokered Environments only) that is indicated in the provisioning information.</p> <p>The <i>environment</i> is read-only.<sup>1</sup></p>
<b><i>provisionRequests</i></b>	<p>Creating a <i>provisionRequest</i> extends the Consumer’s <i>environment</i> by:</p> <ul style="list-style-type: none"> <li>• Setting a new default Zone value</li> <li>• Requesting extensions to the Consumer’s current set of authorized per-Zone / per-Context / per-Service operation rights.</li> </ul>
<b><i>queues and queue</i></b>	<p>A Consumer <b>may</b> create one or more <i>queue</i> instances using the <i>queues</i> service and then <b>must</b> associate one of those instances with every Event Type subscription during Consumer provisioning, and specify one as part of every Delayed Request it issues.</p> <p>Each <i>queue</i> Service instance guarantees <i>first in first out</i> (FIFO) delivery of the oldest stored asynchronous Event or individual delayed Provider Response when an explicit</p>

<sup>1</sup> There may be other information which the Environments Provider is maintaining for the Consumer, such as XML Filtering rules.

	Consumer <i>query</i> Request arrives for the next message.
<b><i>subscriptions</i></b>	Creating a <i>subscription</i> which specifies a Service Provider and a Queue will result in the subsequent asynchronous delivery of all published Events from that Service Events to that Queue.
<b><i>requestsConnector</i> (the URL used by a Consumer to access all Non-Infrastructure Services)</b>	<p>This is the Infrastructure Service that connects a Registered Consumer to all non-Infrastructure Service Providers. It is the URL to which the Consumer issues all Data Object, Functional and Utility Service Requests (i.e. Requests for services other than the Infrastructure Services described in this section).</p> <p>The Requests Connector Service securely routes (and possibly filters):</p> <ul style="list-style-type: none"> <li>• Consumer Requests to the selected Service Provider</li> <li>• Service Provider Responses back to either: <ul style="list-style-type: none"> <li>• The open HTTP connection over which the Consumer request was originally issued (for Requests requiring immediate Responses).</li> <li>• The particular <i>queue</i> instance specified by the Consumer (for Delayed Responses)</li> </ul> </li> </ul>
<b><i>eventsConnector</i> (the URL used by a Service Provider to post its Events)</b>	<p>This is the Infrastructure Service that connects a Provider of one or more Services to all its subscribing Consumers in a Brokered Environment. It is the URL to which the Provider publishes all its Events reflecting changes in the data of the objects it provides.</p> <p>The Events Connector Service maintains the list of subscribers for each Event type and replicates and securely routes (and possibly filters) the published Event to each of the assigned Subscriber Queues.</p>

### 1.3. Infrastructure Service “uniqueness”

While the above Infrastructure services all conform to the Object Service interface, they differ from other Services in several important ways.

#### 1.3.1 Infrastructure Service Operation Invocation

When a Service Consumer invokes an operation on an Infrastructure Service, it does so using a URL *unique* to that Infrastructure service. When a Service Consumer invokes an operation on any other type of Service (including Utility Services) it uses the URL of the Requests *Connector* Infrastructure Service<sup>2</sup> which securely routes the invocation to its intended destination.

<sup>2</sup> In a Brokered Environment, the Connector Service URL corresponds to the SIF 2.x ZIS URL, and provides a gateway to the routing and guaranteed delivery services of the Message Broker middleware. In a Direct Environment, this is typically the URL corresponding to an internal Service queue which accepts incoming requests, contained within the application implementing the Direct Environments Provider Interface.

### **1.3.2 Infrastructure Service Query Requests**

Since every Consumer is directly connected to its set of Infrastructure Services, all Infrastructure Service Requests are “immediate”, and the Responses are always synchronously returned on the same HTTP connection. This implies (for all Infrastructure Services other than the Requests Connector):

- Only “immediate” mode Query Requests are legal (where the Response is returned as the HTTP Response to the HTTP Request).
- There is no XQuery functionality and no “object ranges” involved in Query Requests to an Infrastructure Service. The contents of the entire object will always be returned in the immediate Response.

### **1.3.3 Other Infrastructure Service Requests**

The API simplifications on the remaining Request types reduce the complexity required of Environments Provider Implementations, and reflect common middleware conventions.

- Unless specifically noted, only the singular forms of the Create, Update and Delete methods may be issued to an Infrastructure Service.
- Since all Requests are immediate, a Queue Service instance will not be specified in any type of Infrastructure Service Request. This allows even simple, synchronous Consumers to fully utilize these Services.

### **1.3.4 Infrastructure Service Events**

Infrastructure Services can be viewed as synchronous web services, which together provide the underlying framework for the complete Environments Provider Interface. All Requests have immediate Responses, and Infrastructure Services do not publish Events.

### **1.3.5 Infrastructure Service Scoping**

With the exception of the *queue*, all Infrastructure Services are Environment-wide in the sense that as far as the Consumer can tell, there is only one Infrastructure Service of each type present, and it is independent of Zones and Contexts. Neither a Context nor Zone identifier is ever present in any request posted to an Infrastructure Service.

The URLs of all Infrastructure Services accessible by a Consumer are contained within the Consumer's *environment*.



## 2. Message Header Differences

The following tables describe the differences between the header elements of Requests, Responses and Event messages exchanged with Infrastructure Services and those exchanged with all other Service types. They are a reflection of the unique aspects of Infrastructure Services listed above.<sup>3</sup>

**Important point:** *The XML definitions in this section describe “message related” elements, some of which on a given transport (ex: REST) may be expressed in the URL or the HTTP message header, in non-XML form. Their values and meanings however remain unchanged.*

### 2.1 Universal Header Elements

There are no header differences between an Infrastructure Service and a Utility or Data Object Service.

**Example:**

The following XML instance fragment illustrates the formats of header elements in a message being exchanged between a Service Consumer and an Infrastructure Service Provider.

```
<environmentId>tidewater.virginia.edu/sif/environment/Norfolk/3.0</environmentId>

<infrastructureVersion>3.0</infrastructureVersion>

<timestamp>2012-10-24T15:58:33.984Z</timestamp>

<messageId>12345678-1234-4567-1234-567812345abc</messageId>
```

### 2.2 Request Header Elements

These differences apply specifically to all Requests sent by a Consumer directly to any Infrastructure Service URL. As the URL determines the end-point Service Provider rather than the message header, the normal Zone / Context / Service Type / Service Name qualifiers are also not needed.

Element	Description
zoneId	Is omitted. <sup>4</sup>
contextId	Is omitted.

<sup>3</sup> Please refer to the corresponding element definition tables in the SIF 3.0 Base Architecture document for the general definitions of all elements being contrasted.

<sup>4</sup> This is not true for Utility Service requests, which go through the connector. These have a zoneId value of “environment-global” and like the Infrastructure Services are all “environment-wide” and universally available to all Consumers. The *environment-global* Zone is the single qualifier for where they may be found, is present in all SIF 3.0 Environments, and all contained Services have only the “DEFAULT” Context. Only Utility Services will ever be assigned to the *environment-global* Zone.

<b>serviceType</b>	Is omitted
<b>serviceName</b>	<p>Mandatory. The value of the token <b>must</b> be one of the enumerated values in the InfrastructureServiceNameType. Specifically:</p> <ul style="list-style-type: none"> <li>• environment</li> <li>• requestsConnector</li> <li>• eventsConnector</li> <li>• provisionRequests</li> <li>• provisionRequest</li> <li>• queues</li> <li>• queue</li> <li>• subscriptions</li> </ul>
<b>requestType</b>	If included, must be set to “immediate” (which is also the default value).
<b>requestAction</b>	Unchanged in meaning

### Example:

The following XML instance fragment illustrates the formats of all message header elements as they would be represented in for example a *create* operation issued to *provisionRequests* to provision the Consumer as a subscriber to events from a specified Provider.

The actual destination is determined by the URL assigned to the *provisionRequests* Infrastructure Service in the Consumer’s *environment*. The HTTP Request is sent there, and the synchronous HTTP Response will convey the URL of the newly created *provisionRequest*.

```

<timestamp>2012-10-24T15:58:33.984Z</timestamp>

<messageId>12345678-1234-4567-1234-567812345abc</messageId>

<requestAction>create</requestAction>

```

## 2.3 Response Header Elements

There are no differences which apply specifically to the header elements of all Responses returned by an Infrastructure Service. All header element meanings are identical to those in Immediate Responses returned by any other Service type.

### 3. The Consumer and its Environment

This section describes the sequence of operations that support the process of initially registering an application as a Service Consumer, the range of “environment” functionality offered after successful registration, and how that functionality might be used by both a low end and high end Consumer.

#### 3.1 Environment Scalability

The Environment provided to a successfully registered SIF 3.0 Consumer can range (aside from Utility Services) from a single Zone providing only a single Object Service, all the way up to one encompassing multiple Zones, which organize, manage and connect multiple Object and Functional Service Providers and Consumers across an entire educational Organization.

One of the primary goals of the SIF 3.0 infrastructure is to ensure that the functionality of the minimal Environment is a clean and precise subset of larger ones, so that even a Consumer application deployed on a mobile device can:

- Execute independently of the richness of the *environment* with which it is supplied
- Remain simple (i.e. *environment* defaults allow it to ignore Infrastructure and Utility Services it does not need to use)

The table below shows the usage of each of the SIF 3.0 Infrastructure Services by both a minimal Service Consumer, and one which relies on and takes advantages of the complete functionality offered by a “high end” Brokered Environments Provider.

Infrastructure Service / URL	Minimal Consumer Usage	Full-function Consumer Usage
<b>environments</b>	<p><b>Must issue a Create</b> to the <i>environments</i> URL containing the application's identification and its transport requirements.</p> <p>On success, it gets back its “<i>environment</i>” and a “permanent” Authorization Token which must be used on all subsequent Service Requests.</p>	Identical usage. However based upon its identification, a more featureful <i>environment</i> may be created and returned than for other prospective Consumers.
<b>environment</b>	<p>The <i>environment</i> returned in the create response has the URLs of the other Infrastructure Services comprising and supporting it, along with the Consumers default Zone ID, and initial service operation authorization rights.</p> <p><b>If the Consumer assumes the authorization rights it needs are either pre-provisioned, or will be granted when it issues its Requests, there is no need to access this Service once the Infrastructure URLs are obtained.</b></p>	Examines the <i>environment</i> returned, and uses it as the basis for creating an additional <i>provisionRequest</i> , which after completion requires reexamination of the total <i>environment</i> to see effected changes.
<b>provisionRequests</b>	In the simplest case, the Consumer can assume its environment already contains pre-authorization of all the Requests it needs.	These are created to extend the Consumer's default provisioning to include authorization to issue Requests and subscribe to Events from Object and Functional Services in non-default

	<p>This would allow it to issue requests as needed, logging an error and then either creating a <i>provisionRequest</i> or exiting if an authorization violation occurs.</p> <p><b>If site administration policies support “pre-authorization” of access rights and the Consumer does not subscribe to Events, no <i>provisionRequest</i> need ever be created.</b></p>	<p>Zones and Contexts.</p> <p>Exploring the Consumer Environment for other than pre-allocated Services might require the Consumer to <i>query</i> both the <i>zones</i> and <i>providers</i> Utility Services<sup>5</sup>.</p> <p>If the Consumer is further authorized to be a Service Provider (Brokered Environments only), it must be authorized to issue <i>create</i> requests to the <i>providers</i> Utility Service Registry.</p>
<b>queues and queue</b>	<p>A minimal Consumer may issue only “immediate” Requests, which result in synchronous Responses on the same HTTP connection (nothing is queued). In that case, only Events will arrive asynchronously.</p> <p><b>If all Requests are immediate and no Events are subscribed to, then the Consumer does not need to provision / access any dedicated <i>queue</i> Service.</b></p>	<p>A high end Consumer may utilize the <i>queues</i> service multiple Queue Services in a variety of ways to achieve maximum scalability. One common usage pattern might be to create multiple <i>queue</i> service instances and:</p> <ul style="list-style-type: none"> <li>• Subscribe to “important” Provider Events with Queue A</li> <li>• Subscribe to lesser (or overly chatty) Provider Events with Queue B</li> <li>• Issue any Requests for Delayed Responses with Queue C</li> </ul> <p>Then Queue C can be polled for important delayed responses, the important Events in Queue B can be polled for next, and the lesser Events in Queue A polled for only when all other message sources are “quiet”</p>
<b>requestsConnector</b>	All Data Object, Utility and Functional Service operations are invoked here.	Identical usage.

### 3.2 Environment Initialization

As noted in the table above, a simple application of the sort that resides on a mobile device would need to successfully issue only a single *create* Environment Request to become a registered Consumer that then could start issuing authorized create, query, update and delete Requests to the set of Data Model-specific Object and Functional Services that were made accessible to it in its *environment*.

A more complex Consumer could additionally issue requests to the set of accessible Utility Services to:

- Create one or more *subscription* objects to explicitly subscribe to Events of a specified Service.

<sup>5</sup> Please refer to the Utility Services document for details on these and other Utility Services.

- Create one or more *queue* Service instances which it can use to synchronously poll for asynchronously arriving messages (Events and / or Responses to Delayed Requests).
- Create or use one or more fixed Query Templates within the Query Template Utility Service as the basis of subsequent *query* operations.
- Query the *Providers* and *Zones* Utility services for additional Service Providers (possibly in other Zones) that it may wish to access. Provision itself to invoke operations on them by creating a *provisionRequest* with additional authorization assertions.
- Create a new *Provider* entry, effectively making itself a Service Provider, capable of receiving and responding to Requests from other Consumers, and publishing its own Events (Brokered Environments only).
- Create an *Event* for an object type it provides which will be published to all subscribers.
- Utilize the *Alerts* Utility Service to generate error or warning messages as needed.

The remainder of this document will describe each of the Infrastructure Services which support these processes in further detail.

## 4. Environments Service

The environments Service is the starting point of a client application's interactions with the SIF 3.0 Environments Provider, and its service URL is one piece of information every candidate SIF 3.0 Consumer must have knowledge of.<sup>6</sup> The elements comprising the Environment object are shown below. They will be explained in more detail in the sections that follow.

```
<environment id="" type="">
  <sessionToken></sessionToken>
  <solutionId></solutionId>
  <defaultZone></defaultZone>
  <authenticationMethod></authenticationMethod>
  <instanceId></instanceId>
  <userToken></userToken>
  <consumerName></consumerName>
  <applicationInfo>
    <applicationKey></applicationKey>
    <supportedInfrastructureVersion></supportedInfrastructureVersion>
    <transport></transport>
    <applicationProduct>
      <vendorName></vendorName>
      <productName></productName>
      <productVersion></productVersion>
      <iconUri></iconUri>
    </applicationProduct>
    <adapterProduct>
      <vendorName></vendorName>
      <productName></productName>
      <productVersion></productVersion>
      <iconUri></iconUri>
    </adapterProduct>
  </applicationInfo>
  <infrastructureServices>
    <infrastructureService name=""></infrastructureService>
  </infrastructureServices>
  <provisionedZones>
    <provisionedZone id="">
      <services>
        <service name="" contextId="">
          <rights>
            <right type="QUERY"></right>
            <right type="CREATE"></right>
            <right type="UPDATE"></right>
            <right type="DELETE"></right>
          </rights>
        </service>
      </services>
    </provisionedZone>
  </provisionedZones>
</environment>
```

---

<sup>6</sup> The Consumer registration process supports a fairly broad range of administrative policies. Note that there could be multiple *environments* services within a given organization, each supporting a different type of “solution” (ex: Test, Staging, Production), and creating individual Consumer *environment* objects scoped to that solution, by filling them with an entirely different set of infrastructure service URLs.

```

        <right type="PROVIDE"></right>
        <right type="SUBSCRIBE"></right>
        <right type="ADMIN"></right>
    </rights>
</service>
</services>
</provisionedZone>
</provisionedZones>
</environment>

```

## 4.1 Consumer Authentication

When the client application issues a “create” request to the environments Service URL, that service will authenticate the application via the contents of the provided “initial” Authorization Token, which generally can be mapped to a specific application / product. Only certain applications may be pre-approved to even “attempt” to create an Environment. Whether such a given attempt succeeds depends upon the remaining elements in the create request payload.

Some of these elements indicate additional “aspects” of the actual client’s identity, which, taken together, will determine the authorization limits imposed on the resulting Consumer should the create Environment operation be successful. These include:

- The ID of the requesting application and an associated “shared secret” (ex: *RamseyPortal:a1b2c398*)
- Optionally, the requesting “instance ID” of that application. This could represent:
  - The serial number of a mobile device where the Portal application instance is physically installed
  - A logical instance (ex: all users who are students in the middle school)
  - A parallel connection to the Environment made for scalability reasons or other purposes known only to the Client application.
- Optionally, the ID of the requesting user of that application (or application instance) and an associated “shared secret”. This often takes the form of a User ID and Password (ex: *Martin.Smith:boywonder12*).

Security for the shared secret values contained in this message is provided by the HTTPS line protocol over which it must be sent.

If authentication for the “complete Consumer” fails, or if there is another mismatch between client / service expectations (Data Model or Infrastructure versions supported or choice of Authentication Method) an error response **must** be returned.

Otherwise the basic requirements for interoperability have been satisfied. In that case the Environments Provider will register the issuer as a “Consumer”, and return it an *environment* object containing:

- The URLs of the Infrastructure services it can access

- The list of Data Object and Utility service operations it is guaranteed to be pre-authorized to issue.<sup>7</sup>
- An initial “Session Token”

Once the Consumer is successfully registered, a series of request-response exchanges between Consumer and Environments Provider begin. They last up until the time the Consumer “session” ends when the Environment is deleted by either the Consumer, or by the Environments Provider (typically only after a period of inactivity has exceeded some preset limit).

#### **4.1.1 Authentication Exchange Sequence**

Until session expiration, the Consumer may issue multiple requests, each containing an *authorization* “based upon<sup>8</sup>” the *sessionToken* it received in the payload of the response to its original Environment “create” operation. This *authorization* token allows the Environments Provider to identify the registered Consumer and authenticate that it is the issuer of that request, which it will then process directly (or route it along to the appropriate Service Provider) only if the Consumer has been granted the necessary authorization for the operation it is requesting.

This makes the “create” Environment request unique from all subsequent ones, because the *authorization* token that accompanies it cannot be based upon the value of the Environments-provided *sessionToken*, since at the point when it is issued, there is no established session.

The details of the particular SIF Consumer authentication method used at a given site are encapsulated by the following:

- The original value of the *authorization* provided with the Environment *create* request
- The value of the *sessionToken* returned by the Environment Provider in response to that request
- The algorithm defining how the value of that *sessionToken* is used to calculate the *authorization* tokens for all subsequent requests issued by that Consumer.

There are multiple ways to achieve this, and no one technique is mandated in this standard. Rather a flexible framework for generating the permanent *authorization* token and enforcing a variety of secure authorization policies is provided, involving the use of one or more of the following elements contained in the Environment create request payload.

#### **4.1.2 Authentication Elements**

In addition to the “timestamp” element which accompanies every request (in [xs:dateTime](#) format) and which may be used to ensure token uniqueness, the following elements are specifically involved in the request authentication process.

---

<sup>7</sup> Access to Data Object Services not on this list can be requested via issuing a “create” to the *provisionRequests* Infrastructure service, at which point the status of all needed authorization rights will be explicitly determined. Alternatively the desired Data Object request can simply be issued, whereupon the response will determine whether Consumer authorization of that particular request for that Data Object was granted.

<sup>8</sup> The actual details of how the Consumer calculates the new value of the Authorization Token depends upon the details of the Authentication Method value selected in the Environment *create* request payload.



Element	Details
authorization	<p>Accompanies every Environments Provider request.</p> <p>It is supplied initially in the create Environment request payload as a combination of client initialToken and a pre-agreed “shared secret”, which may correspond to the application itself, or to a particular instance of the application (perhaps one tied to a specific mobile device).</p> <p>For all subsequent requests, its value is calculated from the sessionToken returned in the create Environment response. Depending on the pre-agreed authentication method, this calculation may be done only once or independently performed by the Consumer before each new request.</p>
sessionToken	<p>Appears in the payload of a create Environment response, and defines the “session” corresponding to the new Environment object in which all Consumer / Environments Provider interactions will occur.</p> <p>It is an encoded value which incorporates identifying information about the application and optionally the application instance and current user of the application which together constitute the identity of the registered Consumer.</p> <p>Its value is used by the Consumer in calculating all its subsequent <i>authorization</i> values.</p>
authenticationMethod	<p>The identification of the method used to encode the Consumer’s identity and generate the <i>authorization</i>. There are two possible values for this element, at least one of which must be supported by the Environments Provider:</p> <ul style="list-style-type: none"> <li>• BASIC</li> <li>• SIF_HMACSHA256</li> </ul> <p>Each is explained in further detail below.</p>
userToken	If the entity being authenticated is specific to a user, or a combination of user and application instance, this contains the verifiable user identification information
instanceId	If the entity being authenticated is specific to an application instance, this is the application instance ID. It allows an entity with a single shared secret to potentially register itself as multiple uniquely separate Consumers..
applicationInfo/applicationKey	This is the application identification and it forms the basis of constructing the initial <i>authorization</i> token which accompanies the Environment create request.

#### 4.1.3 Authentication Methods

The right to issue a request to become a registered Consumer of an Environments Provider should be restricted to a set of previously determined applications. Whether or not that request is granted may be influenced by evaluating additional information provided by the application concerning the actual “entity” (device, user, or some combination) making this request.

This is reflected in the value of the *authorization* token accompanying the Environment create request, which is based solely upon the information provided in the Application ID, coupled with the authentication method (BASIC or SIF\_HMACSHA256) the application is using<sup>9</sup>.

**BASIC** - A basic authentication method, which is only considered to be secure when used in conjunction with HTTPS as the “shared secret” information at the heart of the authentication is passed across the network.

**SIF\_HMACSHA256** - HMACSHA256 is a more secure authentication method which has the advantage that “shared secrets” are never passed across the network.

In SIF usage the *authorization* token is also salted with the current date and time in UTC ISO 8601 format (contained in the timestamp element) so it changes on every request to avoid replay attacks. Since the timestamp element is present every request header, any time synchronization dependency between the Consumer and the Environments Provider is avoided.

In terms of the following illustrative example, assume:

- The agreed upon name of the requesting application is RamseyPortal, and its “shared secret” value with the Environments Provider is *a1b2c398*.
- The Application Instance ID is *District7*
- The name of the Student from District 7 for who this request is actually being made is *Martin.Smith* and his password is *boywonder12*
- The time at which the request is sent (contained in the message header) is: *2013-06-22T23:52-07*

#### 4.1.4 Authentication Validation: BASIC

In the create Environment request, assuming Basic authentication

Element	Value	Meaning
authenticationMethod	<i>BASIC</i>	Basic Authentication
consumerToken	<i>RamseyPortal:a1b2c398</i>	Application ID and shared secret information
authenticationToken	<i>Basic UmFtc2V5UG9ydGFsOmExYjJmZk4DQo=</i>	The string “Basic ” followed by the base64 encoded equivalent of the consumerToken
instanceId	<i>District7</i>	Qualifier to the Application Key. This allows the same application to register itself as a Consumer multiple times without conflict.
userToken	<i>Martin.Smith:boywonder12</i>	Verifiable user identification (in this case the unencoded concatenation of user identification and password)

<sup>9</sup> For further information on Basic Authentication, please refer to <http://tools.ietf.org/html/rfc2617>. For further information on HMACSHA256, please refer to <http://tools.ietf.org/html/rfc4868>

## Consumer

The *authorization* value is calculated in the following way:

Operation	Example Value
Concatenate the applicationKey and shared secret and separate them by a “.”	RamseyPortal:a1b2c398
Base64 encode and prefix with the authentication method and a space	Basic UmFtc2V5UG9ydGFsOmExYjJmZk4DQo=

## Environments Provider

When the Environments Provider receives this Authorization Token the following sequence of operations should be performed.

Operation	Legality Test
Identify the Authentication Method. (BASIC)	If it has an unexpected value, an irrecoverable incompatibility has been detected.
Base64 decode the Authorization Token to obtain the Application ID and secret code.  Lookup the Application ID in the list of pre-stored “authorized applications” to retrieve the pre-stored shared secret.	If there is no entry with an ID matching the supplied Application ID, authentication has failed, because the application is unknown.
Compare the pre-stored secret value with the shared secret decoded from the Authorization Token	If the values do not match then authentication has failed because the claimed known application did not provide the correct shared secret information.
The authentication of the issuing application has been confirmed.	The process to verify authorization for what the authenticated application is requesting now begins.

### 4.1.5 Authentication Validation: SIF\_HMACSHA256

In the create Environment request, assuming SIF\_HMACSHA256 authentication:

Element	Value	Meaning
authenticationMethod	SIF_HMACSHA256	Standard SIF 3.0 Authentication
applicationKey	RamseyPortal	Application ID only. Shared secret information is not sent.
authenticationToken	SIF_HMACSHA256 bmV3OjZUVmdZd2JBaG1RYzJ6QUxkYThad XBmcnpmcVorWEQ3ZjJiTUEwQXpXUm89Cg==)	** See details below

instanceId	District7	Qualifier to the Application Key. This allows the same application to register itself as a Consumer multiple times without conflict.
userToken	Martin.Smith:boywonder12	Verifiable user identification (in this case the unencoded concatenation of user identification and password) <sup>10</sup>

## Consumer

The *authorization* value is calculated in the following way:

Operation	Example Value
Concatenate the applicationKey and date / time and separate them by a ":"	RamseyPortal:2013-06-22T23:52-07
Calculate the HMAC SHA 256 value using the unsent Client Application shared secret, and then Base64 encode	6TVgYwbAhmQc2zALda8ZupfrzfzqZ+XD7f2bMA0AzWRo= =
Combine the applicationKey with this string and separate them by a ":"	RamseyPortal:6TVgYwbAhmQc2zALda8ZupfrzfzqZ+XD7f2bMA0AzWRo=)
Base64 encode the result and prefix with the authentication method and a space	SIF_HMACSHA256 bmV3OjZUVmdZd2JBaG1RYzJ6QUxkYThadXBmcnpmc VorWEQ3ZjJiTUEwQXpXUm89Cg==

## Environments Provider

When the Environments Provider receives this Authorization Token the following sequence of operations should be performed.

Operation	Associated Logic
Identify the Authentication Method. (SIF_HMACSHA256)	If it has an unexpected value, an irrecoverable incompatibility has been detected.
Base 64 decode to obtain the Application ID and the Base64 value of the HMAC SHA 256  Lookup the Application ID in the list of pre-stored "authorized applications" to retrieve the pre-stored shared secret.	If there is no entry with an ID matching the supplied Application ID, authentication has failed, because the application is unknown.

<sup>10</sup> Since the authentication of User Identity is the HMAC SHA 256 method, the shared information would not generally be present in the *userToken*.

Retrieve the Current Date and Time value that was used from the message header.	If the Date / Time value is not “reasonably current”, this may indicate the create Environment request was generated as part of a “replay” attack. In that case the request should be rejected.
Use the Application ID and date / time and calculate the HMAC SHA 256 value using the shared secret for this Application.  Compare the result with the string provided by the Consumer.	If the values do not match then authentication has failed because the claimed known application did not use the correct shared secret information in generating the HMAC SHA 256.
The authentication of the issuing application has been confirmed.	The process to verify authorization of the authenticated Environment “Consumer” for the service operations it is requesting now begins.  From this point on, the above process repeats, using the previous sessionToken as the basis for calculating the new authorization value

#### 4.1.6 Client Certificates: Best Practices

Many sites which utilize Client Certificates already have administrative procedures in place for managing application-specific certificates which are roughly equivalent to those needed to manage application-specific shared secrets. For those sites, adding SIF authentication should:

- Not require any additional administrative overhead (shared secret management IS certificate management)
- Leverage rather than duplicate Client Certificate security functionality
- Not introduce significant overhead to daily operations

The following administrative “best practices” are suggested for achieving these goals:

- Use Basic Authentication to minimize the unneeded additional security functionality provided by HMAC SHA256
- In cases where they are unique to the Certificate Authority, use the serial number of the Client Certificate as the basis for the application’s shared secret.

## 4.2 Session Creation

Once the application issuing the create Environment request has been authenticated, additional arguments in the body of the request must be examined to establish the unique identity of the new Consumer, and determine whether that Consumer should be granted a session and provided with an Environment.

The following sequence of operations assumes the Environments Provider supports an active “session table” in which the Consumer identity is maintained, and that no two active sessions can have the same Consumer identity.

Operation	Legality Testing
Compute the “Application Index” (unique identifier for requesting application instance).	If there was an instanceId specified in the create Environment request, combine it with the applicationKey to form the Application Index.

	Otherwise set the Application Index equal to the applicationKey.
<p>Verify any supplied userToken.</p> <p>User identity may take a variety of formats (LDAP UUID, OPENID, email address) within this key, each coupled with verification information (data, algorithm etc.). The method which the Consumer and Environments Provider used to “pre-agree” on the userToken format is outside the scope of this specification.</p> <p>In the simplest case, it could be formatted as an xs:token containing the User ID and Password (in clear text or Base64 encoding) concatenated by a “.”</p>	<p>If no userToken was specified, create the Session ID based upon the Application Index.</p> <p>Otherwise, if verification fails, then the create Environment request must be rejected, because the identity of the user for which the session is being requested cannot be authenticated. A typical example is where the user of a mobile application entered in an invalid password.</p> <p>Once the supplied User ID has been verified, combine it with the Application ID and create the Session ID.</p> <p>The actual value of this ID must encompass or index all information relative to the Consumer that makes that Consumer “unique”.</p>
<p>Construct the sessionToken by taking the Base64 encoded value of the Session ID.</p> <p>ex:UmFtc2V5UG9ydGFsOmExYjJmZk4LU1hcnRpci5TbWl0aDpib3l3b25kZXlzMg==</p>	Authentication of the requester is complete.
Use the new sessionToken, the create Environment payload data and any associated pre-authorization rights for the identified Consumer, to construct the new Environment.	Return the Environment to the successfully registered Consumer.

#### 4.2.1 Generating subsequent Authorization Tokens

The returned sessionToken will be used as the basis for calculating the *authorization* token which the Consumer must generate and include as part of each subsequent request. Effectively the returned sessionToken **replaces the “applicationKey” in the algorithm for the calculation / generation of the authorization token between a Consumer and its Environment.**

Whatever method (BASIC or SIF\_HMACSHA256) was used by the Consumer to construct the original *authorization* token and by the Environments Provider to verify it **should** still be used.

### 4.3 Overview of Basic Operations

The *environments* Service maintains the list of the individual Consumer Environments assigned to the various Consumers and Service Providers which interact together through the underlying Environments Provider implementation.

However as noted, only the *create* method is available to non-administrative Consumers. There is no non-administrative way to *query* the *environments* service for information about all existing Environment Entries.<sup>11</sup>

Once created, the only non-administrative Consumer which can access or delete an *environment* is the Consumer which successfully created it.

Operation	Description
<b>Create</b>	Create a Consumer Environment which will allow the Consumer (a combination of client application, application instance and user running the client application) to access the Infrastructure Services it has been assigned, and issue other Service operations it is authorized for.
<b>Update</b>	Not accessible in the environments Service Consumer API.
<b>Delete</b>	Not accessible in the environments Service Consumer API.
<b>Query</b>	Not accessible in the environments Service Consumer API.

#### 4.3.1 Create Environment

This request creates a unique Environment object for the Consumer, and is the only request that will be issued to a URL (Infrastructure Service or otherwise) which is not contained in or derived from content within that environment. The *create* operation payload provides all necessary information to allow the *environments* service to understand the application's protocol and security requirements, and basic expectations for interoperability.

If successful, the application is a Registered Consumer and has an environment containing:

- A collection of URLs corresponding to the set of Infrastructure Services that support its environment.
- Its pre-provisioning authorization rights, on a service request by request basis
- A unique *sessionToken* which it must use to construct the *authorization* token included in all subsequent Service Provider requests
- An assigned default Zone (typically pre-assigned by the Administrator) which the Consumer should use to scope all subsequent Service Provider requests in cases where the Zone is not explicitly specified.

<sup>11</sup> The actual implementation of the *environments* Service may support one or more of the *query*, *update* and *delete* entry operations for Administrative purposes, but they are not included as part of its required SIF 3.0 Provider interface.

Use Case Components	Details
<b>Actors</b>	External Application <i>consumerEnvironments</i> Service Zone Administrator
<b>Preconditions</b>	The Site Administrator has previously approved this application for admission into one or more Zones, and this fact is known to the <i>environments</i> Service, or the Site Security Policy allows any authenticated application to register (but not necessarily to be provisioned) A “ <i>create</i> ” request has been issued to the <i>environments</i> Service
<b>Actions</b>	Upon reception of the <i>create</i> request, the <i>environments</i> Service: <ul style="list-style-type: none"> <li>• Verifies that the indicated transport is supported by the Environments Provider and the infrastructure version number is supported. If not, either reject or (where the request was unintelligible) ignore it.</li> <li>• Authenticate the provided information (both Application and User ID) and ensure the Application Instance is unique. If that authorization fails, reject the <i>create</i> Environment Request.</li> <li>• The Consumer is now authenticated, although it is not yet provisioned. Compute and assign a <i>sessionToken</i> to the newly registered Consumer (which it will use as the basis of the <i>Authorization</i> field value sent in all subsequent requests).</li> <li>• Create the Consumer Environment, inserting the URLs of the Infrastructure Services which it is authorized to access, its Session Token, the provided Application Application Instance and User IDs, the Authentication Method (BASIC or HMAC-SHA256) and other supplied application data.</li> <li>• If the Environment supports a security policy that is based on pre-provisioning, fill in the service access rights of the new Consumer. This results in the Consumer being both Registered (authenticated) and Provisioned (authorized) for the set of Service operations it can initially Request and subscribe to.<sup>12</sup></li> <li>• Return the contents of the Consumer’s Environment Entry</li> </ul>
<b>Post Conditions</b>	The issuer is a Registered Consumer of the Environment. Depending on site security policy, anywhere from zero to a full complement of authorized operations may be pre-allocated.

<sup>12</sup> There is an important distinction for Subscription provisioning, in that a Consumer may be pre-provisioned to be able to receive Events from a specific type of Service Provider, but it will not actually take effect (Events will not arrive) until the Consumer explicitly subscribes to that Service by supplying an eventSubscription queue for receiving these Events.



The following elements are specified in the payload of the *create* Environment Request operation.

Element / @Attribute	Ch	Description	Type
<i>consumerName</i>	M	A descriptive name for the Consumer that will be readily identifiable to Zone Administrators if the <i>create</i> operation is successful.	xs:normalizedString
<i>solutionId</i>	O	<p>The “solution” the Application would like to participate in. This is optional only, is advisory, and may be ignored by the Administrator.</p> <p>If processed it may be reflected in the URLs of the infrastructure services which are provided in the <i>consumerEnvironment</i>.</p>	<p>xs:token.</p> <p>Typical values might be:</p> <ul style="list-style-type: none"> <li>• “production”</li> <li>• “staging”</li> <li>• “testing”</li> </ul>
<i>applicationInfo/applicationKey</i>	M	An opaque (to the SIF standard) element which contains any required application authentication information. The authorization level assigned to a specific value of this element is site-specific.	xs:token
<i>userToken</i>	O	<p>An opaque (to the SIF standard) element which contains any required (and verifiable) user authentication information.</p> <p>For example, in a Direct Environment which accepts Consumer Registration Requests from a mobile application, this element might contain a combination of the ID and Password of the owner of the mobile device.</p>	xs:any
<i>instanceId</i>	O	The particular instance of the application seeking to register as a Consumer. It might represent a grouping as diverse as a category of end users or the serial number of a mobile device.	xs:token
<i>authenticationMethod</i>	M	Defines the method by which the Consumer registration request will be authenticated by the service.	<p>Enumerated list. The legal values are:</p> <ul style="list-style-type: none"> <li>• <i>BASIC</i></li> <li>• <i>SIF_HMACSHA256</i></li> </ul>
<i>supportedInfrastructureVersion</i>	M	The version of the SIF infrastructure which the	versionType (see

		Consumer supports.	Appendix D)
transport	O	The transport which the Consumer expects the infrastructure to use to interoperate with it.	One of: <ul style="list-style-type: none"> <li>• “SOAP”<sup>13</sup></li> <li>• “REST”</li> </ul> <i>The default is whichever transport the create request was issued on</i>
applicationProduct	M	Details about the vendor, product and version of the application registering as a Service Consumer	productIdentityType (see Appendix D)
adapterProduct	O	If the application is utilizing a separate and distinct Adapter product, this element contains optional vendor details about that product which the Consumer might wish to make known to Zone Administrators.	productIdentityType (see Appendix D)

### Example:

The following XML instance fragment illustrates the formats of the basic elements contained in a *create* Request sent to the *environments* Service.

```

<solutionId>staging</solutionId>
< authenticationMethod>BASIC</authenticationMethod>
<instanceId>District7</instanceId>
<userToken>Martin.Smith:boywonder12</userToken>
<consumerName>DistrictPortal</consumerName>
<applicationInfo>
  <applicationKey>RamseyPortal</applicationKey>
  <supportedInfrastructureVersion>3.0</supportedInfrastructureVersion>
  <transport>REST</transport>
  <applicationProduct>
    <vendorName>Ramsey Software</vendorName>
    <productName>Students-R-Us</productName>
    <productVersion>1.5</productVersion>
  </applicationProduct>
</applicationInfo>

```

<sup>13</sup> Currently unsupported in SIF 3.0.

### 4.3.2 Create Environment Response

The response to a create Environment request is identical to that for any other Request.

#### 4.3.2.1 Success

A successful Response to the *create* Environment Request contains an HTTP Response Status of 201 (Created) and the following additional element in the returned Environment:

Element / @Attribute	Ch	Description	Type
<i>environment/sessionToken</i>	M	The value of this opaque element will be used by the Consumer in constructing the <i>authorization</i> token that <b>must</b> accompany every subsequent request made to the Environments Provider	xs:token

#### Example:

The following XML instance fragment illustrates the formats of the basic elements contained in the *create* Response. In this implementation the *applicationKey* is not embedded in the *environment* URL (implying security is obtained in other ways).

```
<environment id="RaleighRidgeDistrict-Test" type="DIRECT" >
  <sessionToken>
    UmFtc2V5UG9ydGFsOmExYjJmZk4LU1hcnRpbj5TbWI0aDpib3I3b25kZXIzMg==
  </sessionToken>

  ...
  ...
</environment>
```

## 5. Consumer *Environment*

The Consumer Environment Service whose URL is returned in response to a successful create Environment request is the gateway to all other services. It scopes the Consumer's possible interactions with the infrastructure and any Provider Services accessible from it. The Environment Service provides the set of (possibly customized) information the Application needs to interoperate successfully, including:

- Environment Name and level of API supported (Direct or Brokered)
- URLs to the full set of Infrastructure Services supporting this Environment
- The name of the application's default Zone in the Environment
- One or more elements related to the functionality provided to the Consumer within this Environment..
- The full set of authorized operations on all services with specified contexts in specified zones
- The set of active Events actually subscribed to and (Brokered Environments only) Object Service types provided by this Consumer.

### 5.1 Overview of Basic Operations

All contents of the Environment Service Entry are immutable. They cannot be changed by the Consumer, although the Consumer should delete the entry in its entirety as the final step in the *unregister* process.

Operation	Description
<b>Create</b>	Not available. The registering application utilizes the <i>Environments</i> Service to create a Consumer Environment as described above.
<b>Update</b>	Not accessible in the Service Consumer API. A Consumer cannot directly modify the contents of its Environment entry and must use the <i>provisionRequests</i> Service for that purpose.
<b>Delete</b>	Issued by the Consumer to unregister itself from its Environment
<b>Query</b>	Returns the Environment object for this Consumer.

As is true for all operations from this point, both the Environment *delete* and *query* operations require the *authorization* token as a parameter. The *environment* Service does not issue Events.

### 5.2 Query

The Consumer must acquire the information contained in its *environment* object to allow it to function. The following elements are returned in response to a *query* Request, divided into the following conceptual sections.

### 5.2.1 Global Elements

This collection of subelements contains data global to the remaining sections.

Element / @Attribute	Ch	Description	Type
environment		.	
@id	MUI	Identification of the Environment	UUID
@type	M	The type of Environment. This indicates whether certain functionality is optional or mandatory	One of: <ul style="list-style-type: none"> <li>• “MINIMAL”<sup>14</sup></li> <li>• “DIRECT”</li> <li>• “BROKERED”</li> <li>• “ADMINISTRATIVE”</li> </ul>
solutionId	O	The “solution” the Application is participating in.	xs:token. Typical values might be: <ul style="list-style-type: none"> <li>• “production”</li> <li>• “staging”</li> <li>• “testing”</li> </ul>
defaultZone	M	Identification of the default Zone, which is one of the defined Zones in the Consumer Environment’s Zone Registry (ex: “Suffolk Middle School”) and whose ID value will be used as the Zone element for all Consumer Requests and Subscription provisioning whenever the Consumer does not explicitly provide a Zone qualifier.	zoneType
authenticationMethod	M	Defines the way in which the <i>authorization</i> value can be used to enforce security.	xs:token
authorization	M	Derived from the applicationKey (and optionally from the combined applicationKey, instanceId and userToken), this opaque element reflects the current authorization rights of this Authenticated Consumer. It <b>must</b> be included in every message (Request,	xs:token

<sup>14</sup> Please refer to Appendices A, B and C of the Utility Services document for a more complete description of “minimal”.

		<p>Response, Event) the Consumer issues.</p> <p>It can be used (by the Infrastructure Services) to discover the location of the Consumer's Environment. It is never received by other Service Providers (which in fact receive their own <i>authorization</i> tokens with every incoming request so they know it is valid). This "switch" is performed by the Requests Connector.</p>	
userToken	O	<p>Contains any required user authentication information.</p> <p>For an Environment which accepts Consumer Registration Requests from a mobile application, this element might contain a combination of the User Name and encrypted Password.</p>	xs:any
consumerName	M	A descriptive name for the application	xs:normalizedString (max length 64)
<i>pagedBatchQuerySupport</i>	O	True if the Environment will support Paged Batch Query Requests. <sup>15</sup>	xs:Boolean. (If not present, "false" is assumed).

### Example:

The following XML instance fragment illustrates the formats of these global elements contained in a Direct Consumer Environment (with the other sections stubbed), all of which are returned to the Consumer after a successful Query

```

<environment id="RaleighRidgeDistrict-Test" type="DIRECT" >
  <solutionId>staging</solutionId>
  <defaultZone id="SuffolkMiddleSchool"></defaultZone>
  <authenticationMethod>BASIC</authenticationMethod>
  <userToken>johnsmith:ujm6rc50kyb</userToken>
  <consumerName>RafflesLMS</consumerName>
  <authorization>76e34b359d75141a8c3d00aa001ad45ef</authorization>
  <applicationInfo> ... </applicationInfo>
  <infrastructureServices> ... </infrastructureServices>
  <provisionedZones> .... </provisionedZones>
</environment>

```

<sup>15</sup> Please refer to the Base Architecture document for further details.

### 5.2.2 Application-specific Information

The application specific elements are shown below, and reflect in large measure the application information provided when the environment was originally created.

Element / @Attribute	Ch	Description	Type
<i>applicationInfo</i>			
applicationInfo/applicationKey	M	Uniquely identifies the Consumer application, and is the basis for initial authentication.	xs:token
applicationInfo/instanceId	O	This element qualifies the <i>applicationKey</i> so that the same authorized application software can be involved in multiple simultaneous sessions with the same Environments Provider.  It might contain the Device ID of where a particular application instance is executing, or it might indicate a virtual "grouping" of users accessing the set of object services reachable through the Environment (ex: Middle School Students).	xs:token
applicationInfo/dataModelNamespace	M	The URL that uniquely identifies the Namespace of the data model which the application supports	xs:anyURI
applicationInfo/supportedinfrastructureVersion	M	The version of the SIF infrastructure which the Consumer will support. This could be a wildcard, and may be different from the one used in the message header.	versionType (see Appendix D)
applicationInfo/transport	O	The transport which the Consumer supports.	One of: <ul style="list-style-type: none"> <li>• "SOAP"<sup>16</sup></li> <li>• "REST"</li> </ul>

<sup>16</sup> Currently unsupported in SIF 3.0.

applicationInfo/applicationProduct	M	Details about the vendor, product and version of the application registering as a Service Consumer	productIdentityType (see Appendix D)
applicationInfo/adapterProduct	O	If the application is utilizing a separate and distinct Adapter product, this element contains optional vendor details about that product which the Consumer might wish to make known to Zone Administrators.	productIdentityType (see Appendix D)

**Example:**

The following XML instance fragment is an example of the applicationInfo elements contained in a Direct *environment*, and returned to the Consumer after a successful *environment* Query.

```
<applicationInfo>
  <applicationKey>d3e34b359d75141a8c3d00aa001a1652</applicationKey>
  <supportedInfrastructureVersion>3.0</supportedInfrastructureVersion>
  < dataModelNamespace>http://www.sifassociation.org/datamodel/us/3.0</dataModelNamespace>
  <transport>REST</transport>
  <applicationProduct>
    <vendorName>RamseySoftware</vendorName>
    <productName>Students-R-Us</productName>
    <productVersion>1.5</productVersion>
  </applicationProduct>
</applicationInfo>
```

**5.2.3 Infrastructure Service Assignment**

This section contains the URLs of the common set of Infrastructure Services that are available to the Consumer.

Note that the *environments* service is not a member of this group, which means that a given, non-administrative Consumer cannot discover any other Consumers or see their Environments or learn about their provisioning. This limitation is imposed for security reasons.

Cross-consumer interactions do occur when one Consumer requests a change (*create*, *update* or *delete*) which is accepted by a Service Provider. The resulting published Event is visible to and received by all those Consumers who are subscribed to receive Events from that Service Provider.

Element / @Attribute	Ch	Description	Type
<i>infrastructureServices</i>	M	The collection of Infrastructure Services which together define the Consumer's Environment.	



<i>infrastructureServices</i> <i>/infrastructureService</i>	RM	<p>There <b>must</b> be an infrastructureService element present for <u>each</u> defined Infrastructure Service.</p> <p>Each infrastructureService <i>name</i> attribute <b>must</b> equal one of the names defined in the infrastructureServiceNameType enumerated list (see Appendix D).</p> <p>The value of each infrastructureService Property <i>value</i> subelement defines the URL location of that Infrastructure Service.</p>	<i>propertyType</i> (see Appendix D).
--	----	---	---------------------------------------

**Example:** All Infrastructure Service URLs were assigned relative to the URL of the *environments* which implies they are assigned to each individual *environment* created. This represents a common convention that may not be applicable in all situations (as for example when a new Infrastructure Service instance is being tested or phased in, a Consumer at a time). In this case the Consumer is cleared to be a Provider, and has been given access to the *eventsConnector* as well as the *requestsConnector*.

<infrastructureServices>

<!--The presence of the following two Services are mandatory for any Consumer Environment →

<infrastructureService name="environment">

<https://www.raleighridge.nc.edu/sif3.0/staging/environments/1234567>

</infrastructureService>

<infrastructureService name="requestsConnector">

<https://www.raleighridge.nc.edu/sif3.0/staging/requestsconnector>

</infrastructureService>

<!--These two Services support the reception of delayed Responses and asynchronous Events. They are mandatory for all Brokered Environments →

<infrastructureService name="queues">

<https://www.raleighridge.nc.edu/sif3.0/staging/queues>

</infrastructureService>

<infrastructureService name="subscriptions">

<https://www.raleighridge.nc.edu/sif3.0/staging/subscriptions>

</infrastructureService>

<!--This Service is necessary to support self-configuring Consumers →

<infrastructureService name="provisionRequests">

<https://www.raleighridge.nc.edu/sif3.0/staging/provisionRequests>

</infrastructureService>

<!--After provisioning itself as a Provider, this service allows a Consumer to post Events. It is mandatory for all Brokered Environments and never included in a Direct Environment. →

<infrastructureService name="eventsConnector">

<https://www.raleighridge.nc.edu/sif3.0/staging/eventsconnector>

</infrastructureService>

</infrastructureServices>

#### 5.2.4 Service Provisioning

This section describes the extent of the current state of provisioning of the Consumer to all non-Infrastructure (Utility, Data Object and Functional) Service Providers. These Service access rights are grouped by Zone, and so the top level element is *provisionedZones*, which collects one or more *provisionedZone* elements with the structure shown below.

The possible setting for each Consumer access right is one of the following:

- **UNSUPPORTED:** The Provider does not support this operation, at least for this type of Consumer. The corresponding Request cannot be made.
- **SUPPORTED:** The Provider supports this operation, but the authorization to issue it has not been requested by this Consumer (and whether such authorization would be granted is unknown). A Consumer request of this operation from the Provider might succeed or fail. “Supported” is the default value for any Service operation not contained in this section of the Consumer’s *environment*.
- **REJECTED:** The Provider supports this operation, but the Consumer does not have and will not be granted authorization to perform this operation. This status may be the result or pre-provisioning or set in response to a specific Consumer provisioning request.
- **APPROVED:** The Consumer has the authorization to perform this operation on this Service in this Zone.

The type of each possible Consumer access right is one of the following:

- **QUERY:** The ability to Query this Service directly and subscribe to its Events.
- **CREATE:** The ability to Create new Service Objects
- **DELETE:** The ability to delete Service Objects
- **UPDATE:** The ability to update Service Objects
- **SUBSCRIBE:** The ability to subscribe to Object Events from this Service
- **PROVIDE:** The ability to provide this Service
- **ADMIN:** The ability to access this Service in non-standard ways.<sup>17</sup>

---

<sup>17</sup> This right is generally reserved for administrative applications and is meant as a way to provide administrator control over a deployed SIF 3.0 solution. The specifics of what this means for a given Service is outside the scope of the specification.

If an Environment supports self-provisioning Consumers (i.e. Consumers who first determine if they are authorized to use a resource before actually attempting to use it) then either this section must initially contain all the rights the Consumer has to all Services (both Data Object and Utility) in all Zones, or the *provisionRequests* Infrastructure Service must be present in the Consumer's Environment to allow additional rights to be obtained.

The actual elements comprising this section are shown in the table below:

| Element / @Attribute                         | Ch | Description  | Type   |
|--|----|--|--|
| <i>provisionedZone</i>                       | MR | Contains the list of Services and associated provisioning available to the Consumer in the Zone  |  |
| <i>provisionedZone@id</i>                    | M  | The unique id of the Zone, and the key to the Zone element in the Zone Registry. This is the value that is sent as the "owning" Zone of any Service contained within this element, when a Request for that Service is issued to the <i>requestsConnector</i> .   | xs:token<br><br>All Utility Services are contained in the Zone with the ID of " <i>environment-global</i> "  |
| <i>provisionedZone/services</i>              |    | Collection of Services provisioned in this Zone  |  |
| <i>provisionedZone/services/service</i>      | OR | The record of the specific Service provisioning of this Consumer in this Zone  |  |
| <i>provisionedZone/services/service@type</i> | M  | The type of the Service  | One of: <ul style="list-style-type: none"> <li>• UTILITY</li> <li>• OBJECT</li> <li>• FUNCTION</li> <li>• SERVICEPATH</li> <li>• XQUERYTEMPLATE</li> </ul> |
| <i>provisionedZone/services/service@name</i> | M  | The name of the Service as it will appear in a request to that Provider. . For utilities, this is fixed to one of the defined set of Utility Service Names. For objects and functions, it is defined by the Data Model.<br><br>The value of this element may optionally be a Service Path <sup>18</sup> of the form: | xs:token   |

<sup>18</sup> Refer to the Service Path section in the Base Architecture document for further details.

|   |    |   |   |
|---|----|---|---|
|   |    | <p><i>students/{}/sections</i></p> <p>as well as a Service Name (ex: <i>sections</i>). Although Queries to either will return Section objects, in the case of the Service Path, the only “right” which may be granted to the Consumer is “QUERY”.</p>                                 |   |
| <i>provisionedZone/services/service@contextId</i>         | O  | <p>The unique identity of a <i>context</i> element, which is associated with a Provider of this name and type operating in a Zone with this ID.</p> <p>All Services with the same name in the same Zone must have different Context IDs. Only one such Service can have “DEFAULT”</p> | <p>xs:token</p> <p>If not specified, a platform mapping may substitute the value “DEFAULT” here.</p>  |
| <i>provisionedZone/services/service/rights</i>            | M  | The collection of access rights granted to the Consumer for this Service as a result of pre-provisioning or the creation of provisionRequests containing assertions about the additional rights needed.   |   |
| <i>provisionedZone/services/service/rights/right</i>      | MR | A specific Access Right to the owning service   | <p>One of:</p> <ul style="list-style-type: none"> <li>• “UNSUPPORTED”</li> <li>• “SUPPORTED”</li> <li>• “REJECTED”</li> <li>• “APPROVED”</li> </ul>   |
| <i>provisionedZone/services/service/rights/right@type</i> | M  | The type of the Access Right  | <p>One of:</p> <ul style="list-style-type: none"> <li>• “QUERY”</li> <li>• “CREATE”</li> <li>• “DELETE”</li> <li>• “UPDATE”</li> <li>• “SUBSCRIBE”</li> <li>• “PROVIDE”</li> <li>• “ADMIN”</li> </ul> |

**Example:** A Consumer is currently authorized to access both the Alert Log Utility Service and a Student Object Provider in the Suffolk Middle School Zone. The current provisioning indicates:

- **Alerts:** The Consumer can only create Alerts (and in fact Query, Update and Delete Alerts are not available to non-admin Consumers in this Environment). The Consumer has been specifically prevented from subscribing to Alert Events, even before creating a *provisionRequest* to do so.
- **Students:** The Consumer can Query a Student, but whether it can also Create or Update a Student has not yet been determined, and the Consumer has not asked. The Consumer is explicitly forbidden from deleting a Student. It has been granted permission to Subscribe to Student Events and in fact may already have done so.

```

<provisionedZones>
  <provisionedZone id="environment-global">19
    <services>
      <service type="utility" name="alerts" contextId="DEFAULT">
        <rights>
          <right type="QUERY">UNSUPPORTED</right>
          <right type="CREATE">APPROVED</right>
          <right type="UPDATE">UNSUPPORTED</right>
          <right type="DELETE">UNSUPPORTED</right>
          <right type="SUBSCRIBE">REJECTED</right>
        </rights>
      </service>
    </services>
  </provisionedZone>
  <provisionedZone id="SuffolkMiddleSchool">
    <services>
      <service type="object" name="student" contextId="DEFAULT">
        <rights>
          <right type="QUERY">APPROVED</right>
          <right type="CREATE">SUPPORTED</right>
          <right type="UPDATE">SUPPORTED</right>
          <right type="DELETE">REJECTED</right>
          <right type="SUBSCRIBE">APPROVED</right>
          <right type="PROVIDE">REJECTED</right>
        </rights>
      </service>
    </services>
  </provisionedZone>
</provisionedZones>

```

### 5.3 Delete

The Delete Environment Request is the last operation a Consumer issues when removing itself from contact with the Environments Provider Service. If the Consumer is also the Provider of one or more Object or Function Services, the well-behaved (but not mandatory) exit sequence involves, for each such service provided:

---

<sup>19</sup> If present, a Utility Service can always be accessed in the "environment-global" Zone.

- Delete the corresponding *provider* entry which had been previously allocated when the Consumer declared itself a Provider of this Service. This will remove this Service from the Provider Registry and stop any further Consumer Requests to this Service from entering the corresponding Queue previously set up to receive them.
- Ensure all Requests currently in the Request Queue for this Provider are completely serviced.

When the deletion is completed, all Consumer resources are freed and the Consumer's *authorization* token is invalid.

## 6. Provision Requests

The Consumer's Environment Service only supports Query operations. The *provisionRequests* service allows a Consumer to create a *provisionRequest* to modify the contents of its *environment* in a controlled manner.

The following types of changes can be made as a result of a successful *provisionRequest*, and each change will be reflected in the *environment* the next time it is queried.

- Extending the current set of authorized per-Zone / per-Context / per-Service rights.

### 6.1 Data Elements

In many ways, the payload of a *provisionRequest* create operation resembles what the payload of a limited update operation to an *environment* might look like. It contains a subset of *environment* elements required to acquire or enhance new functionality or resources for the issuing Consumer.

These elements are confined to the *queues* and *provisioned Zones* elements in the *environment*.

| Element / @Attribute                         | Ch | Description   | Type  |
|--|----|---|---|
| <i>provisionRequest</i>                      |    |   |   |
| <i>provisionRequest@completion</i><br>Status | O  | <p>The completion status of the assertions and requests for additional resources made in this <i>provisionRequest</i>. This attribute is omitted on creation, and is only present when the <i>provisionRequest</i> is queried successfully.</p> <p>A "MIXED" status indicates one or more assertions failed to be granted. A REJECTED indicates all assertions were rejected.</p> <p>In the case of a single assertion (ex: Subscribe to Service X) the final status will be either ACCEPTED or REJECTED.</p> | One of: <ul style="list-style-type: none"> <li>• "ACCEPTED"</li> <li>• "MIXED"</li> <li>• "REJECTED"</li> </ul> |
| <b><i>provisionedZones</i></b>               | O  | <p>Contains the list of per Service provisioning requests.</p> <p>Whether successful or not, the results of any additional Service rights assertions (ACCEPTED or REJECTED) will be merged into the existing set currently found in the <i>environment</i>.</p>   |   |
| <i>provisionedZones/provisioned</i>          | OR | A specific Zone defined for this <i>environment</i> .   |   |

|   |    |   |  |
|---|----|---|--|
| <i>Zone</i>                                       |    |   |  |
| <i>provisionedZones/provisionedZone@id</i>        | M  | The unique id of the Zone, and the key to the Zone element in the Zone Registry. .  | xs:token<br><br>All Utility Services are contained in the Zone with the ID of “environment-global”   |
| <i>provisionedZone/services</i>                   |    | Collection of Services within this Zone for which additional provisioning is desired  |  |
| <i>provisionedZone/services/service</i>           | OR | The record of a specific Service for which provisioning is being extended   |  |
| <i>provisionedZone/services/service@type</i>      | M  | The type of the Service   | One of: <ul style="list-style-type: none"> <li>• UTILITY</li> <li>• OBJECT</li> <li>• FUNCTION</li> <li>• SERVICEPATH</li> <li>• XQUERYTEMPLATE</li> </ul> |
| <i>provisionedZone/services/service@name</i>      | M  | <p>The name of the Service as it will appear in a request to that Provider. For utilities, this is fixed to one of the defined set of Utility Service Names. For objects and functions, it is defined by the Data Model.</p> <p>The value of this element may optionally be a Service Path of the form:</p> <p style="text-align: center;"><i>students/{}/sections</i></p> <p>as well as a Service Name (ex: <i>sections</i>). Although Queries to either will return Section objects, in the case of the Service Path, the only “right” which may be granted to the Consumer is “QUERY”.</p> | xs:token   |
| <i>provisionedZone/services/service@contextId</i> | O  | The unique identity of a <i>context</i> element, which is associated with a Provider of this name and type operating in a Zone with this ID.  | xs:token<br><br>If not specified, a platform mapping may substitute the value “ <i>DEFAULT</i> ” here.   |



|   |    |   |   |
|---|----|---|---|
|   |    | All Services with the same name in the same Zone must have different Context IDs. Only one such Service can have "no Context"   |   |
| <i>provisionedZone/services/service/rights</i>            | M  | The collection of additional access rights being asserted   |   |
| <i>provisionedZone/services/service/rights/right</i>      | MR | A specific Right assertion to issue the specified Request type to the owning service.<br><br>When the authorizations contained in this provisionRequest are resolved, the equivalent value of this element will be either "ACCEPTED" or "REJECTED". | The only correct value is REQUEST   |
| <i>provisionedZone/services/service/rights/right@type</i> | M  | The type of the requested Access Right. Note that if Query is authorized, the Consumer may also subscribe to Events.  | One of: <ul style="list-style-type: none"> <li>• "QUERY"</li> <li>• "CREATE"</li> <li>• "DELETE"</li> <li>• "UPDATE"</li> <li>• "SUBSCRIBE"</li> <li>• "PROVIDE"</li> </ul> |

## 6.2 Overview of Basic Operations

The basic operations are split between the following two services.

### 6.2.1 *provisionRequests*

The provisionRequests service supports the following operations

| Operation     | Description   |
|---------------|---|
| <b>Create</b> | This creates a <i>provisionRequest</i> whose URL is returned to the issuing Consumer. |

**Example:** The Consumer creates a *provisionRequest* to assert create and update privileges on the identified Student object.

```
<provisionRequest>
  <provisionedZone id="SuffolkMiddleSchool">
    <services>
```

```

    <service type="object" name="student" contextId="DEFAULT">
      <rights>
        <right type="CREATE">REQUESTED</right>
        <right type="UPDATE">REQUESTED</right>
      </rights>
    </service>
  </services>
</provisionedZone>
</provisionRequest>

```

### 6.2.2 provisionRequest

The provisionRequest service supports the following operations.

| Operation     | Description  |
|---------------|--|
| <b>Query</b>  | Provides the status of each assertion or resource allocation in a completed provisionRequest after it has been serviced. The changes are also reflected in the Consumer's <i>environment</i> . |
| <b>Delete</b> | The provisionRequest object is normally deleted by the Consumer after it has been queried successfully   |

**Example:** The above *provisionRequest* payload as returned by the following Query. The ability to update a Student was granted and the ability to create a student was rejected.

Note that the completionStatus is now present (MIXED), both new Queue Requests have a filled in *queueUri* so they can be accessed, and the ACCEPTED and REJECTED access right status now accompanies the original assertion request.

```

<provisionRequest completionStatus="MIXED">
  <provisionedZone id="SuffolkMiddleSchool">
    <services>
      <service type="object" name="student" contextId="DEFAULT">
        <rights>
          <right type="CREATE">REJECTED</right>
          <right type="UPDATE">ACCEPTED</right>
        </rights>
      </service>
    </services>
  </provisionedZone>

```

## 6.3 Lazy Authorization

A *lazy authorization* policy is one where the Site Administrator pre-determines only what authenticated applications will be registered as Consumers and where they will be deployed (ex: into a Test or a Production Environment). The details

of what Provider Service operations each application will be authorized to utilize are determined only at the time the Consumer first attempts to create its *environment*.

In such cases, the initial *environment* created for most new Consumers will contain a partly or totally empty *provisionedZones* element. In order to provision themselves adequately, Consumers will then need to issue one or more successful *provisionRequest* create operations. Each one of these requests will then have to be reviewed by administrative personnel and manually approved or rejected. This introduces a potentially very long lag between the time the *provisionRequest* is created, and the time it is executed, and the results can be viewed.

To handle this use case, extending a Consumer's authorization rights has been made into a three stage process. When the *create* request is issued to the *provisionRequests* Service, a new *provisionrequest* is created, and its URL is returned.

The Consumer then must query the *provisionRequest*. Since possibly long processing delays can occur at this point, an acceptable Response to this query would be an HTTP Return Code of 202 indicating that the query has been "accepted", but cannot as yet be completed. This requires the Consumer to reissue this identical query at some later time, and to keep repeating this action (perhaps at 5 minute intervals) until an HTTP Return Code of 200 is received, which will be accompanied by a data payload containing the completed *provisionRequest* with all assertions resolved.

The Consumer can examine the returned *provisionRequest* and adjust its expectations accordingly. All indicated results will also be reflected in the *environment* (the new default Zone ID and / or the acceptance or rejection of each authorization rights assertion). The sequence concludes when the Consumer deletes the *provisionRequest*.

The two possible Site Authorization strategies are illustrated in the table below.

| Site Strategy   | Meaning   | Consumer Actions   |
|---|---|--|
| <b>Pre-allocation of Consumer Authentication and Authorization</b>                | <p>The Consumer Environment was immediately created upon registration, and is pre-populated with all the rights to the Provider Services which the Consumer can use.</p> <p>The status of all Rights Assertions is <i>APPROVED</i>.</p>   | <p>The Consumer can either Query its <i>environment</i> to determine the Zones and Contexts of those Service Providers which it can access, or it can be prewired with this knowledge.</p> <p>In either case it can begin making Requests of the Providers it was pre-authorized to access.</p>  |
| <b>Pre-allocation of Consumer Authentication only (<i>lazy authorization</i>)</b> | <p>The Consumer Environment was immediately created upon registration, but the <i>provisionedZones</i> section is either empty or only partially filled with granted Service Provider rights.</p> <p>The Administrator will be informed when the Consumer makes any additional authorization assertions</p> | <p>The Consumer needs to request some or all of the authorization rights it needs. This information can either be prewired into the Consumer, or it can search the Provider Registry for those Provider Services it requires (possibly using its default Zone ID as a qualifier).</p> <p>In either case, once it obtains the set of Providers and determines the "rights" it needs to assert, it creates a <i>provisionRequest</i>, and then issues a <i>Query</i> and keeps doing</p> |

|  |  |  |
|--|--|--|
|  | <p>via creating a <i>provisionrequest</i>, and a manual go / no go decision will be made at that time.</p> | <p>so until the <i>provisionRequest</i> is returned.</p> <p>If the <i>provisionRequest</i> status was successful, all the authorization assertions of the Consumer are approved and the Consumer should delete the <i>provisionRequest</i> and begin issuing Provider Requests through the <i>requestsConnector</i>.</p> <p>If the status was failure or partial failure, the Consumer should examine the <i>provisionRequest</i> to see the current status of its authorization, and adjust its behavior accordingly (or exit after creating an Alert).</p> |
|--|--|--|

## 7. Requests Connector

A Consumer accesses every Infrastructure Service through its individual URL, provided within the Consumer environment when the Consumer originally registered. For access to all other services (Utility, Object and Functional) the Consumer uses the URL of the Requests Connector Infrastructure Service.

This URL provides the “pipe” into which all Consumer Requests are entered, and the *requestsConnector* owns securely and efficiently routing every entered request (and its response) to the assigned Queue of its correct destination.

### 7.1 Overview of Basic Operations

When a Service Consumer wants to invoke a method on any non-Infrastructure Service Provider (including a Utility Service), it does so by invoking the parallel method on the URL supplied by the *requestsConnector* Service (allocated within its *environment* at the time when it first registered as a Consumer). For example, to invoke the *create* operation on a specified Object Service, the Consumer invokes the identical *create* operation on the Requests Connector Service.

Specifically the supported operations are:

| Operation     | Description  |
|---------------|--|
| <b>Create</b> | On success, will deliver a <i>create</i> Request to the URL of the designated Service Provider   |
| <b>Update</b> | On success, will deliver an <i>update</i> Request to the URL of the designated Service Provider  |
| <b>Delete</b> | On success, will deliver a <i>delete</i> Request to the URL of the designated Service Provider   |
| <b>Query</b>  | On success, will deliver either a <i>query</i> (possibly paged and / or containing an XQuery Template ID) Request to the URL of the designated Service Provider. |

In a Direct Environment (no middleware), the above requests are all directly processed by the single implementation which supports all Infrastructure Services including the Requests Connector, as well as all available Utility, Object and Functional Services. Any Request routing occurring there is between the internal components of that implementation, and is therefore outside the scope of this specification.

The following discussion therefore applies to the flow of Requests issued to a Connector Service in a Brokered Environment only.

### 7.2 Identification of Service Provider

In order to correctly route a Consumer Request, the ultimate destination of the Service Provider must be determined. Scope the destination by identifying the Zone, Context and Service of the desired Provider with the Request, allowing the

Requests Connector to supply “content based routing” functionality to determine the ultimate recipient<sup>20</sup>. If there is no registered Provider which meets those scoping requirements, or if the Consumer does not have appropriate authorization rights to issue that request to that Service Provider, the Request will be rejected.

In the course of moving a Request from the Service Consumer to the Service Provider and the Provider's Response back to the Consumer, the Requests Connector may internally route to modules which compress, encrypt, authenticate and possibly strip the contents (XML filtering), but all of this **must** be transparent to both the Consumer and Provider.

### 7.3 Secure Request / Response Routing

Any Consumer Request can indicate that its Response is to be either “Immediate” or “Delayed”. In the Immediate case, the Request is issued as an HTTP Request, and the Requests Connector will route it to the appropriate Provider, and then route the Provider Response back to the open HTTP connection and return it synchronously in the HTTP Response.

If the Response is to be “Delayed”, the issuing Consumer provides the identity of one of its assigned Consumer Queues to which the Response from the Service Provider **must** be routed. The actual delayed Provider Response arrives back at the issuing Consumer's specified Queue, in the form of an asynchronous incoming HTTP Request.

The Requests Connector encapsulates several levels of security functionality in the course of this Request / Response routing. These will be described below by following the complete exchange sequence and examining the security provided / imposed at each step.

#### 7.3.1 Consumer to Requests Connector

Every Request is sent out as an HTTP Request to the Requests Connector and includes the Consumer's *authorization*. For delayed Requests, a source ID is also included, corresponding to a previously allocated Consumer Queue which is assigned to receive the asynchronous response(s) to this request.

A Consumer must have previously been granted the appropriate authorization to issue a Request to the destination Service type in the specified Zone and Context, and the Provider for that Service must be available.

#### 7.3.2 Internal to Requests Connector

The Requests Connector determines the ultimate destination of the request (see previous section). If the corresponding authorization right was not granted, the Requests Connector will discover this from the supplied *authorization*. In that case it will reject the Request.

Even if authorization to invoke the request was granted, a Consumer may still be restricted as to the elements it can query and retrieve from objects of a given type. The Requests Connector may interpose XML filtering modules on outbound *query* Requests.

---

<sup>20</sup> If the Zone is not specified in a Request, the default Zone value for the issuing Consumer is used. If the Context is not specified in a Request the “DEFAULT” Context value is used.

The Requests Connector then associates the Request with the ultimate Response destination. For immediate (synchronous) responses, this is the HTTP connection opened by the Consumer, on which the Request arrived. That HTTP Request remains “open”.

For delayed (asynchronous) responses, this is the Consumer Queue specified as the source ID for the original Request. Here the Requests Connector issues an HTTP Response with a status of 202 (*Accepted*), “closing” the HTTP Request. The Provider Response will arrive asynchronously at a later time.

### **7.3.3 Requests Connector to Provider Queue**

At this point, the Requests Connector can deliver the Request. It replaces the Consumer’s *authorization* token with the one the Service Provider was granted when it registered as a Consumer, before placing it in the Provider’s Queue. This serves two purposes:

- It proves the validity of the incoming request to the Service Provider, which can now verify that its own *authorization* token is included.
- It avoids compromising the security of the Consumer, since the Consumer’s *authenticationToken* remains known only to the Requests Connector.

A Service Provider has assigned a URL to receive its incoming Consumer Requests. Therefore once the Requests Connector determines the correct Service Provider, it delivers the Consumer Request (in the order received) at that URL as an incoming HTTP Request of exactly the same type, and with exactly the same payload as it had when it was issued.

### **7.3.4 Provider Response**

The Provider services the request and composes a Response. It issues it as an immediate HTTP Response to the earlier HTTP Request. While unnecessary for security reasons, the Provider includes its own *authorization* token in this response.

### **7.3.5 Requests Connector to Consumer**

When the Response from the Provider is received, XML filtering modules may be imposed before passing it along to the ultimate destination, completing the Request/Response exchange sequence.

If this is an immediate response, the Provider’s Response is sent back as the HTTP Response to the Consumer, completing the HTTP connection.

If this is a delayed response, there is no open HTTP connection. The Provider’s Response is sent as an HTTP Request to the Queue specified in the original Consumer Request.

## 7.4 Data Structure

There is no specific Infrastructure Data Object associated with the Request Connector. However the following global elements (external to the contents of the actual data object involved) are among those contained “outside the body” of any request (*query*, *create*, *update* or *delete*) sent to the Request Connector.<sup>21</sup>

- `messageld`
- `timestamp`
- `zoneld`
- `contextId`
- `objectName` (data model specific if not a Utility Service)
- `authorization`

The `requestsConnector` inserts an additional global element only when the request is being sent to a Utility Service:

- `environmentId` (the id attribute of the Consumer's *environment*).

This may be used to determine which objects a given Consumer has created, allowing the Utility Service to optionally offer a Consumer the ability to query, update and delete only those objects which it has created.

There may be additional elements specific to the request type (ex: the starting page and number of pages defined in a Query request) that also accompany the Request and / or the Response. .

For the REST transport, the above elements and others are conveyed in non-XML format, in either the HTTP Header of the Request, or in matrix parameters or query string extensions to the base URL of the Request Connector Service, and are delivered the same way to the Provider's incoming Request Queue. This allows the developer of the Provider to unmarshal any object data directly from the HTTP Body of a POST or PUT without the need for writing special code to deal with a wrapper of type `xs:any`.

---

<sup>21</sup> Please see the Base Architecture document for a detailed definition of these global elements.



## 8. Events Connector

As described in the previous section, a Consumer issues one of four operations to the Requests Connector: *query*, *create*, *update* or *delete*. In a Brokered Environment, these operations are routed, and reissued to the specified Service Provider.

A Consumer which is also a Service Provider must be able to issue two additional message types above and beyond a Request.

- **Response:** A Service Provider receives all Consumer Requests on its supplied URL. This means they arrive as asynchronous HTTP Requests. The Provider processes the Request and sends its Response back as a synchronous HTTP RESPONSE, completing the connection. There is no need for a Connector Service to accept and route Provider Responses.
- **Event:** As a result of either processing a Consumer Request, or because of some internal data change, the Provider will publish a corresponding change Event of type *create*, *update* or *delete*. These are replicated and delivered to each Subscriber as an Event.

The Service Provider relies upon the Event Connector to securely deliver these published Events to their correct destination Queues in much the same way as the Service Consumer relies on the Requests Connector.

### 8.1 Overview of Basic Operations

When a Service Provider publishes an Event, it does so by posting the Event to the *eventsConnector* URL within its *environment* (allocated at the time when it successfully provisioned itself as a Provider)<sup>22</sup>. For example, to invoke the *create* operation on a specified Object Service, the Consumer invokes the identical *create* operation on the Requests Connector Service.

Specifically the supported operations are:

| Operation     | Description  |
|---------------|--|
| <b>Create</b> | On success, will deliver the published <i>event</i> to the assigned Queue of every valid Subscribing Consumer. |

### 8.2 Security, Replication and Delivery to Subscribing Consumers

A Service Provider publishes an Event by issuing a “*create*” to the *eventsConnector* in the form of an HTTP Request. This includes the Provider’s *authorization* token, and the Zone and Context identification as well as the Service name of the publisher.

---

<sup>22</sup> Whether the value of the *eventsConnector* URL is unique to the Service or identical in all cases is an implementation detail.

- The *eventsConnector* uses the token to verify that the Provider is authorized to issue this Event. If the publisher is not the authorized Provider, the *create* fails, an error is returned and an *alert* is created to report the problem.
- Otherwise the *eventsConnector* returns an HTTP status of 202 (“Accepted”) to the Publisher in the form of an HTTP Response, completing the exchange. There are no further interactions with the Event Publisher from this point on.
- The remaining elements are then used to “scope” the Event type and locate any / all previously subscribed Consumers.<sup>23</sup>. If none are found, the processing is completed.
- Otherwise, the Event is replicated and delivered to the Queue of every subscribing Consumer specified.

In the course of moving an Event from the publisher to an individual subscriber’s queue, the Events Connector may internally route to modules which compress, encrypt, authenticate and possibly strip the contents (XML filtering), but all of this **must** be transparent to both the Subscriber and Publisher.

### 8.3 Data Structure

The following global elements are contained “outside the payload” of any Event (*create*, *update* or *delete*) posted to the Environment Connector<sup>24</sup>.

- *messageId*
- *timestamp*
- *zoneId*
- *contextId*
- *serviceType*
- *serviceName* (data model specific if not a Utility Service)
- *authorization*
- *eventType*

The above values are conveyed in non-XML format, in the HTTP Header of the Event. They are delivered the same way to the subscribing Consumer’s Queue, and retrieved that way in response to a poll. This allows the developer of the Consumer to unmarshal the object data directly from the Body of the HTTP Response.

---

<sup>23</sup> Directed Events (where the Provider specifies a single specific recipient) are not supported in this release.

<sup>24</sup> Please refer to the Base Architecture document for a more complete description of these elements.

## 9. Queues, Queue Instance and Queue Messages

There are three services involved with supporting the queuing and ordered retrieval of arriving asynchronous messages.

- A *Queues* Service which supports creation, deletion and querying of Queue (Instance) / Queue Messages Service pairs
- A *Queue* Service which is a service instance “dedicated” to a single Consumer owner, and responsible for collecting arriving asynchronous messages (delayed Responses and Events) and guaranteeing their first in / first out” (FIFO) ordered delivery. It also collects and maintains message traffic and Consumer usage statistics which can be queried.
- A *Queue Messages* Service which provides a way to retrieve the “oldest” non-accepted asynchronous message from the Queue without requiring the Consumer to support incoming HTTP connections (i.e. a “Polling Queue”<sup>25</sup>).

The mechanism by which incoming Request, Event or delayed Response messages are placed into a Queue Instance is an implementation detail of the Environment infrastructure. This section will define only the Consumer-facing interfaces of the three Queue-related services.

### 9.1 Service Operations

The following Consumer Requests (Service Operations) are supported by the set of **Queue** Services described above.

| Queues Registry Operation | Service Description ( <a href="https://www/.../queues">https://www/.../queues</a> )   |
|---------------------------|---|
| Query                     | Returns the contents of one or more Queue Instance objects belonging to the Consumer issuing the Query, or optionally, all Queue Instance objects if the Consumer has Administrative level authorization                          |
| Create                    | Create a Queue Instance<br><br>Only one Queue can be created (singular form only). This action will return a Queue object, with a uniquely assigned ID.   |
| Delete                    | Optionally supported only for Consumers with Administrative level authorization.<br><br>Only the Queue Instance owner can delete the Queue, and this is achieved by issuing a <i>delete</i> request directly to the Queue object. |

<sup>25</sup> A Polling Queue is the (considerably extended) equivalent of the SIF 2.x “Pull Mode” Queue.

|                                 |  |
|---------------------------------|--|
| Update                          | Unsupported.   |
| <b>Queue Instance Operation</b> | <b>Service Description (<a href="https://www/.../queues/{queueid}">https://www/.../queues/{queueid}</a> )</b>  |
| Query                           | Returns the contents of the Queue object   |
| Create                          | Unsupported. A Queue is created by issuing a <i>create</i> Request to the Queues Registry.   |
| Delete                          | <p>Destroys any associated Subscription Objects, throws away all queued messages and deletes the Queue Instance.</p> <p>This action sequence also automatically happens for all Queues owned by a Consumer when it deregisters by deleting its Environment object.</p> |
| Update                          | Unsupported. Once a Queue instance is created, all Queue parameters are immutable.   |
| <b>Messages Operation</b>       | <b>Service Description (<a href="https://www/.../queues/{queueid}/messages">https://www/.../queues/{queueid}/messages</a> )</b>  |
| Query                           | <p>Synchronously retrieves the oldest undelivered asynchronous message (delayed Response or Event) in a synchronous manner from the (FIFO) Queue Instance.</p> <p>Optional arguments and supported functionality will be examined in the subsections below.</p>        |
| Create                          | <p>Optionally supported only for Consumers with Administrative level authorization, as a possible way of adding messages to a Consumer's Queue.</p> <p>Unsupported otherwise.</p>  |
| Delete                          | Remove the indicated message from the Queue Instance. Note that it also possible (and more efficient) to delete messages from the Queue using optional functionality in the Query Messages operation.  |
| Update                          | Unsupported.   |

## 9.2 Queue Instance Object

The following data elements comprise the Queue Instance Object, and represent all the information available to a Consumer about its own Queue Instances, and / or to Administrative applications about the Queue instances currently assigned to all Consumers in its Environment.

| Element / @Attribute | Ch | Description   | Type   |
|----------------------|----|---|--|
| queue                |    |   |  |
| queue@id             | M  | The ID of the Queue Instance element in the Queues Registry.<br><br>A Queue may be assigned as the destination for the delivery of asynchronous Events by specifying its Queue ID when creating a <i>subscriptionRequest</i> <sup>26</sup> or the destination for a delayed Response by specifying it in the Request.                 | xs:token (may or may not be a UUID) as this ID is not persistified.        |
| queue/polling        | O  | Determines, when no messages are queued, whether the Consumer will have to periodically reissue Polling Requests at timed intervals or can reissue a Polling Request immediately.   | One of:<br><br>IMMEDIATE<br><br>LONG<br><br>The default value is IMMEDIATE |
| queue/ownerId        | C  | The Environment ID of the owning Consumer for which this Queue is buffering messages.<br><br>It is not specified at Queue <i>create</i> time, but is returned as part of the response to a <i>query</i> operation, and serves to allow an administrative application to identify which application the Queue instance is assigned to. | xs:token   |
| queue/name           | O  | A name which the Consumer assigns to the Queue. It is useful for reporting purposes, but is not guaranteed to be unique.  | xs:token   |
| queue/queueUri       | M  | The URI of the "Messages" Service associated with the Queue instance. This is the endpoint where the "get next message" requests are issued.  | xs:anyURI  |

<sup>26</sup> See the following section which describes the Subscription Service.

|   |   |  |  |
|---|---|--|--|
|   |   | Its value is typically “queues/{queueId}/messages” but it can be different.  |  |
| queue/ownerUri                                  | O | When present, this contains the URL which the Queue instance will use when it determines the owner needs to be asynchronously alerted that a new message has arrived in the Queue. In that case, it will send a “wake-up” HTTP POST whose payload contains a single integer indicating the current number of messages queued up for the Queue Owner.<br><br>This value is provided at Queue Instance creation.   | xs:anyURI                              |
| queue/idleTimeout                               | M | Non-zero only if the Queue instance is “Long Polling”. If so, this is the maximum time in seconds the Messages Service will wait for a message to arrive, before returning an HTTP Response of “no message seen” to the Consumer in response to an open HTTP Request for the next message.<br><br>Its upper end value is “suggested” by the Consumer on the Queue <i>create</i> Request, but may be limited by the Service to a value less than that.  | xs:unsignedInt                         |
| Queue/minWaitTime                               | C | Not specified during Queue creation. It is returned in the resulting Queue object by the Queue Service.<br><br>The value is generally zero only if the polling value is “LONG”. Otherwise it indicates the minimum time in seconds that the Consumer should wait after receiving a “no message seen” response to a previous <i>query</i> , before posting another.<br><br>If the Consumer does not wait that long before issuing the next Query, rather than being accepted, the response <b>should</b> be rejected with a General Service Error of “Service is Busy”. | xs:unsignedInt                         |
| queue/maxConcurrentConnections                  | O | “Suggested” by the Consumer on the Queue <i>create</i> Request, but may be limited by the Queue Service to a value less than that.   | xs:unsignedInt<br>(default value of 1) |
| Statistical / Preventative Maintenance Elements |   |  |  |
| queue/created                                   | M | Time that Queue Instance was Created   | xs:dateTime                            |

|                    |   |  |                |
|--------------------|---|--|----------------|
| queue/lastAccessed | M | Time that Queue Instance was last Accessed and a message was dequeued. A value significantly greater than the maxWaitTime, with a messageCount greater than one may indicate the owning application may be inactive<br><br>Initialized to time of Queue creation.                          | xs:dateTime    |
| queue/lastModified | M | Time that Queue Instance was last Modified by receiving a new incoming message. A value significantly greater than the maxWaitTime indicates low message traffic (Events, delayed responses) from the data source(s) assigned to this Queue.<br><br>Initialized to time of Queue creation. | xs:dateTime    |
| queue/messageCount | M | The number of messages currently residing in the Queue. A high number may indicate the application is overburdened.  | xs:unsignedInt |

The XML example below reflects a typical “Long Polling” Queue Object payload, where the Queue will hold the connection open for up to 30 seconds after a Request for the next message is received, before returning failure due to no message being available. As is true with most Long Polling Queues, according to the minWaitTime setting, the Consumer can repost a new Request immediately. Only one simultaneous connection is allowed.

```
<queue id="53756efe-18d0-4fd4-84ce-fe824719426d">
  <polling>LONG</polling>
  <ownerId>ef87654fa0987dc09abc</ownerId>
  <name>TestQueue</name>
  <queueUri>http://www. .../queues/queue/message</queueUri>
  <idleTimeout>30</idleTimeout>
  <minWaitTime>0</minWaitTime>
  <maxConcurrentConnections>1</maxConcurrentConnections>
  <created>2013-06-03T14:42:36.330-07:00</created>
  <lastAccessed>2013-06-03T14:42:38.810-07:00</lastAccessed>
  <lastModified>2013-06-03T14:42:38.810-07:00</lastModified>
  <messageCount>7</messageCount>
</queue>
```

### 9.3 Message Service

There is only a single dedicated Messages Service URL (the `queueUri`) associated with the Queue Object, which is used by the Queue owner to request delivery of the “oldest” message in the queue. The Messages Service delivers all Event and delayed Response messages as immediate HTTP Responses to Consumer issued HTTP Requests.

There are several important options and considerations regarding how a Consumer may access its FIFO collection of asynchronously delivered messages.

#### 9.3.1 “Get Next and Pop” (Query with `deleteMessageId`)

There is an optional “`deleteMessageId`” matrix parameter in the *query* Messages operation which effectively provides the owning Consumer with the two Query alternatives shown below.

| Polling Operation   | Description   |
|---|---|
| <b>Query (no <code>deleteMessageId</code>)</b>  | <p>If no message is available for delivery, a “<i>noQueuedMessage</i>” error is returned. The Queue owner <b>must</b> delay for at least the number of seconds specified in the <code>minWaitTime</code>, before reissuing another Query operation to see if further messages have arrived.</p> <p>If there is at least one message on the Queue, it is returned in the HTTP Response. The owner should save the Message ID and process the message. When processing is completed the owner should issue a Query with this <code>messageId</code> as the “<code>deleteMessageId</code>” parameter (see below)</p>   |
| <b>Query (<code>deleteMessageId</code> set to the Message ID of the last successfully delivered and processed message obtained from this Queue)</b> | <p>If no message is available for delivery, a “<i>noQueuedMessage</i>” error is returned. The Queue owner <b>must</b> delay for at least the number of seconds specified in the <code>minWaitTime</code>, before reissuing another Query operation to see if further messages have arrived.</p> <p>If there is at least one additional message in the Queue later than the one with the specified Message ID, that is returned in the HTTP Response. The Messages Service then automatically “deletes” the identified message (i.e. removes it from the Queue).</p> <p>When message traffic is high, this option cuts down the number of messages exchanged between the Messages Service and its owner by a factor of two over the alternative of issuing a <i>query</i> followed by a separate <i>delete</i> for each message.</p> |

There are two “*get next and pop*” edge cases that deserve special consideration. Assume the Consumer issues a “*Query and delete last message*” to the Message Service, and the Service response “gets lost”. This leaves the Consumer in doubt over whether the last message has in fact been deleted. In this situation, the Consumer can “re-synch” by issuing a Query (without specifying the `deleteMessageId`).

- If the response contains the ID of the message that it had earlier indicated should be deleted, the Consumer can safely assume that its previous request had never made it, and **should** immediately re-issue the previous “*Query and delete last message*”.



- Otherwise the Consumer can safely assume that the previous request was successful and it was the Service response containing the just received message that was lost. In that case, since it now has the message at this point, the Consumer can proceed as normal.

From the Message Server's point of view, a problem can arise when it is asked to delete a message corresponding to a messageId it does not have, or which is not currently "outstanding" in terms of having been previously sent to the Consumer for processing.

- The response here should be the same as for handling any delete request containing an invalid messageId: send back an HTTP error message with code 404 (*"Not Found"*) to report this.
- Any Consumer receiving such a "could not find message to delete" error **should** post an Alert, but then could go back to normal message processing.

### 9.3.2 Long Polling

If during the *create* Queue operation, the Consumer set the polling element to LONG, a "special" type of Messages Service will be created which supports "long polling".

The default (IMMEDIATE) Queue behavior is that a Consumer issues a *query* operation to retrieve the next message, and is informed immediately if no message is queued. In periods of low message traffic, this might result in the Consumer issuing periodic *query* Requests every 60 seconds or so (assuming that was at or greater than the value demanded by the *minWaitTime* element). If a message arrives anywhere in that timeframe, it is not reported until the next Consumer Query arrives, introducing a possible processing delay of from 1 to 59 seconds until it is discovered.

However if the Consumer had created a LONG Polling Queue with an *idleWait* value of 60 seconds (and a *minWaitTime* value of 0), the Queue will hold the query, and either respond with the incoming message when it arrives, or, after 60 seconds, return the Query with a *noQueuedMessage* error. On receiving that error, the Consumer **should** immediately issue a new Query.

The end result is that by creating a LONG Polling Queue, the Consumer is always alerted immediately that a message has arrived, because it effectively always has a Query request pending. This functionality is identical whether there are one or multiple connections.

### 9.3.3 Support for multiple concurrent Messages Service Connections (optional)

A Consumer will receive messages from a Messages Service sequentially in FIFO order, and can create and utilize the services of one or more Messages Queues. However, a given Consumer **may** also be written to support (when available) more than one multiple simultaneous concurrent connection with a single Queue, which could be useful in periods of high message traffic

Such a Consumer may request this functionality when it creates the Queue instance by setting the value of *maxConcurrentConnections* to greater than 1. If the created Queue also supports concurrent connections (it is not required to do so) the request will be honored. Specifically the value of *maxConcurrentConnections* would

be set to a value greater than 1 in the resulting Queue object. Otherwise if either the Consumer or Service does not support this feature:

- The value of `maxConcurrentConnections` is set to 1 and only one Queue Connection is supported
- No Message *query* Request or Response can include a *connectionId* HTTP header element

If multiple Concurrent Connections are supported, the HTTP header element *connectionId* **must** be used by the Consumer to identify which Connection each given query operation applies to. When provided for the first time in a Query, it indicates a new concurrent connection is to be initiated and assigned that ID (assuming the maximum concurrent connection limit has not been reached). When the *connectionId* matches that of an existing connection, the query is processed concurrently, specific to that connection stream. The successful response to such a query will always include the *connectionId* contained in the original Request, to which the returned message applies.

### Server-side Requirements

The Messages Service must maintain a separate “last message” for every active connection. Upon receiving any *query* Request, a Messages Service which supports multiple Concurrent Connections should reject the Request with the “Not Found” Error<sup>27</sup> if:

- The *connectionId* element is not included in the HTTP header
- The *connectionId* is new (implying a new connection is being requested) but the specified connection limit has been reached.
- The *connectionId* is new, but a *deleteMessageId* parameter is included.

Queries issued using a *connectionId* will execute in parallel with active Queries issued using other *connectorIds*, resulting in multiple HTTP Requests awaiting HTTP Responses. From the Messages Service perspective, there is a single FIFO queue of messages.

Whenever the Messages Service returns a message in response to a Query request, that message is “pulled” from the single FIFO queue and associated with the connection on which the Query was returned. After that point:

- A new Query with a matching *deleteMessageId* and known *connectorId* arrives. The retained message is freed and a new message is extracted from the FIFO queue, sent back in response, and associated with this connection.
- A Query without a *deleteMessageId* but with this *connectorId* arrives. The retained message is returned.

There is a further consideration that must be understood by Consumers availing themselves of multiple simultaneous connections to the same Messages Queue.

---

<sup>27</sup> Please refer to the appropriate table in Appendix C.

- **Messages may be processed out of order:** Assume two events (create Object X, update Object X) are published “back to back”, and are handed off to Consumer M’s Queue connections A and B respectively.

Thread A working to process the *update* may reach the point where it is trying to identify an internal record corresponding to the ID of object X, before thread B which is concurrently creating object X has saved its ID. Specific multithreading logic may be required to prevent this type of confusion that could be ignored otherwise.

### 9.3.4 Support for “Wake Up”

If the Consumer includes the *ownerURI* element in a *create* Queue Request, then either:

- A special Queue Instance will be created that will issue asynchronous “wake up” HTTP POST Requests back to that URL when it determines the owner needs to be asynchronously alerted that a new message has arrived in the Queue. This is typically (but not required to be) whenever the arriving message is the first message in the Queue. The payload of each “wake up” POST will contain a single unsigned integer containing a value less than or equal to the number of messages currently in the queue.
- An HTTP error of 405 (*Service not Allowed*) will be returned if this type of Queue is not available. In that case the Consumer **should** reissue a Create Queue request without the *ownerURI*.

Where available, a “wake up” Queue Instance can serve as an alternative to “Long Polling”, as it eliminates message response latency times without requiring the presence of a continuously open HTTP connection. The owning Consumer can create such a Queue, and use it as the destination when issuing delayed Responses or Event subscriptions. At that point the following sequence of operations goes into effect:

- A message arrives in the Queue Instance. If no messages were previously in the Queue, a “wake up” HTTP POST is dispatched to the URL originally contained in *ownerURI*.
- The Consumer processes the wake up, and in response, issues a “*get next and pop*” request to the Queue Instance associated with the URL the “wake up” arrived at.
- If the *get next and pop* was successful, the Consumer processes the returned message and repeats this step.
- If the *get next and pop* failed because the Queue Instance was empty, the Consumer should close the HTTP connection to the Queue Instance.
- This entire sequence will repeat when the next asynchronous message arrives..

Note that any Consumer creating such a Queue Instance **must** be capable of receiving and processing incoming HTTP Requests (i.e. the Consumer must be capable of functioning as a web service).

## 10. Subscriptions

A Consumer uses the *Subscriptions* Infrastructure Service to *subscribe* to Events published by one or more Service Providers. The Events of each Provider will be asynchronously delivered in FIFO order to the assigned Queue, and can then be accessed by the Consumer.

### 10.1 Supported Operations

A *subscription* may either be created or deleted. Any Consumer which intends to subscribe to Events from Service Provider X must first:

- Ensure it has authorization to subscribe to data from Service Provider X

This may have been pre-authorized by site administrators before the Consumer registered, in which case the subscribe access will be set to APPROVED in the Environment, or it may involve successfully creating a provisionRequest which asserts that authorization. For Report or Composite object types that do not issue Events, the access value should indicate UNSUPPORTED, and any *create* Subscription request will fail.

- Successfully create at least one Queue Instance
- Successfully create a Subscription Instance which joins the specific Service Provider and the specific Queue

At that point, Events published by the Service Provider will begin asynchronously arriving in the Queue Instance, and can be retrieved by the Consumer in FIFO order. They will stop arriving when and if the Consumer deletes the *subscription* object.

| Operation | Description   |
|-----------|---|
| Query     | Returns the contents of one or more Subscription objects belonging to the Consumer issuing the Query.   |
| Create    | Create a Subscription.<br><br>Only one subscription can be created (singular form only). Success will result in the future arrival in the specified Queue, of any Events published by the specified Service Provider.<br><br>If this Consumer is already subscribed to the specified Event type, the <i>create</i> request will fail. |
| Delete    | Delete a Subscription. Published Events for this object type will no longer be placed into the specified Queue, although existing Event messages will remain until polled for by the Consumer.  |
| Update    | Unsupported. Once a Subscription is created, all supplied data about the object type subscribed to and the Queue assigned to receive Events are immutable.  |

### 10.2 Subscription Data Elements

The following data elements comprise the contents of a Subscription Object. All are immutable.

| Element                         | Char | Description   | Type   |
|---------------------------------|------|---|--|
| <b>subscription</b>             |      | Individual Subscription   |  |
| <b>subscription@id</b>          | M    | The unique id of the Subscription.  | xs:token   |
| <b>subscription/zoneld</b>      | M    | The unique Zone in which the Service Provider may be found. If none was specified in the Subscription <i>create</i> Request, the Consumer's default Zone value is used. | xs:token   |
| <b>subscription/contextId</b>   | M    | The Context of the Event data which is supplied by the Service Provider. If none was specified in the Subscription <i>create</i> Request, a value of DEFAULT is used    | xs:token   |
| <b>subscription/serviceType</b> | M    | The "generic" type of Service being provided.   | One of:<br><br>* "UTILITY"<br>* "OBJECT"<br>* "FUNCTION" |
| <b>subscription/serviceName</b> | M    | The name of the Service (ex: "students")  | xs:token   |
| <b>subscription/queueId</b>     | M    | The identification of the Queue which will receive the asynchronous events when published by the Object Provider.   | xs:token   |

### 10.3 XML Examples

The payload of a *create* Subscription request for Student Object Events is shown below. Note the Subscription ID attribute is an empty token. The value will be supplied by the Subscriptions service.

```
<subscription id="">
  <zoneld>SuffolkMiddleSchoolSI</zoneld>
  <contextId>DEFAULT</contextId>
  <serviceType>OBJECT</serviceType>
  <serviceName>Student</serviceName>
  <subscriptionName>RamseyPortal</subscriptionName>
  <queueId>ef87654fa0987dc09abc</queueId>
</subscription>
```

## Appendix A: The Role of the Environment Administrator

The responsibilities of a SIF 3.0 solutions Administrator are outlined below.

### A.1 Setting up the Environments

- Determine the “types” of solution environments that will be created (such as *test*, *staging* or *production*).
- Within each Environments type, create multiple “Zones” each corresponding to a different level within the educational organization such as a school or district. Optionally create additional Zones dedicated to one aspect of that organization such as Special Ed students.
- Install any associated Utility Services
- Pre-provision the Zone assignment(s), authorization rights and resource limits for a set of applications, each having a known authentication identifier.
- Pre-provision any associated XML filters (to limit the set of elements the application can access within the set of objects it has already been approved to access).

### A.2 Administrating the Environments

- Ensure critical Service Providers successfully register as Consumers, and subsequently register their objects and functions in the desired Zone or Zones.
- Optionally review and respond to additional provisioning requests from existing Consumers already registered into an Environment.
- Monitor the Alert Logs and detailed information from individual Queues to allow early detection of inactive applications and / or network errors.
- Monitor the per-connection message traffic statistics from the Collector and Queues to measure and track solution performance and scalability.

## Appendix B: Connection Topologies

There are three types of connection topologies available to developers, integrators and end users who are constructing and deploying SIF solutions.

- SIF 2.x “Classic” Zone
- SIF 3.0 Direct Environment
- SIF 3.0 Brokered Environment

The following table highlights the differences between them.

| Alternative / Functionality   | SIF 2.6 / 2.7 “Classic” Zone   | SIF 3.0 Direct Environment  | SIF 3.0 Brokered Environment   |
|-------------------------------|--|---|--|
| <b>Summary</b>                | SIF 2.x Data Model compliant XML exchanged via the Classic SIF Brokered infrastructure   | SIF 3.0 Data Model compliant XML exchanged over a direct connection between Consumers and a single Provider implementation                      | SIF 3.0 Data Model compliant XML exchanged between Consumers and multiple Providers via an intermediary.   |
| <b>SIF Data Model Payload</b> | SIF 2.x  | SIF 3.0, 3.1, ...   | SIF 3.0, 3.1, ...  |
| <b>Data Model Flexibility</b> | No. Infrastructure is tied to specific version of the SIF Data Model   | Yes. Infrastructure is independent of data payload contents.  | Yes. Infrastructure is independent of data payload contents.   |
| <b>Payload Format</b>         | XML only   | XML only over the wire. JSON conversions defined  | XML only over the wire. JSON conversions defined   |
| <b>Middleware required?</b>   | Yes. The ZIS   | No. All communications in a Direct Environment are 2-Party  | Yes. All data exchanges are mediated by a 3rd Party Message Broker / ESB utilizing middleware APIs.  |
| <b>Flexibility</b>            | Additional data object types may be provided by new Zone registrants who provision to supply them.<br><br>They can then be queried and updated by all registered clients in the Zone with the proper Authorization rights. | The Consumer can query and update data of only the fixed set of Provider Services supported by the application implementing the Zone interface. | Additional data object types may be provided by Consumers who are authorized to provision themselves as Service Providers<br><br>Their data can then be queried and updated by all registered Consumers with the proper Authorization rights.<br><br>The Brokered Environment extends but does not replace the functionality provided to Direct Environment Consumers. |

|                                 |  |  |  |
|---------------------------------|--|--|--|
|                                 |  |  | As a result, any Consumer limiting itself to Direct Environment functionality is capable of being redeployed within a Brokered Environment as well.  |
| <b>Zones</b>                    | Zones are totally isolated. There is no client / service communication unless both are registered in the same Zone.  | Anything more than a single Zone accessible within a Consumer's Direct Environment represents optional functionality.  | There are as many Zones as necessary for the needs of the organization. These are accessible within a Consumer's Brokered Environment, and each Consumer is assigned a default Zone.   |
| <b>Application Interactions</b> | <p>Multiple clients are connected to multiple Object Providers in the Zone.</p> <p>All Client and Object Providers in a Zone can see each other's published Events</p> <p>Any client can provision itself as an Object Provider and can discover other clients in the Zone.</p> <p>Each client and Object Provider has access to only a single FIFO queue for its arriving messages.</p> | <p>Multiple Consumers are connected to the Objects and / or Functions supported by a single Environments Provider implementation.</p> <p>Each Service Consumer is unaware of any other Consumers in the Direct Environment.</p> <p>Cross-Consumer communication is possible through Events ... if one Consumer updates the state or the data of a supported Service, every subscribing Consumer will see the corresponding Event.</p>                  | <p>Multiple Service Consumers are connected to multiple Service Providers in multiple Zones.</p> <p>As with the Direct Environment, Service Consumer clients can see the Events resulting from each other's Service invocations,</p> <p>Given proper access rights, any Consumer can provision itself as a Service Provider.</p> <p>Consumers and Service Providers can open multiple queues to prioritize certain message types and increase Zone performance scalability.</p> <p>Additional Utility Services are available to Consumers including a full featured Alert Log.</p>   |
| <b>Manageability</b>            | Management tools are whatever the ZIS product supplies, and traditionally this has been equivalent to the management functionality provided by the combination of an ESB and a Service Registry.   | <p>Management tools are whatever the application implementing the Environments Provider Interface supplies.</p> <p>While this is generally less extensive than management functionality provided by a ZIS, ESB or Service Registry, the deployment is smaller, and management of services is made much easier by the fact that all Service Provider interfaces are provided by the same implementation which provides the Direct Environments API.</p> | <p>In addition to the SIF 2.x Management functionality, Brokered Environment Providers may supply:</p> <ul style="list-style-type: none"> <li>• Retrieval of Alert Log information through straightforward Queries (ex: it is now possible to find out what was logged against application X)</li> <li>• Query capability of the Queue Manager statistics to determine the number of messages, size and time of last delivery of the messages in the queues of each Consumer and Provider across all the Zones of the Environment.</li> </ul> <p>This should be extremely useful in predicting and then isolating application-</p> |



|                         |  |  |   |
|-------------------------|--|--|---|
|                         |  |  | specific problems.  |
| <b>Typical Use Case</b> | <p>When a collection of applications at the school, district or regional levels need to interoperate NOW in support of an educational solution.</p> <p>All applications can be SIF-certified, and there is every expectation that each will interoperate “out of the box” with other SIF 2.x partners.</p> | <p>When an application only needs data from one other application.</p> <p>Consider a Student Contact application needing to access Student Name, Addresses and Phone Numbers from an SIS system, or a student dashboard application running on a mobile device.</p> <p>If the SIS system is a Direct Environments Provider, it enables both these applications to access its data directly without the need to install middleware.</p> <p>The Student Contact System can be additionally deployed in a SIF 3.0 Brokered Environment without change</p> | <p>When an organization is trying to construct and securely manage a complex group of intercommunicating applications.</p> <p>The SIF 3.0 Brokered Environment extends the remote management functionality of earlier releases (including both preventative maintenance and security).</p> <p>It supports a growing set of “Functional” services as well (which encapsulate educational processes such as Student Record Exchange”, “Locate Student” and “Assess Scoring” as well as provide data).</p> |

## Appendix C: HTTP Codes and Header Fields

A SIF Status Code will be returned in the HTTP Status field in the HTTP Header, whenever an Error object is returned in response to a Request. This field can have one of the following values:

### C.1 HTTP Success Codes

This class of 2XX and 3XX status codes indicates the action requested by the issuing Service Consumer was received, understood, accepted and processed successfully.<sup>28</sup>

| Error Code | Meaning         | Example of Use  |
|------------|-----------------|---|
| 200        | OK              | The standard HTTP response code for all successful HTTP requests, with the exceptions noted below   |
| 201        | Objects Created | One or more objects have been successfully created  |
| 202        | Accepted        | The SIF Request contained in the HTTP request has been accepted for routing, but the processing has not been completed. This is the status code returned in the HTTP response to every delayed SIF Consumer Request, as well as every published SIF Provider Event. |
| 204        | No Content      | All change Requests have Responses with contents. This is the response to a Query for which no existing object qualified.   |
| 304        | Not modified    | The specific response when a Query asks for objects which have changed, and none have   |

### C.2 HTTP Error Codes

An HTTP Error Code will be returned in the HTTP Status field in the HTTP Header, whenever an Error is returned in response to a Request. This field can have one of the following values:

| Error Code | Meaning         | Example of Use   |
|------------|-----------------|--|
| 400        | Bad Request     | XML error, version problems or error specific to a particular object type such as the omission of a mandatory element  |
| 401        | Unauthenticated | Illegal Consumer Authorization token accompanying the request  |
| 403        | Unauthorized    | Consumer Authorization token is legal, but Consumer is not authorized to issue the requested operation   |
| 404        | Not Found       | Object ID does not correspond to an existing object. This can occur for Query as well as Update or Delete operations<br><br>No Service Provider has been found to match the parameters (Zone, Context, Service name) in the Request. |

<sup>28</sup> A Service Provider issuing an Event to the Event Connector is treated as a Consumer issuing a Request.

|     |                        |  |
|-----|------------------------|--|
| 405 | Method not Allowed     | Paged Query Request issued to Object URL rather than Object List URL.<br>Create Request issued to Provider Registry Service in a Direct Zone.  |
| 409 | State Conflict         | An attempt has been made to create an existing object.<br>A Paged Query  |
| 412 | Precondition Failed    | An attempt has been made to modify an object when the Requester was not basing the modification on the latest version.   |
| 413 | Response too large     | A non-paged Query returning all objects was too large for the Service Provider (or Broker) to include in a single Response message.  |
| 500 | Internal Service Error | An unexpected error occurred in servicing the Request.   |
| 503 | Service Unavailable    | Returned only for Consumer Requests requiring an immediate Response. This error indicates that the expected Service processing time for the Request is great enough that the Consumer must reissue it as a Request requiring a delayed Response. |

### C.3 HTTP Header Fields

The following table describes the HTTP Header Fields and where they appear.

| HTTP Header Field Name | Where Used                                       | Explanation   |
|------------------------|--|---|
| Authorization          | Every Request                                    | It is used to authenticate the Consumer and is the basis for determining whether the Consumer has the necessary authorization to issue the Request.   |
| connectionId           | Request from multi-threaded queue consumers only | Identifies the established connection over which the next message in the Queue is being requested and delivered   |
| contextId              | Event  | The “context” of the service provided. The range of possible Context token values for a given Object or Functional type Service is defined by either or both the Data Model which the Environment is supporting, and the administrators of the Zone. If not provided, it will default to DEFAULT before delivery to the Service Provider in a Brokered Environment..<br><br>This is normally carried as a matrix URL parameter in Requests, but is returned as an HTTP Header field on Events |
| eTag                   | Response and then Request                        | Optionally returned by a Service Provider within a Query Response, equivalent to a “checksum” on <u>all</u> the objects of the type being queried, which are maintained by the Service Provider.<br><br>If it is returned in a Response, the Consumer <b>may</b> include it the next time it issues any Query to that Service Provider.   |
| eventAction            | Event  | The specific type of Event being reported: CREATE/UPDATE/DELETE   |

|                     |  |   |
|---------------------|--|---|
| generatorId         | Request and Event                          | The optional identification token of the "generator" of this request (ex: the administrative clerk who entered in the data that was responsible for generating a Create request).   |
| messageId           | All Messages                               | UUID  |
| messageType         | All Messages                               | One of: "EVENT"/"REQUEST"/"RESPONSE"/"ERROR"  |
| methodOverride      | Multiple Delete Request only               | Included in an HTTP PUT message when it is conveying a multiple delete request, since an HTTP DELETE is not allowed to have a payload. Only value is DELETE   |
| mustUseAdvisory     | Create Request only                        | Informs the Service Provider that if the "suggested" RefId in the Request cannot be assigned to the new object, the Request should be rejected  |
| navigationCount     | Paged Response                             | The total number of objects in the set of results generated by the initial Paged Query that is associated with the returned navigationId.   |
| navigationId        | Paged Response, then Paged Request         | Identifies state maintained in the Service Provider for the Consumer issuing the Paged Query Request. If returned, the Consumer must supply the navigationId value when requesting subsequent Pages of that object type from that Service Provider.   |
| navigationLastPage  | Paged Response                             | It is included as an aid for the Consumer in detecting when to stop issuing Paged Query Requests.   |
| navigationPage      | Request than Response                      | The number of the Page to be returned. If it is outside the range of results (which does not constitute an error) an HTTP Response with a code of 204 (No Content) will be returned.  |
| navigationPageSize  | Request, than Response                     | <p>This is included in every Request, and indicates the number of Objects to be returned in the corresponding Response Page. If the Page Size specified is too large for the Service or Environments Provider to supply, an Error with code 413 (Response too large) will be returned.</p> <p>When contained in the Response, it indicates the actual number of objects on the returned Page.</p> |
| queryIntention      | Request                                    | If the Consumer intends to follow up with further Paged Queries after this one, this field must be included in the Paged Query Request.   |
| queueId             | Delayed Request                            | <p>Contains the identity of one of the Consumer's assigned Queues to which the asynchronous Response from the Service Provider to this request <b>must</b> be routed.</p> <p>If this field is not present, the Request is immediate.</p>  |
| relativeServicePath | Response, inserted by Environment Provider | <p>Replicates all information contained in the segments of the Request URL following the Request Connector. This could include the Service name, XQuery Template name or Service Path defining the payload format, and any accompanying URL matrix parameters (Context and Zone).</p> <p>No URL Query parameters are included</p>   |
| replacement         | Change Event only                          | Set to FULL (current values of all object elements) or PARTIAL (only elements whose values have changed)  |

|                |   |  |
|----------------|---|--|
| requestId      | Request and then Response                 | A Consumer specified “token” that uniquely identifies every delayed Request issued by the Consumer. It could be as simple as a monotonically increasing integer. Used to correlate the delayed (asynchronous) Response with the original Request. It could be as simple as a monotonically increasing integer. |
| requestAction  | Request                                   | One of CREATE, UPDATE, DELETE or QUERY   |
| requestType    | Request                                   | One of IMMEDIATE or DELAYED. If not set, it defaults to IMMEDIATE.   |
| responseAction | Response                                  | This must exactly match the requestAction value contained in the HTTP header of the Request being responded to. (CREATE, UPDATE, DELETE, QUERY)  |
| serviceType    | Request, Response Event                   | One of UTILITY, OBJECT, FUNCTIONAL, SERVICEPATH or XQUERYTEMPLATE. If not provided, it will default to OBJECT  |
| sourceName     | All Request s reaching a Service Provider | Added by Broker to all Requests before forwarding to the Service Provider.<br><br>Used by the Service Provider in Brokered Environments when issuing an Alert concerning an erroneous Request. The value <b>must</b> uniquely identify the Consumer,   |
| timestamp      | All messages                              | Date / Time of Event creation (in ISO-8601 format also used as the basis of xs:dateTime)   |
| zoneld         | Events                                    | Zone which scopes possible service providers for a given Consumer Request, or qualifies the Service posting an Event.<br><br>This is normally carried as a matrix URL parameter in Requests, but is returned as an HTTP Header field on Events.  |

#### C.4 URL Matrix Parameters:

The following table describes the URL matrix parameters utilized in Request Messages.

| URL Matrix Parameter Name | Where Used                                   | Value   |
|---------------------------|--|---|
| contextId                 | Any request                                  | The “context” of the service. If not provided, the value will be set to DEFAULT before delivery to the Service Provider in a Brokered Environment.                        |
| deleteMessageId           | Query Requests to the Queue Service Instance | The ID of the last message received and processed by the Queue Owner  |
| zoneld                    | Any request                                  | The “Zone” of the Service.. If not provided, the value will be set to the Default Zone of the Consumer before delivery to the Service Provider in a Brokered Environment. |

#### C.5 URL Query Parameters:

| URL Query Parameter Name | Where Used    | Value  |
|--------------------------|---------------|--|
| where                    | Dynamic Query | A restricted XPATH expression which qualifies which among the set of all objects supplied by the Provider will |

|  |  |                                    |
|--|--|------------------------------------|
|  |  | satisfy the query and be returned. |
| [name matches XQuery Template parameter] | Supplies value to a parameter of the XQuery template whose token is in the last URL segment of the Query Request | Depends on the XQuery Template     |

## Appendix D: Common Types

Common and supporting types referenced in this specification are included here as a reference.

### D.1 ErrorType

| Element or @attribute | Char | Description  | Type or Value  |
|-----------------------|------|--|--|
| <b>errorType</b>      |      | Used to indicate an unsuccessful response.   |  |
| <b>@id</b>            | M    | The identity of the Error Object   | UUID   |
| <b>code</b>           | M    | Corresponds to the value contained in the HTTP Header Status field in which the Error Object is the payload. <sup>29</sup><br><br>Its presence allows the Error Object to be self-contained when / if persisted  | xs:unsignedInt<br><br>Ex:<br>* 400 (Bad Request)<br>* 404 (Object not found) |
| <b>scope</b>          | M    | Attempted operation. Ex: "Modify Student failure"  | xs:string<br><br>(xs:maxlength = 80)   |
| <b>message</b>        | M    | A simple, easy to understand, compact description of the error. The primary consumer of this message is the application user. Example: "Unable to open database."  | xs:string<br><br>(xs:maxlength = 1024)                                       |
| <b>description</b>    | O    | An optional error description that is more complete and technical in nature. It is to be used as a diagnostic message in trouble-shooting procedures.<br><br>Example: "The 'Students' table is opened in exclusive mode by user 'ADM1' (dbm.cpp, line 300)." | xs:string  |

### D.2 UUIDType

SIF format for a UUID.

| Element         | Char | Description  | Type or Value  |
|-----------------|------|--|--|
| <b>uuidType</b> |      | Unique Identifier of the form:<br><br>00000138-c8f6-4a48-000a-7b229286dd57 | xs:token<br><br>xs:pattern [a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[14][a-fA-F0-9]{3}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12} |

<sup>29</sup> See the HTTP Error Code section below for the complete list of such codes.

### D.3 VersionType

A SIF infrastructure version number

| Element            | Char | Description   | Type or Value   |
|--------------------|------|---|---|
| <b>versionType</b> |      | A SIF infrastructure version number of the form:<br><br>3.0. or 3.0.1 | xs:token<br><br>xs:pattern<br>[0-9]{1,3}[.][0-9]{1,3}([.][0-9]{1,3})? |

### D.4 PropertyType

| Element/@Attribute | Char | Description  | Type or Value           |
|--------------------|------|--|-------------------------|
| propertyType       |      | Value of Property.<br><br>When contained in an update Request or Event, a property value of xs:nil indicates that the parameter is being deleted rather than its value is being changed. | xs:token                |
| property@name      |      | Name of Property   | xs:token (maxlength 80) |

### D.5 ProtocolType

| Element/@Attribute  | Char | Description   | Type or Value |
|---------------------|------|---|---------------|
| <b>protocolType</b> |      | Contains Protocol Information regarding how to contact a Service Provider   |               |
| location            | M    | The URL to use when sending Requests to a Provider  | xs:anyURI     |
| properties          | O    | May contain zero or more <i>property</i> elements containing <i>name/value</i> pairs describing any protocol settings required to ensure proper communication |               |
| property            | OR   | Individual setting  | propertyType  |



## D.6 ProductIdentityType

| Element/@Attribute         | Char | Description   | Type or Value               |
|----------------------------|------|---|-----------------------------|
| <b>productIdentityType</b> |      | Contains Product information used in Consumer and Provider Registration, and returned as part of Zone Status  |                             |
| vendorName                 | O    | The name of the company supplying this Product.   | xs:token<br>(maxlength 256) |
| productName                | M    | The name of the Product   | xs:token<br>(maxlength 256) |
| productVersion             | O    | The format of this field is undefined, but it should match the format used in the agent's conformance statement, if the agent is SIF Certified.   | xs:token<br>(maxlength 80)  |
| iconUri                    | O    | HTTP URL referencing an icon for graphical representation of the application/agent. Should range from 16x16 pixels to 128x128 pixels and be of an image MIME type commonly supported by Web browsers (e.g. PNG, JPEG, GIF). | xs:anyURI                   |

## D.7 InfrastructureServiceNameType

| Element/@Attribute                   | Char | Description  | Type or Value  |
|--------------------------------------|------|--|--|
| <b>infrastructureServiceNameType</b> |      | A choice list containing one of the names of the infrastructure services that together form the basic Consumer Environment | An enumerated list restricted to the following values:<br><i>environment</i><br><i>requestsConnector</i><br><i>eventsConnector</i><br><i>provisionRequests</i><br><i>provisionRequest</i><br><i>queues</i><br><i>queue</i><br><i>subscriptions</i> |

## D.8 UtilityServiceNamesType

| Element/@Attribute             | Char | Description   | Type or Value  |
|--------------------------------|------|---|--|
| <b>utilityServiceNamesType</b> |      | <p>A choice list containing one of the names of the defined utility services that provide infrastructure (rather than Data Model or Educational Process) functionality.</p> <p>Not all of these may be present in every Zone.</p> | <p>An enumerated list restricted to the following values:</p> <p><i>alerts</i></p> <p><i>zones</i></p> <p><i>providers</i></p> <p><i>xQueryTemplates</i></p> <p><i>codeSets</i></p> <p><i>nameSpaces</i></p> |

## Appendix E: Acknowledgements

SIF 3.0 is an open standard.

Architecting, designing, documenting and validating the SIF 3.0 Infrastructure technology was a complex and challenging effort which would have been impossible without the contributions of many SIF members. The efforts and accomplishments of the following core group of participants in particular deserve to be recognized.

In alphabetical order, they are:

- Bill Duncan – Principal Architect, Pearson
- Ian Tasker - Director of Innovation and Development, SchoolsICT Limited
- Joerg Huber - SIF Solution Architect, Systemic Pty Ltd
- Robert Hutchinson – CEO, VisualSI
- John W. Lovell - Senior Systems Developer, SIF Association
- Mike Reynolds - Technical Architect, TIES
- Ron Kleinman - CTO, SIF Association
- Xavier Wiechers - Principal Developer, Pearson