Access 4 Learning Community

Powered by: SIF

Simple. Secure. Scalable. Standard.

SIF Infrastructure Specification 3.2.1: Base Architecture

www.A4L.org

Version 3.2.1, March 2017

1. Introduction

SIF 3.0 infrastructure represented a major release of the SIF standard. Currently unused functionality in SIF 2.x was deprecated or replaced entirely, and significant new, non-backward compatible functionality has been added.

As with the release of previous major versions of the SIF standard, extremely valuable feedback regarding confusing or inconsistent statements, specification conflicts with unexpected developer tool limitations, and important missing functionality was received from document reviewers, interested developers and early adopters during the first 90 days after the specification was released to the public. This resulted in follow up efforts to address all these concerns

The end result is the creation of SIF Infrastructure 3.0.1. This was a “fix release”, correcting errors inadvertently contained in SIF 3.0 and standardizing additional requested SIF infrastructure functionality. With the issuance of that version of the documentation we believe the basic goals of SIF 3.0 were completely met, in a stable, secure and powerful REST-based infrastructure release that is unlikely to be “broken” in the foreseeable future.

As this solid base gained in popularity attention was paid to ease of use. Particularly meeting the expectations of programmers leveraging large cloud services. With this 3.1 release access to data was streamlined, especially for those typical use cases where authentication (and roles) yields authorization.

The detailed set of changes to the SIF Infrastructure 3.0.1 documentation which are contained in this SIF Infrastructure 3.1 release are listed in Section 1.13 below.

1.1. Preamble

The Systems Interoperability Framework (SIF) is not a product, but a technical blueprint for enabling diverse applications to interact and share data related to entities in the pK-20 and workforce instructional and administrative environment. SIF is designed to:

- Facilitate data sharing and reporting between applications without incurring expensive customer development costs;
- Enhance product functionality efficiently; and
- Provide best-of-breed solutions to customers easily and seamlessly.

The SIF Implementation Specification defines architecture requirements and communication protocols for software components and the interfaces between them; it makes no assumption of specific hardware or software products needed to develop SIF-enabled applications and middleware service implementations, other than their ability to support technologies leveraged as the foundation for SIF.

1.2. Guiding Principles

The set of guiding principles used during its development determined that the SIF 3 Infrastructure documented here **must**:

1. Be re-usable without change to support the Data Model of any SIF locale (i.e. be payload independent).
2. Leverage functionality provided by existing infrastructure standards by making them normative where possible.
3. Provide a clear transition path for current suppliers and end users of both SIF SOAP and Classic (HTTPS) infrastructure technology.
4. Lower the barrier to entry for new vendors, thereby helping to increase the number of SIF infrastructure technology providers.
5. Not drop any existing functionality in the previous release which has been utilized by significant numbers of SIF adopters, without supplying a viable alternative (because this new release must still address all underlying use cases).
6. Strengthen overall data security.
7. Support “out of the box” interoperability between SIF conformant applications.
8. Extend the SIF standard into new developer environments such as REST without fragmenting SIF adopters into separate non-interoperable communities.

If any of those requirements are not met by the SIF 3 Infrastructure Standard, details of the specific violation should be reported back to the SIF Association (dba Access 4 Learning (A4L) Community) using the specification feedback instructions contained on the A4L website.

1.3. Disclaimer

The information, software, products, and services included in the SIF Implementation Specification may include inaccuracies or typographical errors. Changes are periodically added to the information herein. The SIF Association may make improvements and/or changes in this document at any time without notification. Information contained in this document should not be relied upon for personal, medical, legal, or financial decisions. Appropriate professionals should be consulted for advice tailored to specific situations.

THE SIF ASSOCIATION, ITS PARTICIPANT(S), AND THIRD PARTY CONTENT PROVIDERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY, RELIABILITY, TIMELINESS, AND ACCURACY OF THE INFORMATION, SOFTWARE, PRODUCTS, SERVICES, AND RELATED GRAPHICS CONTAINED IN THIS DOCUMENT FOR ANY PURPOSE. ALL SUCH INFORMATION, SOFTWARE, PRODUCTS, SERVICES, AND RELATED GRAPHICS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. THE SIF ASSOCIATION AND/OR ITS PARTICIPANT(S) HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS INFORMATION, SOFTWARE, PRODUCTS, SERVICES, AND RELATED GRAPHICS, INCLUDING ALL IMPLIED WARRANTIES AND CONDITIONS OF: MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT.

IN NO EVENT SHALL THE SIF ASSOCIATION, ITS PARTICIPANT(S), OR THIRD PARTY CONTENT PROVIDERS BE LIABLE FOR ANY DIRECT, INDIRECT, PUNITIVE, INCIDENTAL, SPECIAL, CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF USE, DATA, OR PROFITS, ARISING OUT OF OR IN ANY WAY CONNECTED WITH THE USE OR PERFORMANCE OF THIS DOCUMENT, WITH THE DELAY OR INABILITY TO USE THE DOCUMENT, THE PROVISION OF OR FAILURE TO PROVIDE SERVICES, OR FOR ANY INFORMATION, SOFTWARE, PRODUCTS, SERVICES AND RELATED GRAPHICS OBTAINED THROUGH THIS DOCUMENT OR OTHERWISE ARISING OUT OF THE USE OF THIS DOCUMENT, WHETHER BASED ON CONTRACT, TORT, STRICT LIABILITY, OR OTHERWISE, EVEN IF THE SIF ASSOCIATION, ITS PARTICIPANT(S), OR THIRD PARTY CONTENT PROVIDERS HAVE BEEN ADVISED OF THE POSSIBILITY OF DAMAGES. IF YOU ARE DISSATISFIED WITH ANY PORTION OF THIS DOCUMENT OR WITH ANY OF THESE TERMS OF USE, YOUR SOLE AND EXCLUSIVE REMEDY IS TO DISCONTINUE USING THIS DOCUMENT.

1.4. Certification & Compliance Claims

Though a product may be demonstrated to comply with this specification, no product may be designated as *SIF Certified*™ by an organization or individual until the product has been tested against and passed established compliance criteria, published separately. Organizations and

individuals that are currently paying annual membership dues to the SIF Association and dedicating resources to the initiative may also use the designation *SIF Participant* to describe their involvement with the SIF Association and SIF in marketing, public relations and other materials.

1.5. Permission and Copyright

1.6. Infrastructure Artifacts Overview

The new SIF 3.0 infrastructure is delivered via the following set of release artifacts.

Infrastructure Volume	Description	Primary Audience
Read This First	The overview, introduction and guide to the other SIF 3 Infrastructure artifacts.	Anyone interested in understanding the functionality of the SIF 3 Infrastructure
Base Architecture	Defines the “core” concepts and detailed service operation framework of the SIF 3 infrastructure, and is the base document on which all the other infrastructure volumes	Those interested in learning about the SIF 3 Infrastructure at a conceptual level.

	identified below depend. **This is the volume you are reading now.**	
Infrastructure Services	Defines the complete specification (data structures, operations, and actions) for the set of directly accessible infrastructure services that together comprise the SIF 3 Environments Provider Interface (comparable to the SIF 2.x ZIS interface). It details which operations must be supported and which are optional for both the Direct and Brokered Architectures.	Those interested in learning about the SIF 3 Infrastructure at a detailed level.
Utilities	Defines the additional set of Services providing secondary infrastructure functionality, which are accessed identically to Object and Functional Services. When added to the Infrastructure Services, the combination provides the complete description of the SIF 3 infrastructure.	A reference work for developers utilizing the full functionality of the SIF 3 infrastructure
Functional Services	Defines the additional infrastructure pieces needed to manage jobs and route messages in-order-to support multiphase services with a beginning, middle, and end.	Those interested in defining interaction with more than one step or supporting a predefined functional service.

1.7. Organization of Document

This volume, the *SIF Infrastructure Specification 3.2.1: Base Architecture*, will be of interest to technical readers, including software architects, developers and integrators. It defines the "logical core" of the SIF Infrastructure, which is utilized in defining both the Infrastructure and Utilities Services.

The documentation it contains is organized as follows:

- The *Introduction* includes material common to most SIF specification volumes:

- The SIF Association disclaimer
- Details regarding certification, compliance and copyright claims
- Document conventions and organization
- Guidelines under which the technology described in the document was developed,
- Overview of the major functionality enhancements provided with this release.

- The *Glossary* provides the vocabulary that will be used in this and all other SIF 3 infrastructure documents.
- Normative References to existing standards, XML Namespace usage and HTTP and other conventions used in this document are identified.
- *Basic Infrastructure Concepts* provide a component-by-component review of the SIF 3 infrastructure, and the framework for writing Application (object and functional) services from the viewpoint of both Service Consumers and Providers.
- *The SIF 3 Environment* details the message framework which underlays both Direct and Brokered Architectures. This includes documentation of the SIF Message Exchange Patterns, and the error handling and data security requirements, which must be supported.
- *The Service Operation Framewor*k (the bulk of this document) describes the operations comprising the complete Consumer and Provider interface (data structures, operations and actions) in both a Direct and Brokered Architecture.

1.8. Document Conventions Definitions

The first time a term or concept is defined, it may be ***emphasized***.

1.8.1. References

References to other works occurring in this text are given in brackets, e.g. [REFERENCE].

1.8.2. Terminology

The key words **must, must not, required, shall, shall not, should, should not, recommended**, and **may**, when indicated in lower case **bold**, **must** be interpreted as described in [RFC 2119].

1.8.3. Element Characteristics

The possible values of the "Char" column shown in the Element Definition Tables used throughout this and other documents include one of the following primary (and mutually exclusive) element characteristics:

- **M – Mandatory**. The element **must** appear whenever the complete data object is conveyed, so they **may not** appear in a partial Update or where explicitly excluded by a Query. If another message does not specify one or more Mandatory elements, the request is erroneous.
- **Q – ReQuired**. If the element appears in the original Create Event or is eventually included in an Update Event (i.e. once it is known to the Service Provider), it **must** be returned in all corresponding queries as if it were Mandatory.
- **O – Optional**. The element may or may not appear in any message relating to the object. The Provider need not support it.

  One or more of the following qualifiers may also appear with the above characteristics:

- **C – Conditional.** The element is treated as the accompanying primary characteristic only if the specified conditions are satisfied. Otherwise the element is omitted from the message. Specifically:
  - **MC** – If conditions are such that the element can legally be included, it **must** be
  - **OC** – If conditions are such that the element can legally be included, it **may** be.
- **I – Immutable.** The value of the element cannot be changed once it is supplied.
- **U – Unique.** The value of this element for each object of this type **must** be unique (ex: *ID*)
- **N – Non-Queryable**. The element value is often calculated (ex: an aggregate), and cannot be used as a search key in a conditional Query Request.
- **R – Repeatable.** The element may appear more than once.

1.9. Assumptions

This is the base architecture document for the SIF 3 infrastructure. It is independent of both Data Model and transport layer technology. As such, the following assumptions are made of readers of this specification.

They have an understanding of, or prior working experience with, one or more of the following:

- XML, XML schema, and the role and use of XML namespaces
- HTTP and HTTPS and the security, encryption and authentication features of the latter.
- Service Oriented Architecture (SOA) and such concepts as *interface*, *implementation, adapter* and *message queue.*
- Common middleware components including the Service Registry and the Enterprise Service Bus (ESB), as well as the associated functionality they provide.

1.10. Version Numbers

The SIF infrastructure is completely independent of the Data Model (SIF or otherwise) which defines the payload of the data it will carry in specific deployments. As a result, when the term "version numbers" is used below, the conventions discussed apply only to the Infrastructure version.

The SIF Infrastructure uses the following version numbering scheme:

major version.minor version.revision number

Major new versions typically introduce additions/changes that impact a significant percentage of SIF-enabled applications (e.g. making previously optional elements mandatory, removal of deprecated objects, elements or values). The first release of a major version has a minor version of 0 (ex: 3.0); major version numbers start at 1 and are incremented as major versions are released (1.0, 2.0, 3.0 …).

Minor infrastructure releases typically may include minor infrastructure extensions/changes that do not impact existing SIF-enabled applications, whether applications or middleware. The first minor version released subsequent to and within a major release has a minor version of 1 and is incremented as new minor versions are released (3.1, 3.2, …). If a significant number of minor release features is introduced in a specification, the SIF Association may decide to increment the minor version number by more than 1 (e.g. 3.1 to 3.5), though a number like 3.5 is not an indication of being halfway to a major release, as minor version numbers may be incremented significantly past 10 (3.10, 3.11, …) as minor version features are released.

Corrections resulting from identified errata, as well as textual changes, may be incorporated into a revision release. These typically include minor corrections to messages or data objects, corrections of typographical errors, or corrected/expanded documentation. If major errors in

any release are identified, a revision release may incorporate changes more typical of a major or minor release. First major and minor releases have a revision number of 0, which is omitted from the version number (3.0, not 3.0.0); subsequent revision release numbers start at 1 and are incremented as new revisions are released (3.0.1, 3.0.2, ...).

This document pertains to the SIF 3.2.1 Infrastructure release.

1.11. SIF 2.6 Infrastructure Functionality not carried forward

SIF 3 provides an extensive set of new infrastructure functionality when compared to its immediate SIF 2.6 predecessor. However several pre-existing features have been EOL'd and replaced with alternatives that address the underlying use cases.

The list of functionality not carried forward in its earlier form, includes:

- Selective Message Blocking capability (used to block delivery of Events so Responses to issued Requests, could be received immediately) is gone, superseded by the ability to declare, assign and utilize multiple input message queues.
- The ability of a non-Object Provider to publish an Add, Update or Delete Object Event is gone, superseded by the ability to send a request for the equivalent change operation directly to the Object Service Provider. In SIF 3, only the Service Provider can post Events, and it **must** do that whenever there is a change to the underlying data it supports (whether or not this change was the result of a previous request). As a result, other applications can now "synchronize" themselves to a Data Provider by processing each arriving Event.
- The "Bundled Events" capability introduced in SIF 2.6 has been superseded with a more general "multiple-object" create, update and delete operations and events defined for all Object Services.
- The Response *Message Packet* functionality has been superseded by the more general "Paged Query Response" functionality, which supports both interactive and Batch mode Query Requests.
- The SIF-specific Extended Query syntax has been superseded by standard XQuery notation, which may be handled like a named query.
- Directed Read requests routed to a client-specified secondary Object Provider have been superseded with normal Query requests targeted at an Object Service of the specified type, with a "secondary" context.

- The “classic” SIF transport has been replaced with support for the REST transport and design patterns, which now form the basic underlying layer of the SIF 3 infrastructure.

1.12. New functionality introduced in SIF 3.0

SIF 3.0 represented a major release of the SIF Infrastructure, and as such it introduced a wide range of new functionality. Of particular note are three groundbreaking design advances that satisfy long standing requests from SIF 2.x developers and implementers. They are summarized below.

1. **The infrastructure is based on REST technology and design patterns.**

   REST is the underlying technology used to provide the foundation for exchanging SIF-compliant data. The specific changes made include:

   - The SIF 3.0 infrastructure Service Architecture has been aligned with the Get/Post/Put/Delete standard REST operations to make them more easily map-able to a set of standard REST resources.
   - The new “immediate return” option for Request / Response exchanges conforms to common REST usage.
   - An “interactive query” option which supports a set of incremental page reads has been introduced which closely maps to a common REST design pattern.
   - The SIF REST Developer Sandbox (SIF-RS) utilizes and illustrates the agreed on set of REST Developer-specific mapping specifications (including URL query parameters and HTTP Header arguments) that will guarantee REST applications can smoothly interoperate in a SIF 3 Environment.

2. **The infrastructure does not mandate the deployment of middleware components**

   In addition to the Brokered (3-party) Environment SIF architecture which had previously defined all SIF 2.x deployments, there is now a new Direct (2-party) option in which a Client Application “consuming” a SIF Service can connect directly with the “provider” of that Service. There is no longer a requirement to interpose generic message broker middleware such as the Zone Integration Server (ZIS) between them.

   Essentially a relatively thin Environments Provider Interface wrapper is placed around an Object Service, to allow such a Direct Architecture to be provided by an SIS or LMS system.

This wrapper (the "*Direct Architecture Interface*") is a subset of the "*Brokered Architecture Interface*" which provides access to a broader set of middleware services. As a result, ***every SIF 3 client application which was written to function in a Direct Architecture will also function in a Brokered Architecture***

There are several use cases for a Direct Architectures that are of particular interest:

- When an application wants to make its SIF-compliant data accessible to multiple users running simple RESTful client applications on mobile devices (much as the application might already be available to users via a browser). In such cases where only simple Request/ Response sequences are needed (i.e. no Events or asynchronous IO) the "Lite" form of the Direct Architecture Interface may be supported, which is very little more than is provided to a client of any RESTful Service.
- When an application only utilizes data from one other application. An example is a Student Contact application needing to access / update Student Name and Addresses and Phone Numbers from an SIS system. If the SIS system provides a Direct Architecture interface, this enables the Student Contact system to access its SIF-compliant data directly without the need to install middleware.

All applications (client or service) in the above examples can be SIF Certified, and there is every expectation that each will interoperate "out of the box" with its opposite number.

3. The infrastructure is independent of the Data Model defining the payloads it carries

In a major advance from SIF 2.x, the SIF 3.0 infrastructure can be utilized without change to carry payloads conforming to locale-specific SIF releases in the AU, UK and US. The SIF 3.0 Infrastructure version is decoupled from the SIF Data Model version and this allows SIF-conformant infrastructure products to be sold globally "out of the box".[1]

4. Other important advances provided by this release

These advances include:

[1] *The fact that the SIF 3.0 infrastructure is payload-independent enables it to securely and efficiently support exchanges of any data whether the format of that data is defined in the SIF specification or not.*

- Supports “Service Paths” that provide data for common uses cases
- Support for Named XQuerys and dynamic Where XPath driven query expressions
- Increased performance scalability, especially during periods of high message traffic
- Extensive new organizational support for centralized Service Management and Administration

1.13. Changes introduced in SIF 3.1

The major differences between SIF Infrastructure 3.1 and SIF Infrastructure 3.0.1 are summarized below.

- Simpler authentication through stronger SSO accommodation (including OAuth)
- Simpler consumers through seamless Environment creation (pre-configuration)
- Simpler apps through JSON support (Gossner Notation)
- Simpler messages though fewer HTTP Headers (defaults & query parameters)
- Simpler Environment Providers through optional utilities (alerts is now optional)
- Simpler navigation through ordered results sets (order query parameters)

1.14. Changes introduced in SIF 3.2

The major differences between SIF Infrastructure 3.2 and SIF Infrastructure 3.1 are summarized below.

- Addition of “Query By Example” (See 5.9 for details)
- Addition of “Changes Since” Functionality (See 5.10 for details).
- Mime Types Support Generalized.
- Addition of HEAD functionality for Request Connector (See 5.15 for details).
- Named XQuerys Parameters are now enumerated outside the Script (See Utility Services 6 for details)
- Functional Services now have their own connector and job object definition (See Infrastructure Services, section 11 and the new Functional Services document)

1.15. Why was a 3.2.1 necessary?

The primary driving factor in creating this fix release was difficulties encountered when using the jobId to setup and deliver events to the owning Adaptor. The updates enumerated below covers changes in this mechanism and other points of confusion discovered through real world usage. Please use this release instead of SIF Infrastructure (Global) 3.2.

- Clarified OAuth authentication in the Environment (using “Bearer”).
- Dropped footnote about Directed Events as that concept is not yet fully describe in the specification.
- Dropped jobId header.
- Added option for service consumers to setup a subscription for job objects (similar to any other object).
- Added fingerprint header.
- Added a fingerprint element to the environment.
- Added “initialization” to the job object as a more scalable way to start one or more jobs, especially when many are created at the same time.
- Clarified the role of the applicationKey throughout the documentation.
- Dropped references to XML Filtering and other possible “value add” technology that are not explicitly described as part of the specification.
- Corrected the Code Sets Registry utility service XML Schema to *not* require a source and namespace throughout.
- Relaxed the contentType (and Accept) header to allow for any media type throughout the documentation.

2. Infrastructure Overview

This section provides an overview of the SIF 3 Infrastructure. It is intended to serve as both an initial tutorial and later reference for infrastructure knowledgeable developers and architects, whether or not they are familiar with the infrastructure supporting SIF 2.x

2.1. Glossary of Terms and Concepts

The following infrastructure terminology will be used during the remainder of this document, and all other documents describing the SIF 3 infrastructure. The individual terms are defined in the table below and will be referred to, repeated and expanded upon in subsequent sections of this and other Infrastructure documents.

Term	Meaning
Basic Terminology	**The terms used in the definitions of other terms**
Data Object Type	A collection of elements that has some coherent meaning, which is collected under a single complex element, given a unique name and treated as a single data entity. Similar to a "Class" in an object oriented programming language. A Data Object Type is defined by its corresponding XML Schema.
Data Object	An "instantiation" or instance of a Data Object Type. A Data Object is created as a tree structure and can be validated against the XML schema of the Data Type which it represents.
ID	The unique and immutable identifier of a specific Data Object
Service	A set of defined functionality encapsulated behind a standardized CRUD[2] interface. Services are broadly categorized into Infrastructure, Utility, Functional, and Object.

[2] *"Create, Read, Update and Delete", corresponding to the REST POST, GET, PUT and DELETE operations.*

<table>
<tr><th>Infrastructure Components</th><th>The set of Infrastructure Building Blocks</th></tr>
<tr><td>Service Consumer</td><td>A Service Consumer implementation makes requests of, and subscribes to and receives Events from, one or more Service Provider components.
Ex: A Teaching & Learning cloud application, supporting a course, requesting enrolled students.
SIF 2.x Equivalent: Subscriber Agent</td></tr>
<tr><td>Service Provider</td><td>A Service Provider implementation accepts, processes and responds to requests from Consumers for object type or function-specific services, and publishes related Events in accordance with the type of Service Provider Interface it is implementing.
Every independent Service Provider initially registers as a Service Consumer, and may be (and generally is) a Consumer of one or more other Services.
SIF 2.x Equivalent: Object Provider</td></tr>
<tr><td>Environments Provider</td><td>An Environments Provider reliably and securely connects the Service Consumers to the Service Providers by implementing a set of Infrastructure Services which taken together, comprise the Environments Provider interface.
The implementer of the Environments Provider Interface may optionally (and transparently) implement one or more Services Provider Interfaces as well. In the case of a Direct Architecture, this is exactly what it must do.
SIF 2.x Equivalent for a Brokered Architectures Provider: ZIS
There is no SIF 2.x Equivalent for a Direct Architectures Provider</td></tr>
<tr><td>SIF Adapter</td><td>There are two basic varieties of Adapter, each corresponding to an earlier SIF 2.x equivalent:
<table>
<tr><th>SIF 2x. Component</th><th>SIF 3.0 Adapter equivalent</th></tr>
<tr><td>Subscriber Agent</td><td>Service Consumer Adapter (or in simple cases, a straightforward RESTful Client)</td></tr>
<tr><td>Object Provider Agent</td><td>Service Provider Adapter (Brokered Architectures only)</td></tr>
</table>
Essentially each Service Consumer and Service Provider application may include a separate adapter component to communicate with other applications via the Environments Provider interface. For example, a SIF 3 Brokered Architecture may integrate a student information application, a learning management application, and a library automation application. An adapter component acts as a bridge</td></tr>
</table>

	between each application and the Environments Provider interface. Adapters never communicate with other adapters directly. Instead, each adapter uses the Environments Provider interface as a trusted intermediary that brokers the exchange of data with other adapters.
Environment	The "SIF 3 Environment" is made available to a Service Consumer when it initially registers. It comprises the totality of every service the Consumer might possibly provision itself to access. Based upon authentication constraints however, the Consumer's access to some services it can see might be restricted. The Environment is defined by the set of Infrastructure Service URLs returned to a Service Consumer in response to a successful Registration Request or referenced by URI when making a data request where preregistered or automatically registered or dynamically created. These URLs allow the Environments Provider to provide a "customized" environment for each Consumer. For example, depending on the authentication provided by the Consumer at registration time, the URLs returned might insert it into either a production or test environment, or one that provides access to only a limited subset of authorized available Service Providers. As noted, the physical topology behind the Consumer's Environment interface can take one of two forms, each of which is described below.
Defined Architectures	**Implementation Topologies or Environment Types**
Direct	A Direct Architecture connects a single Consumer to a fixed set of one or more directly accessible Service Providers. These include, at a minimum, the mandatory set of Infrastructure Services, all mandatory Utility Services and at least one Data Object or Functional Service.[3] A Direct Architecture conceptually does not leverage middleware. All Consumer to Provider connections are direct (no intermediary), because the Environments Provider Interface and all Service Provider Interfaces are implemented by an Environments Provider Adapter front-ending a single application (such as an SIS or LMS). *This means that a Service Consumer cannot dynamically provision itself as a Service Provider when registered in a Direct Architecture.* Such an Adapter implementation could simultaneously provide a separate

[3] *All these service types are defined elsewhere in this glossary.*

	Environment to each of several Service Consumers, to enable them to directly access and update the data of its provided application. In that common case: • Each Service Consumer is operating in an Environment of its own, and has no knowledge of any other Consumers • When any Service Consumer request causes a change to the data in the Service application, every appropriately subscribed Service Consumer in every Environment supported by the Environments Provider Adapter receives the identical corresponding Event. **Details**: Introduced in SIF 3, a Direct Architecture standardizes SIF-compliant message exchanges between Consumer and Provider in the absence of a central Message Broker As described earlier, the typical Service Consumer registered in a SIS-provided Environment could be a simple data entry application running on a mobile device, or a Student Contact system that only needed to access the Student's ID, Name, Addresses and Phone Numbers.
Brokered	The Brokered Architecture securely and reliably connects N Service Consumers to a dynamically changing list of M Service Providers through a centrally secure, separate and discrete Message Broker. Unlike the Direct Architecture, any Service Consumer with *the proper authorization rights* can provision itself as a Service Provider, and receive Requests from and publish Events to, other Service Consumers with the appropriate authorization rights. **Details**: All of the functionality provided by the SIF 2.x Zone has been maintained in the SIF 3 Brokered Architecture, and in many cases has been extended. The formerly monolithic SIF 2.x ZIS operations have also been "repackaged" into more modular SIF 3 Infrastructure Service interfaces. The "Message Broker" functionality requirements of a SIF 3 Brokered Architecture can be implemented (among other alternatives) by SIF "business logic" layered on top of an Enterprise Service Bus (ESB), by internally coupled middleware components or by an upgraded SIF 2.x Zone Integration Server (ZIS). The Brokered Architecture offers a superset (rather than replaces) the functionality of the Direct Architecture. As a result, any Consumer interoperating successfully in a Direct Architecture can be redeployed into a Brokered Architecture without reprogramming.

Service Provider Types	Encapsulations of Data and Process
Object	While the SIF 3 infrastructure is independent of the Data Model defining the payloads it carries, all Service Providers must support the following general Object Service framework. An Object Service is the "authoritative source" for all data elements contained in all data objects of a specific type, and services some or all of the following requests: ● Query ● Create ● Update ● Delete Depending upon the object type, the corresponding Data Object Service may: • Publish an Event whenever an object is Created • Publish an Event whenever an object is Deleted • Publish an Event whenever certain (or any) elements in an Object are updated • Restrict the range of possible Queries Whenever an Object Service receives a Request for an operation it does not support, it **must** return an error.
Functional	A Functional Service encapsulates stateful process behavior as well as the data exchanged between applications implementing that process. It does this by supporting all four methods of a Data Object Service Provider interface, but applies them to a Jobs Phases rather than Data Objects. When a Consumer issues a "*create*" Request to a Functional Service, it results in the creation of a new executing instance of the Service (a "Job") rather than a new instance of a data object. From a conceptual point of view, each Job instance contains a set of named "phases", identical to every other Job created by that Function Service. These discrete phases define and encapsulate the sub actions, which need to be done, but they do not explicitly determine the ordering (since the phases defining a Function may be executed in different order, depending upon the implementation and the needs of the site where the Functional Service is deployed). Once created, the Job instance can be queried to find out where in the process it is (what is happening, what is the current status of each completed phase) and the Job

	may issue Events. Each Job Phase is represented by: • A Phase name • A status (N*otStarted, InProgress, Completed, Failed)* • A defined Object Service corresponding to that Phase (which supports some or all of the set of service operations) The creator of the Job can therefore: • Monitor the status of the Job (through querying the Job instance or by receiving Job level Events) • Interact with the Job at any phase by issuing Query, Create, or Delete requests. • Impact the Job indirectly through its defined Phases. • Receive Events from the Job Example: *StudentLocator, EndOfYearRollover*[4]
Object Service Subtypes	**Sub classifications of Object Services**
Infrastructure Service	The following Infrastructure Services[5] when taken together define the Environments Provider Interface. • **Environments** (defines and controls the level of Consumer contact with all other Services) • **Provision Requests** (used to request authorization to invoke additional Service methods) • **Connectors** (accepts all Requests, Responses and Events and routes them to their intended destination(s)) • **Queues** (collects incoming asynchronous Responses and Events, and guarantees their delivery, in FIFO order) • **Subscriptions** (Created to allow a Consumer to subscribe to Events from a specified Service Provider, which will be deposited into a specified Queue).

[4] *Explicit and more detailed examples of such Functional services will be provided at a later date.*

[5] *For further details on individual Infrastructure Services, please refer to the Infrastructure Services document*

	Separate infrastructure components may implement one or more of these Service Interfaces or they may be implemented by a single unified Environments Provider Adapter or Broker (the SIF 3 equivalent of the SIF 2.x ZIS). Unlike both the Utility and Application Services, which use the Connector URL for all Requests, each Infrastructure Service has an Environment-provided URL, which Consumers must use to invoke its operations directly.
Utility Service	Each of the following Utility Services[6] in SIF 3 conforms to the Data Object Service Interface, where the service being supported relates to the infrastructure and is independent of any locale-specific Data Model. • **Alerts** (Problem and exception reporting) • **Zones** (Registry of potentially reachable Zones) • **Providers** (Registry of available Service Providers) • **XQuery Templates** (Registry of "safe" XQuery scripts) • **External Code Lists** (Provider of codes from normative external standards) • **Namespaces** (XML Namespace Registry) Some Utility Services may be mandatory in both Direct and Brokered Architectures, others may be mandatory only in Brokered Architectures and some may be optional everywhere.
Application Service	Every Application Service supports either the Object, or Functional Service Provider Interface, typically by utilizing a Provider Service Adapter. Their specific payloads and actions are defined in the Data Model documentation associated with each locale-specific SIF data model release and any profile documents. In a Direct Architecture, Application Services are "pre-registered" and are components closely coupled to (or a direct part of) the implementation of the Environments Provider Interface. It is not possible for a Service Consumer to successfully provision itself as a Service Provider in such an Architecture. In a Brokered Architecture, Application Services are separate and distinct components which must first register and provision themselves as Consumers before they are allowed to provision themselves as Service Providers. The associated logic to implement the Service Provider Interface is typically contained in

[6] *For further details on individual Utility Services, please refer to the Utility Services specification*

	a Service Provider Adapter, which may or may not be tightly coupled to the underlying Application, which is the "owner" of the data or function.
Service Scoping	**The "destination" elements in each Service Request that allow, "content based routing" to determine the Service Provider that ultimately receive it.**
Environment	As indicated above, the Environment provides the totality of all Services that a Consumer can interact with, and includes the operational access rights that will govern those interactions. It can contain two or more Zones (one dedicated to Utility Services, and at least one dedicated to Application Services)
Zone	A Zone (similar to its meaning in SIF 2.x) is basically a collection of Application Services within the Consumer's Environment, pre-organized by the site Administrator to correspond to discrete components within the owning educational organization (such as a school or district) or similar criteria (ex: Special Ed students). Each Service "instance" accessible within the Consumer's Environment is scoped to a Zone, although a given Service Provider implementation may support the same Service Provider interface in several Zones. Unlike an Object or Functional Service, every Utility Service is applicable to all SIF 3 Zones in an Environment, and wherever present, is accessible by any properly authorized Service Consumer. This is achieved by assigning all Utility Services to a preset unique Zone (*infrastructure-utilities*) and giving the Consumer access rights to the appropriate operations of each of these Services. Each Service Consumer is assigned a "*default*" Zone at Registration time, which is used whenever a specific Zone is not otherwise included in one of its Application Service Requests. If any Consumer Request does not have a matching Service Provider registered within the specified Zone, it **must** fail. The *Zones* Utility Service provides a Registry of all available Zones within the Consumer's Environment.
Context	A Context is optional Data Model-specific metadata that may accompany a Consumer Request as a way of further scoping and restricting the possible Provider. For example a supplied Context might indicate that the Student Schedule Provider Service being requested in Zone XYZ is the one dealing with next term's data, rather than the current one. A Context consists of a unique (for a given type of Service in a given Zone) name, which is used by the Consumer when a request on that service is invoked. It also

	has an associated description and a set of parameter names and values, which may be defined by the Data Model the Service Provider conforms to, and is contained in the entry for that service in the Provider Registry. Contexts are not global … they apply only to a specific Service Provider instance assigned to a single Zone. A Zone can contain multiple Object Provider Services, each offering its data in a differently named context. **Uniqueness** Taken together, the Zone, Service Type and Context combine to identify a unique Service Provider instance (included in the Service Provider Registry) which the Consumer can make requests of. There can be only one Service of a given type with a given Context in any one Zone. Since the data model namespace is part of an Environment, messages bound to a service in different namespace **must** carry a different applicationKey. The Consumer can include at most a single Context in any given request[7]. If there is no matching Service Provider that supports the specified Context for the specified Service Provider type in the specified (or default) Zone, the Request **must** fail. The default Context is DEFAULT, and that is unique as well. If a Service Instance has no Context defined, requests to that Service instance must either not include a Context Name element or have that URL matrix parameter set to DEFAULT. The Zone and Service type (and the lack of a Context) provide all the information needed to determine the destination for that request.
Message Types	**Request / Response and Publish / Subscribe**
Request	Issued by a Service Consumer via the *Connector* Infrastructure Service, the Request invokes the corresponding operation on the selected Service Provider. This Provider **must** match the specified Service type, the Context name (if any) and either the explicit or defaulted Zone name the Consumer supplied. If no Service Provider qualifies, the Connector cannot deliver the Request, and an error will be returned to the Consumer. *Create, Update* and *Delete* requests may span multiple objects and on success will

[7] *Note that the Context is represented by contextId which is defined as an xs:token. An individual Data Model release might impose a Context hierarchy or other Context relationship within this token which could effectively bypass this "one Context per Request" restriction.*

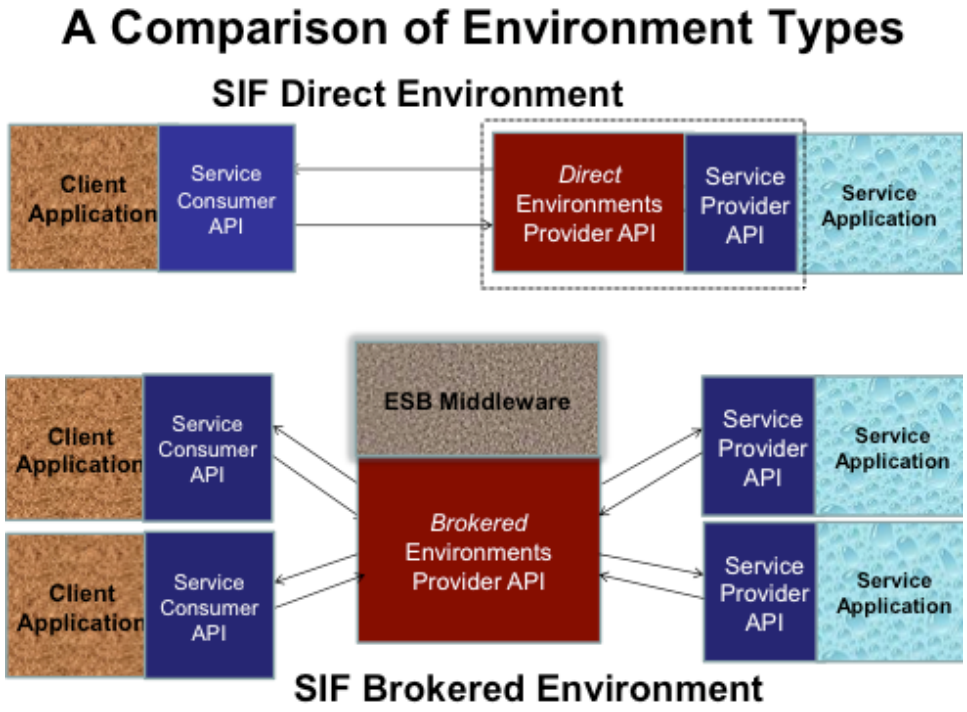
	result in a corresponding Event being issued by the Service Provider as a result of the changes they cause.
Event	Issued by a Service Provider via the Event Connector Infrastructure Service, either in response to a specific Consumer change request, and / or if the internal data in one or more of the Objects it is providing has changed. The Event type can be either *Create, Update* or *Delete,* and the Event message can report data changes of that type for one or more objects. Event messages are received by all Service Consumers who earlier successfully provisioned themselves as subscribers to data changes in that Object type, and represent an efficient way for Consumers to stay synchronized with the contents of the data maintained by a Service Provider.
Response	Issued by a Service Provider as an HTTP response, to a specific Consumer Request delivered earlier as an HTTP request. A Response to a multi-object *Create, Update* or *Delete* Request will convey (on success) a list of matching success / failure indicators. In the case of *Create*, the IDs of any newly created objects will also be returned. A Service Provider does not know whether the Response Mode selected by the Consumer (see below) was *Immediate* or *Delayed*.
Response Mode	**Synchronous or Asynchronous**
Immediate	The Response to a Request is provided synchronously in the immediate HTTP response, and the Requester thread for that connection "blocks" until the Response arrives. This was added in SIF 3. It matches the standard RESTful Client design pattern and must be supported in both Direct and Brokered Architectures.
Delayed	The Consumer issues the Request which is replied to with an "*Accept*" status code in the immediate HTTP response, which indicates "*Request is legal and can and will be delivered to the indicated Service Provider*". This frees a single-threaded Consumer to do other things. The Response issued by the Service Provider arrives asynchronously at a later time, in a manner identical with that of an incoming Event. It contains the Message ID of the original Request it completes.

Query Request Options	A variety of ways for a Consumer to request data
By ID	Only a single object is requested. The specific object desired is indicated by its *id*. If successful, every supported element in that object is returned. *By ID* requests can only be issued in *Immediate* mode.
Paged Interactive	A "Paged" Query is typically one in a series of Service Consumer queries for object data. Each such Query is idempotent … it contains both a "*starting page number*" and "*page size*" element". Together they define the start and end of the particular Page of results from the Query. It is common, but not required that each interactive Query issued by the Service Consumer will have the starting page number set one higher than the previous one. The response to each Interactive Query is immediate. The Service Consumer may stop issuing Interactive Queries at any time. Example: A teacher holding a tablet device runs a simple Service Consumer REST application that interactively queries the Assessment System for the first 30 Student scores, which are returned immediately. After they are displayed, the teacher may hit "next" whereupon a new interactive Query will be issued, and the next 30 Student scores will be immediately returned and displayed.
Service Paths	This is an alternative to Service Name, and where defined and available its use can greatly enhance the speed at which the Consumer obtains the desired result. For example to obtain all the Sections in which Student 1234 is currently enrolled, the single Query Request URL to do that would contain the string: *../students/1234/sections*
Paged Batch	A "batch" Query posted by the Service Consumer, indicates that the requested data from all objects satisfying the Query parameters is to be returned by the Service Provider as a series of delayed (asynchronous) Response messages, where each Response consists of a Page containing a defined number of Data Objects. The maximum number of objects per page is the lesser of any limits the Consumer might have requested or the Provider might have imposed. Example: A Student Contact System at installation time uses a single Batch Query to synchronize itself to the entire collection of Student names and phone numbers held by the Student Information System. In this case a single Batch Paged Query request will result in multiple paged

	asynchronous Query Response messages appearing in the selected Consumer Queue.
Named XQuery	A more complex query in which conditions are set on the selection of objects to be returned and / or only a subset of elements is specifically requested in the response. This is done by including the token of a pre-registered Named XQuery in the Request, optionally (depending on the XQuery) accompanying it with parameter values used when the script is retrieved and executed at the Service Provider. *Named XQuery* requests can be issued in both *Delayed* and *Immediate* Modes. XQuery is used to unambiguously communicate the behavior of the query and **may** have *nothing* to do with how the Service Provider executes the named query indicated.

2.2. Environments

There are two types of Environments a Consumer may register with.



3. Conventions, Dependencies & Metrics

This section provides the "global" conventions, normative references and generic design metrics reflected in this version of the SIF infrastructure. It serves as a precursor to further descriptions in following sections

3.1. XML Name Spaces

In the SIF 3 releases, common Infrastructure components are assigned to one namespace whereas locale-specific Data Model elements are contained in locale-specific namespaces. All SIF defined namespaces are versioned in a similar manner – the new version (even in a minor release) is a complete replacement for the older version rather than an incremental addition. Wildcard version numbers have been removed.

The version number of every XML namespace will be included in its name. For this release, the infrastructure XML namespace (of whatever platform) is:

http://www.sifassociation.org/infrastructure/3.2.1

Namespace only include the Major and Minor parts of the version number, excluding the Revision to allow for backwards compatibility within a minor version.

This namespace may include references to other namespaces including but not limited to:

http://www.w3.org/XML/1998/namespace

http://www.w3.org/2001/XMLSchema-instance

http://www.w3.org/2001/XMLSchema

3.2. Normative References (Standards, versions and options)

The following set of web standards, versions and options are used by the SIF 3.1 Infrastructure to exchange XML documents. The collection of these normative dependencies is referred to as the Normative Infrastructure Dependency Framework. All SIF 3 applications and middleware **must** support the appropriate parts of this framework.

Technology	Choice	Options / Qualifications
Data Format	XML 1.0	JSON 3 via Gossner Notation

Data Encoding	UTF-8	
Line Protocol	HTTP 1.1 (Mandatory)	Support for persistent (“keep-alive”) connections
Transport	REST	Common REST HTTP header element and URL parameter design patterns are used. Detailed examples of these can be found in the SIF 3 REST Developer Sandbox (SIF-RS)
Security	TLS 1.1 TLS 1.2	
XML Query Language	XQuery 1.0	
XML Document Structure Representation	XPath 2.0	
Payload Compression	gzip	Dynamically agreed to via content-encoding and accept-encoding elements in the HTTP Header

3.3. Infrastructure Protocol Layer (HTTPS)

The infrastructure depends upon the protocol layer to provide a reliable connection to move messages back and forth between Consumer and Provider. This layer is also responsible for providing protocol-level security by means of encryption and authentication, and may optionally be utilized to provide data compression, which can be an important factor when place a large volume of messages or data on the wire.

By delegating the authentication, compression, and encryption to the protocol layer, it makes the application interface to the transport simpler. For example, a Service Consumer that wishes to send a Request to a Service Provider first assembles the payload and then invokes the Create (Message) method on the Connector Infrastructure Service via the standard REST transport layer conventions. The chosen transport layer takes the message and utilizes its mapping to the protocol layer to transfer it to the Connector Service where it is taken from the transport layer and either processed or routed in turn to the specified Service Provider.

In moving through the path from the Service Consumer to the Service Provider, the protocol layer may have compressed and encrypted the payload and authenticated the Consumer but

this is transparent to the higher layers. At the application or adapter level, it is data in and data out.

3.3.1. HTTPS Guidance

In order to ensure that Service Consumers, Service Providers and the implementations supporting the Environments Provider interface can interoperate with each other regardless of vendor or platform, all SIF 3 communicating components must support the HTTPS protocol.

The SIF 3 REST mapping specifically defines HTTPS usage feature-by-feature, including REST-compatible values for the HTTP Status codes and header elements.

Where HTTPS options or conventions increase performance and scalability, they have been adopted. A summary of protocol-level functionality is shown in the table below.

HTTP Feature	Usage	Reason
Unsecured (HTTP)	Optional functionality for Brokered Architectures. A Service Consumer utilizing HTTPS can be assured that any Environments Provider implementation (whether Direct or Brokered) will interoperate with it. A Service Consumer utilizing HTTP cannot have this assurance.	HTTP has potential value during testing, and in production environments that are behind firewalls and made secure in other ways.
Error Codes	Utilized. REST detected faults are mapped directly to accepted HTTP error codes.	Provides error logic consistency
Persistent Connections	Support is required.	Establishing TLS connections are expensive. Persistent connections are therefore utilized to enable scalable solutions.
Compression	Supported	Helps scalability where bandwidth is an issue

		such as in cloud computing. Compression technology is limited to gzip, and gzip compressed message exchanges **should** be supported by all SIF 3 components.
Pipelining	Not supported	The SIF infrastructure strives to be stateless.
Multiple Connections	Supported	While this has similar state requirements to pipelining, the server can control the number of incoming connections supported.

All Environment Consumers and Providers must encode the message using UTF-8; and must be able to process UTF-8-encoded messages.

3.3.2. Infrastructure Protocol Layer (SIF HTTPS)

In order to ensure that Consumers, Providers and any middleware supporting the Environments Provider interface can interoperate with each other regardless of vendor or platform, all implementations **must** support the SIF HTTPS protocol.

When using HTTP 1.1 with SIF, [RFC 2616] can be used as a reference. The default behavior for HTTP 1.1 is to use persistent or "keep-alive" connections. When operating in this mode, the Consumer may send additional HTTP messages and receive the HTTP responses using the same connection. Consumers **must** use persistent connections.

SSL/TLS security is assumed to be supported by the servers involved and minimum levels set in and enforced by the Environments Provider. The minimum key length of the SSL Encryption certificate that can be used is 2048 bits. Additional details of such support are no longer conveyed within the SIF standard.[8]

3.3.3. HTTP Codes

Any of a number of 2XX and 3XX status codes may be returned in HTTP Responses to indicate that the action requested by the Consumer (or in the case of publishing an Event, the Provider) contained in the HTTP Request was received, understood, accepted and processed “successfully”.

[8] *See the SIF 3 Product Standard for Requirements: http://cert.sifassociation.org/Shared_Documents/SIF_3_Product_Standard.pdf*

There are also a range of standard HTTP Error Codes (4XX and 5XX) which will be returned in case of Error. All these codes are defined and explained in Appendix C of the Infrastructure Services document.

3.4. UUIDs

The SIF 3 infrastructure leverages Universally Unique Identifiers (UUIDs), per [RFC 4122]. To avoid the possibility of ID collisions, SIF 3 systems generating UUIDs *when an IEEE 802 MAC address is available,* **should** use version 1 GUIDs (a "1" in character 13) which are unique in space as well as in time. *If an IEEE 802 MAC address is unavailable* or if the inclusion of that address in a GUID poses a compromising security risk, systems **must** use version 4 GUIDs (a 4 in character 13) which use a (pseudo-) random number-based algorithm.

All infrastructure object UUIDs **must** then conform to the following XML pattern:

[0-9a-f]{8}[-][0-9a-f]{4}[-][14][0-9a-f]{3}[-][0-9a-f]{4}[-][0-9a-f]{12}

The unique object identifiers of the data model defining the message payloads may also conform to this requirement. However since they are only referenced by the infrastructure in constructing single object URLs (ex: students/12345) data model object identifiers are required only to be valid XML tokens.

3.5. Message-level element snippets and examples

All message level elements are documented below in terms of names and formats. However they could be represented in one of several ways:

1. As an element in the body of an HTTP Request (SIF Request or SIF Event) or an HTTP Response (SIF Response or SIF Error).
2. As a unique field in the HTTP Header
3. As a segment in the URL an HTTP Request is being issued to (as the unique object identifier is represented)
4. As a Matrix Parameter in the URL an HTTP Request is being issued to
5. As a Query Parameter in the URL an HTTP Request is being issued to

Note that only in the first case would the format of the element actually be defined in XML/JSON when sent across the wire.

4. Basic Infrastructure Framework

This section describes, in order:

- The basic interfaces and components of a SIF Environment
- The message exchange patterns which define the ways in which these components can exchange data
- The common set of "non-payload" elements contained in all messages being exchanged
- The message interchange "orchestration" required to process a Consumer Request which changes Service Provider data
- The set of identifiers which "scope" the data in a given exchange
- The requirements imposed to make sure all data exchanges are secure.

4.1. Service Hierarchy

The SIF 3 Infrastructure defines a specific framework for use in constructing a scalable, secure networking solution for educational data exchange, although it can be utilized to transmit data specific to other domains. This framework encompasses a collection of *Service Providers* accessible to one or more *Service Consumers,* linked together by a *Environments Provider*, and is organized into the following hierarchy.

4.1.1. SIF Environment

The SIF 3 Environment is defined by the set of Service URLs returned to a Service Consumer in response to a successful Registration Request. These URLs allow creation of a "customized" Environment. For example, depending on the authentication provided by the Consumer; the URLs may connect the Consumer to services for either production or testing, each encompassing a totally different set of Service Provider implementations.

The set of Service Providers available to a Consumer is subdivided into one or more "Zones", and the implementation supplying the SIF Environment for one or more SIF Consumers is called an "Environments Provider".

There are two types of SIF Consumer Environment support topologies, which are indistinguishable to the Consumer.

- A *Direct Architecture* provides only the set of Services bundled with the Environments Provider implementation. There are no "independent" Service Providers but starting in SIF 3.1 how to behave like one is described.[9] Such a Direct Architecture often consists of a single "Zone". It is similar to a single service implementation supporting multiple Service APIs.
- A *Brokered Architecture* provides multiple Zones where each Zone offers every Consumer access to one or more Service Providers which are independent of the Environments Provider middleware. It is similar to a set of Service APIs each implemented by a separate component.

Each Environment has a globally unique URL path that should identify the Educational Organization deploying the SIF solution, and optionally extend to a sub-scope within that organization. An example of such a URL path in a State-wide deployment might be:

https://tidewater.virginia.edu/sif3/production/Norfolk/12345

This corresponds to a SIF 3 production environment for the Norfolk District, within the Tidewater Regional Area of the Virginia DOE.

Environments Provider implementations provide data security, service discovery, guaranteed message delivery, and publish / subscribe capabilities which support complex communications between applications that have no direct information about each other, and that may or may not be accessible at any given point in time.

The *Environments Provider* interface falls into one of the following "types":

4.1.1.1. Direct Architecture

In the Direct Architecture configuration an Application such as an SIS or LMS implements the Environments Provider Interface to make its data available to a Client application that supports the Service Consumer Interface. As noted, there is only one Consumer within any Direct Architecture, although the SIS or LMS could support multiple simultaneous Direct Architectures each provided to a different Consumer application.

A single Service Application in a Direct Zone provides all available Services (Infrastructure, Utility, and the specific Object and/or Functional ones), possibly front ended by an Adaptor that includes an *Environments Provider Interface*. There

[9] *For an example of this see: xPress - Roster*

is no middleware component, and all client/service communication is direct (2-Party). As a result, the Direct Architecture Interface cannot support the ability of a registered Consumer to provision itself as a Service Provider. The Direct Architecture implementation receives all Consumer Requests and publishes all Service Events.

A defined subset of the Direct Architecture Interface requires support for only that set of service functionality that would be provided by a typical RESTful Service (no Events, single object per Request and the omission of most Utility Service Interfaces). It is intended to be implemented by those service applications needing to provide SIF-compliant Data Object support for straightforward RESTful clients, such as a Dashboard application running on a mobile device.

4.1.1.2. Brokered Architecture

In the Brokered Architecture configuration, middleware (typically in the form of a Message Broker or Enterprise Service Bus (ESB)) is enhanced to support the Environments Provider Interface. This middleware ensures that all Requests, Response and Events are securely routed between multiple client applications, which support the Service Consumer Interface, and multiple Service applications that support the Service Provider Interface.

In all Environments, any non-supported service operation will return an immediate, "*Requested Operation is Unsupported*" response.

4.1.2. SIF Zone

The Zone in which the Service is to be found always qualifies every Consumer request for any Provider Service. The size of a Zone is flexible and could encompass the educational applications in a single building, a school, a small group of schools, a district or a region. A SIF solution consists of one or more Zones deployed and configured to meet educational data sharing and reporting needs.

The presence of multiple Zones allows two or more Service Consumers to each register as default Zone providers of the same object type within a single Consumer Environment. This enables SIF solution administrators and integrators to better define how systems that publish similar objects cooperate within the same organization (e.g. Student Information Systems and Special Education Systems), by optionally defining and "clustering" each Service Provider and its set of associated Service Consumers into a separate Zone in which they can more closely interact. A typical example would be

applications in a district level SIF Environment that only need to share data within the same specific school.

Each Zone has a unique (within the Solution) identifier that corresponds to its scope in the educational organization. Examples of possible Zone identifiers within the Norfolk District integration above include:

- *RamseySchool*
- *RamseySchoolTesting*
- *SpecialEducation*
- *Districtwide*

Scoping might be by individual school, by whether the SIF Environment was test or production (which can be decided at the Solution level as well), by whether the student information came from an SIS or Special Education system, or by a combination of all three. The Zone identifiers are chosen by the administrator and can follow any convention that best meets the needs of the deploying organizations.

Every Consumer is provisioned with a default Zone, which will be used to scope Service Requests if no *zoneId* parameter is specified in the Connector URL when the request is issued.

4.1.3. SIF Context

The Zone is the primary means of partitioning educational data, applications, and policies. A SIF Context offers the ability to further partition the data within a Zone and reflect different perspectives of the data based on end user and administrator needs and application abilities.

Contexts are not global, they apply only to a single Object or Function Service type, and are specific to the Data Model of the payload being carried. A Zone can contain multiple Service Providers that support the same type of objects, as long as they have registered to do so for different contexts.

For example a supplied context might indicate that the Student Schedule being requested is for the next term rather than the current one.

If the Consumer knows no Context, or if none are defined for the Object Service type, the *contextId* parameter in the Connector URL when the request is issued is set to **DEFAULT.** Only one Provider of a given object type in a given Zone may supply objects with this Context.

Any given Service Provider in a Brokered Architecture can provision itself to support multiple contexts in one or more Zones. If there is no matching Service Provider that supports a Request qualified by Zone and Context, the Request **must** fail.

4.2. Message Exchange Patterns (MEPs)

Service Consumers and Service Providers exchange data within an Environment via three message types: Request, Response and Event

These are combined to support two message exchange patterns:

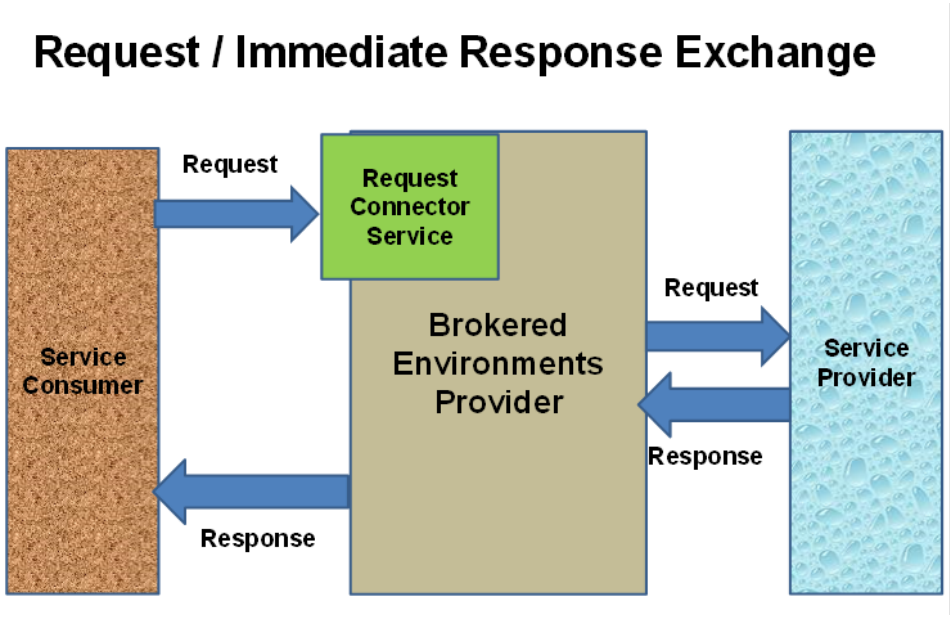
- Request / Response
- Event Publish / Subscribe.

Each will be described in terms of a Brokered Architecture. In Direct Architectures, all Service Provider and the Environments Provider implementations are identical, and any exchanges between these components shown in the diagrams and descriptions below are internal.

4.2.1. Request / Response

A Service Consumer invokes an operation (makes a Service Provider Request) via an HTTP Request to the Request Connector Service (part of the Environments Provider Interface). The Request is evaluated, and the specified Service type (ex: *Student*), Zone name (specified or Default) and selected Context are used to determine the correct registered Service Provider to receive it. If no Service Provider corresponds to these service-scoping parameters, the Request **must** be rejected.

Otherwise the Consumer Request is routed to the assigned URL of the selected Service Provider, and the response is immediately returned in the HTTP Response, as shown in the figure below.



4.2.1.1. Single or Multi-object Requests

There are two forms of the Create, Update and Delete requests, which are generally supported by all Service Providers unless specifically noted:

- **Single:** Change is being requested for only one object.
- **Multiple:** Change is being requested for multiple objects.

The Response to a multiple *Create, Update* or *Delete* Request will return the results of the operation for each internal object indicated in the Request. These results may not match the object ordering in the Request, but they will include both the ID of the changed object (allowing sub-request / sub-response correlation by the Consumer) and a "*success / failed*" status, where "success" indicates that:

- The suggested data change was completely accepted[10]
- The Service Provider altered its object data accordingly
- The requested change was or will be reflected in an Event issued by the Service Provider.

[10] *"Completely accepted" means all requested changes to all elements the Provider supports were successfully made. If there was a requested change to an optional element the Provider does not support, this is not reported as an error. If needed, the Consumer can determine this happened by examining the corresponding Event the change generated.*

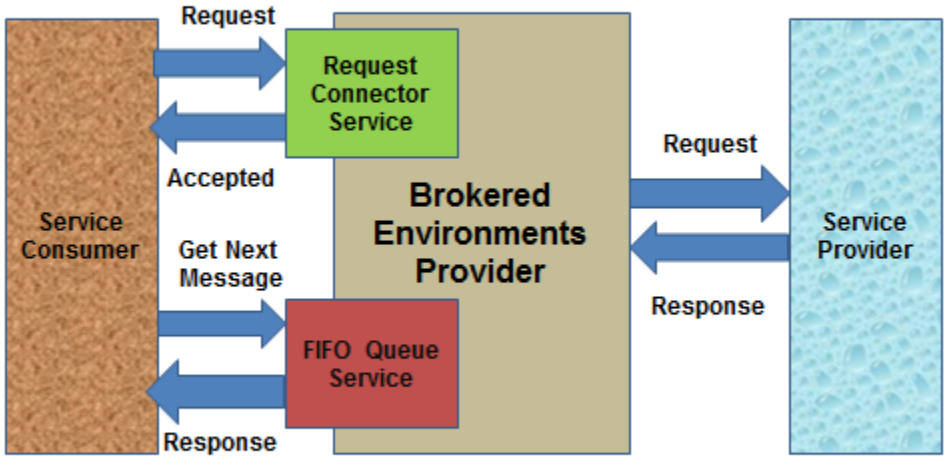
4.2.1.2. Immediate or Delayed Response

Any Consumer Request can indicate that the Response is to be either "Immediate" or "Delayed". In the Immediate case (illustrated above), the Request is issued as an HTTP Request, and the Provider Response is returned synchronously in the HTTP Response. If the Environments Provider determines it cannot obtain the immediate response information before a Consumer HTTP Request is likely to time out, it **should** immediately return an Error Response with HTTP Code 503 (indicating the immediate Request has been rejected and should be reissued as a delayed Request).

In the Delayed case, a FIFO Queue **must** be specified to receive the asynchronous Response from the Service Provider. The immediate synchronous HTTP Response is from the Environments Provider and indicates only whether the Request has been "accepted" and is being routed.

After the Environments Provider inserts the Service Response at the back of the FIFO Queue, the Consumer must retrieve it from the Queue via an explicit request for the "next" asynchronous message[11].

Request / Delayed Response Exchange



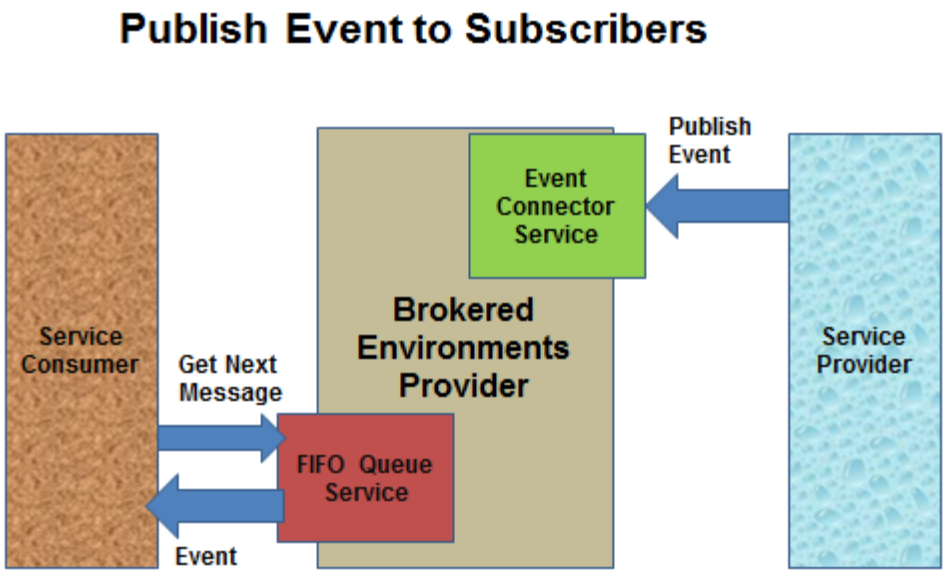
[11] *Please refer to the Queues Infrastructure Service section in the Infrastructure Services document.*

4.2.2. Event Publish / Subscribe

Service Consumers may subscribe to one or more Event types (*create, update* and *delete*) issued by one or more Service Providers during its initial "provisioning."

Service Providers (whether Utility, Object or Functional) post a change Event message whenever their internal data is created, updated or deleted. These Events are then routed to all subscribing Consumers, allowing them to synchronize exactly with the internal data of the Provider[12]. Any given change Event message can report data changes of that type for one or more objects (allowing one multi-object change Request to generate one multi-object Event).

In effect, a Service Consumer subscribes to a given Service once, and receives all subsequent Events as they occur. Every Event published by a Service Provider goes to every subscribed Consumer. The *Event Connector* Infrastructure Service owns keeping track of the active subscribers, and multiplexing each published Event to all of them. This frees the Service Provider from having to maintain the subscriber list, or of even being aware whether there are any active subscribers at all.



[12] *SIF 3.0 restricts the issuer of an Event for any Service to be the Service Provider. This is in contrast to SIF 2.x where non-providers of a Service could, if properly provisioned, publish Events for that Service.*

4.2.3. Subscriber Error Handling Logic

The following rules describe how a subscriber should process incoming Events when they are not consistent with its current internal state.

1. An unexpected Create Event arrives with the ID of an object the subscriber thought already existed. The subscriber **should** issue an Alert (whether or not it kept or discarded the Event data).
2. A Delete Event arrives with an ID for an object that the subscriber didn't know existed. The subscriber **should** issue an Alert and ignore the Event.
3. An Update Event arrives with an ID of an object that the subscriber didn't know existed. The subscriber **should** issue an Alert and **may** do a Query to get the current contents unless the replacement header is set to FULL, then the change event **may** be handled like an add event.
4. An Event of object type A arrives containing an ID which indicates the existence of an object of type B that the subscriber should have known about, but didn't. The subscriber **should** issue an Alert and do a Query on type B with the supplied ID to get the current contents of the indicated object.

4.3. Message Parameters

Enabling exchanges of Consumer issued Requests and Provider issued Responses and Events in a secure and robust manner, over a SIF-conformant REST transport layer is the primary function of the SIF Infrastructure. Every message exchanged has the following elements, provided by the sender that specifies the source of the message and the security, destination and context information. In REST, these element values may be carried as:

- XML/JSON elements in a message payload.
- Fields in an HTTP header (case-insensitive as per HTTP specification)
- Matrix parameters in a Request URL (located after the last URL path segment only and case-sensitive as per HTTP specification)
- Query parameters in a Request URL (after the "?" and case-sensitive as per HTTP specification)

4.3.1. Design Paradigm

In defining the message headers below, the principle of "*what you send is what is received*" was used. Every Service Provider in a Brokered Architecture has had to previously register as a Service Consumer. For consistency (and except for the *authorization* value which for reasons of security changes en route):

- Every HTTP header element and URL parameter inserted by the Consumer into a Request is seen in unchanged form by the Provider, which receives that Request, except for those elements directly involved with routing its response (ex: Queue ID).
- Except where explicitly noted, every HTTP header element inserted by the Provider into a Response is seen in unchanged form by the Consumer, which receives that Response, whether immediately or delayed (in response to a "Get Next Message" request sent to its FIFO Message Queue).
- Every HTTP header element and URL parameter placed by the Provider into an Event is seen in unchanged form by the subscribing Consumer, which receives that Event.

As a result, the responsibilities of the Broker in a Brokered Architecture are concentrated around secure message routing rather than data transformation. In addition, the numbers of steps necessary to convert a SIF-conformant application between *Direct Architectures Provider* and *Service Provider in a Brokered Architecture* are minimized.

Each of the three message type headers (Request, Response, Event) will be considered in the table below.

4.3.2. Parameter Details Summary

The Request, Response, and Event columns use the standard SIF Characteristics.

The Conveyed column using the following abbreviations:

H: HTTP Header

Q: URL Query Parameter

M: URL Matrix Parameter

P: URL Path

When more than one conveyance is utilized or a conditional is indicated, see the explanation for details of its use.

HTTP Header Field Name	Request	Response	Event	Conveyed	Explanation
Accept	O			HP	Used to indicate when the format that is expected in the response (ex: application/json). If omitted, may also be indicated by including an extension in the URL's path (ex: ".json"). Otherwise results will be conveyed using the default, XML.
accept-encoding	C		C	H	Indicate what payload encoding is accepted in the response. Valid values are: identity (not compressed) or gzip (compressed).
access_token	MC			Q	The token used to authenticate the sender of the message, authorizing the requested action. Usually the token/hash value of the Authorization header. This query parameter is only *required* when the Authorization header is not set or another authentication standard is leveraged.
applicationKey	MC			HQ	If the Application Key is not contained in the Authorization header, then this header must convey this key together with the authentication. The consumer may choose to convey this value in either place, so providers **must** honor it in either place.
authenticatedUser	OC			H	Set to the users identification (depending on the authentication used) *when verified by the middleware*. The receiving Service Provider can trust this field by confirming the middleware's credentials.
authenticationMethod	MC			Q	The identifier for the authentication method used. **Note:** Placing basic access authentication information in a URL query parameter is highly unsecure and **should not** be used in any production systems. Unless otherwise specified the prefix from the Authorization header is used: SIF_HMACSHA256/Bearer/Basic
Authorization	MC		MC	H	It is used to authenticate the Consumer and is the basis for determining whether the Consumer has the necessary authorization to issue the Request or *publish* the Event. When conveyed in *URL Query Parameters*, access_token and authenticationMethod are used.
changesSinceMarker	OC	OC		HQ	Request: URL Query Parameter. Only required if a changes since request is performed. Response: HTTP Header. Only required if the request had the changesSinceMarker as a URL query parameter and no paging is used or paging is used but the first page is requested.

connectionId	MC			H	Identifies the established connection over which the next message in the Queue is being requested and delivered. It must be a unique unsigned integer ranging in value from 0 to one less than the current value of maxConcurrentConnections. **Must** be included *only* when the consumer utilizes multiple connections to the same queue.
content-encoding	C	C	C	H	Indicate the payload encoding. Valid values are: identity (not compressed) or gzip (compressed). See also: accept-encoding
content-Type	MC	M	M	HP	Tells the receiver how to parse the body of the message. Supported generally, however SIF data models are generally conveyed with types application/json or application/xml (default). **Must** be conveyed *whenever* a body is present. **May** be omitted in a request. In that case the mime type is either: • The mime type indicated on the URL (i.e. .json) • XML if not defined on the URL or the HTTP Header See also: Accept
contextId	O		O	HM	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. This is carried as a matrix URL parameter in Requests, but is conveyed as an HTTP Header field on Events See also: relativeServicePath
deleteMessageId	OC			M	The ID of the last message received and processed by the Queue Owner. Only used when making Query Requests to the Queue Service Instance when deleting a previously retrieved message from the queue.
environmentURI		OC		H	May be returned by the environment provider where the environment is pre-provisioned.
ETag	OC	O		H	Optionally returned by a Service Provider within a Query Response, equivalent to a "checksum" on all the objects of the type being queried, which are maintained by the Service Provider. If it is returned in a Response, the Consumer **may** include it the next time it issues any Query to that Service Provider.
eventAction			M	H	The specific type of Event being reported: CREATE/UPDATE/DELETE
fingerprint	MC		MC	H	Unique environment identifier that can be safely shared with others. In order to not compromise security it MUST NOT match the environment's refId, sessionToken, userToken, or applicationKey. Added by the Broker to all Requests before forwarding to the

					Service Provider. Added by functional service provider to events that are intended only for the job's owner.
generatorId	O		O	H	The optional identification token of the "generator" of this request or event (ex: the administrative clerk who entered in the data that was responsible for generating a Create request).
messageId	O	M	M	H	UUID that uniquely identifies the message that carries it.
messageType	O	M	M	H	One of: EVENT/REQUEST/RESPONSE/ERROR If not provided, it will default to REQUEST.
methodOverride	MC			H	HTTP PUT: 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. Valid values are DELETE or UPDATE. HTTP POST: Included in an HTTP POST message when it is conveying a QBE request because the HTTP GET is not allowed to have a payload. Valid values are GET (QBE) or POST (Create).
mustUseAdvisory	O			HQ	Informs the Service Provider that if the "suggested" RefId in the Request cannot be assigned to the new object, the Request should be rejected. Valid values are true and false.
navigationCount		O		H	The total number of objects in the set of results generated by the initial Paged Query that is associated with the returned navigationId.
navigationId	MC	O		H	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. This **should not** happen when queryIntention is set to NO-CACHING or ONE-OFF.
navigationLastPage		O		H	It is included as an aid for the Consumer in detecting when to stop issuing Paged Query Requests.
navigationPage	O	MC		HQ	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. The first page is indicated with the value 1 (i.e. navigationPage=1).
navigationPageSize	MC	MC		HQ	This is included in *every* Paged Query 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. When contained in the Response, it indicates the actual number of objects on the returned Page.
Order	O			Q	Orders the result set by one or more specified elements and directions.

queryIntention	O			H	If the Consumer intends to follow up with further Paged Queries after this one, this field **must** be included in the Paged Query Request. Valid values are: ALL/ONE-OFF/NO-CACHING ALL: The Consumer intends to come back for the remaining pages of data. It is expected that the provider would return a navigationId HTTP Header in this case. ONE-OFF: The Consumer intends to make only this query, however the results **may** come from a cached source. NO-CACHING: The Consumer needs the data returned as it currently exists in the provider's data store. It is a "hint" to the Service Provider that maintaining Consumer state (and supplying a navigationId) would be advantageous. When not provided the default value is ONE-OFF.
queueId	MC			H	Contains the identity of one of the Consumer's assigned Queues to which the *delayed* Response or Job Object Events from the Service Provider related to this request **must** be routed. See also: requestType, jobId
relativeServicePath		MC		H	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). URL Query parameters are included. The Environments Provider places it into all delayed Responses (and would therefore not be supplied by a Service Provider in a Brokered Architecture), as an aid to stateless Consumers. It is optional for immediate Responses.
Replacement			O	H	Set to FULL (current values of all object elements) or PARTIAL (only elements whose values have changed) If not set, it is defaulted to PARTIAL.
requestId	MC	MC		H	Only required for delayed Requests. 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	O			H	Indicates what the request is trying to do. Defaults: • POST: CREATE • PUT: UPDATE • DELETE: DELETE • GET: QUERY

					• HEAD: HEAD
requestType	O			H	One of IMMEDIATE or DELAYED. If not set, it defaults to IMMEDIATE.
responseAction		M		H	This must exactly match the requestAction value contained in the HTTP header of the Request being responded to. Valid values are: CREATE/UPDATE/DELETE/QUERY/HEAD
serviceName			M	H	The name of collection being conveyed in the event.
serviceType	O	O	O	H	One of: UTILITY/OBJECT/FUNCTIONAL/SERVICEPATH/XQUERYTEMPLATE. If not provided, it will default to OBJECT
sourceName	MC			H	The applicationKey is added by Brokered Architecture to all Requests before forwarding to the Service Provider. Used by the Service Provider in Brokered Architectures when issuing an Alert concerning an erroneous Request.
Timestamp	MC	M	M	HQ	Date / Time of Event creation (in ISO-8601 format also used as the basis of xs:dateTime) If not need for authentication, **may** be omitted in the request. If needed, *only* for requests this value **may** be provided as a URL query parameter instead of a header.
Vary		O		H	This HTTP Header can be set by the provider to indicate that it supports compression. It would only be set if the consumer or broker calls the provider with uncompressed payloads where the provide could deal with compression. In such a case the HTTP header vary would take the following value: Vary: Accept-Encoding
Where	O			Q	A restricted XPATH expression that qualifies which among the set of all objects supplied by the Provider will satisfy the query and be returned.
zoneId	MC		M	HM	Indicates the Zone the Request should be routed to. It is a token that **must** have a value which: • Is unique from any other Zone ID • Identifies an entry in the Zone Registry if that Registry Service is present. If not specified in the Consumer's Request, the Request Connector will insert the Consumer's "Default" Zone ID (assigned to the Consumer when it initially created its Environment). This is normally carried as a matrix URL parameter in Requests, but conveyed as an HTTP Header field on Events.
[name matches XQuery Template parameter]	MC			Q	Supplies value to a parameter of the XQuery template whose token is in the last URL segment of the Query Request.

Note: *There are a number of elements that can either be provided as HTTP Header or as URL query parameter (i.e. navigationPage, navigationPageSize etc). If a value is provided as HTTP header and as URL query parameter then the HTTP Header must take precedence over the URL query parameter.*

Note*: The Authorization Token HTTP Field value in the Request as originally issued by the Service Consumer in a Brokered Architecture will be replaced by the Request Connector with one based on the sessionKey of the recipient Service Provider before re-issuing the Request to that Provider.*

Effectively, the Service Provider receives a Request with an Authorization Token value identical to the one it would have used if it had issued the Request. This both maintains security for the Consumer, and serves to validate the delivered Request to the Provider.[13]

4.3.3. URL Matrix Parameters

Where URL matrix parameters are present, they are restricted to the rightmost segment of the URL located directly to the left of the "?" which marks the start of the Query parameters. This conforms to common usage and is supported by the majority of REST Developer toolsets.

Matrix parameters are primarily used in SIF 3 to convey Zone and Context qualifiers for the service destination of a Consumer Request. This serves several purposes.

- Zone and Context "qualify" the service receiving the request, rather than being part of the arguments passed to the service in the request. Matrix Parameters were specifically designed for such usage.
- When the Zone or Context changes, any previous request information caching by the recipient Service should be discarded. Specifying them as matrix parameters allows this to happen naturally (since they are effectively part of the Service URL).
- It avoids any confusion between the Zone and Context ID and the Query or XQuery Template parameters which are included after the "?"
- It provides clean removal of the previous message so the current one can be retrieved.

[13] *Further details about the authorization token and the authentication methods utilized in SIF 3 may be found in the description of the Environments Service in the Infrastructure Services document.*

Here is an example of a Query request URL containing both types of URL parameters:

../students;zoneId=DuncanHigh2014;contextId=current?where=[(name/nameOfRecord/familyName ="Smith")]

4.3.4. Notation Headers

Java Script Object Notation (JSON) is an alternate way to represent an object in a tree like structure. JSON is desirous for a variety of reasons including: good wire efficiency, strong programming support, and satisfactory human readability. Since SIF's data models are formally defined using XML Schemas, JSON conversion is handled through a set of generic patterns (Gossner Notation). This results in immediate support for any convertible object.

Patterns

XML	JSON
<e/>	"e": null
<e>text</e>	"e": "text"
<e name="value" />	"e":{"@name": "value"}
<e name="value">text</e>	"e": { "@name": "value", "#text": "text" }
<e> <a>text text </e>	"e": { "a": "text", "b": "text" }
<e> <a>text <a>text </e>	"e": { "a": ["text", "text"] }
<e> text <a>text </e>	"e": { "#text": "text", "a": "text" }

Considerations

While several considerations need to be made in order to ensure the XML object is readily convertible to JSON and back again[14], one is of significant impact for those developing SIF solutions. For privacy reasons, the extension points introduced with the data models, intended to run over the SIF 3 infrastructure, rely on one or more third party namespaces and schemas. However, when converting to JSON

[14] *http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html*

using the adopted patterns namespace information is lost. This means the adaptor receiving the JSON representation of the object may need to know of all utilized extension points. The receiver must choose how to handle these extensions. Known approaches include: employing a path mechanism[15] to pull only the data of interest and use the JSON object either without validation or with a custom validation mechanism, changing the namespace of the affected elements to what is expected ahead of validation, or even stripping the extended elements out completely.

For this release of the SIF Global Infrastructure all events continue to be assumed to be in XML format. In a future release we will allow for subscriptions to objects in JSON.

Conveying Notation

When receiving a message it is important to know how to parse it. Likewise when creating a response, it should be packaged so that the receiver can understand it. In order to handle the notation of the payload or convey the desired notation of the response there are a collection of methods. They are presented here from most authoritative to least.

Headers

In order to indicate the payload of this message the Content-Type header should be used. When indicating compatible notations for the response the Accept header should be used.

Accept	application/xml application/json [others]
Content-Type	application/xml application/json [others]

[15] *http://goessner.net/articles/JsonPath/*

Extensions

As an alternative to setting the above headers a consumer may instead employ a URL postfix to its requests. When doing so, request and response payloads, if any, must be in this format. This indicator is only checked in the absence of the Accept and Content-Type headers.

XML: ...students.xml

JSON: ...students.json

[others]

Default

In the absence of the above indicators, the software must assume XML is both being sent and is desired in any response.

Precedence

If the HTTP headers are set and the URL postfix is used then the HTTP headers will take precedence.

4.4. Request / Response / Event Message Exchange Choreography

The following steps occur in the processing of every Request issued to a Service Provider, including the optional generation of an Event reflecting one or more changes to the internal Provider data resulting from the object changes contained in the Request.

Process Table

Step	Process	Flow Control
1	The Consumer of a Service constructs the Request payload and provides the required routing elements described above. The Consumer sends the Request to the Connector Service via an HTTP "Request".	
2	The Request Connector determines the appropriate Service Provider. • If no Zone is specified, the default Zone for that	These are combined with the specified Provider Type and Provider Name. If no existing Service Provider can

	Consumer is inserted into the Request *zoneId* matrix parameter. If no Context is specified, a *contextId* matrix parameter value of DEFAULT is inserted into the Request URL.	be found which meets these specifications, go to step 3. If the Consumer does not have proper authorization to invoke this request on the selected Service Provider, go to step 3. Otherwise go to step 4.
3	The Request is rejected. The Requests Connector sets the error code to indicate the reason.	The Connector sends an Error HTTP Response back to the Consumer, and terminates this message thread.
4	The Request is valid.	If the *requestType* element indicated an "Immediate" Response, go to step 6 (leaving the HTTP Request still "opened").
5	A "Delayed" Response was asked for. The Connector sends an "Accept" HTTP Response back to the Consumer to indicate the Request can be routed and will be delivered to the Provider.	At this point, Request delivery is guaranteed.
6	The Requests Connector replaces the Consumer's Authorization Token with the value of the Authorization Token the selected Service Provider would have used if it had generated this request as a Consumer[16]. This both prevents the Consumer's authorization rights from being abrogated by the Provider, and proves to the Provider that the incoming request message can be trusted. The Connector reissues the Request to the URL supplied by the Service Provider.	

[16] Please refer to the initial Consumer Registration section (specifically the Environments Provider create request) in the Infrastructure Services Document for further details about how Consumers are authenticated.

7	The Service Provider receives, unmarshals and processes the Request, according to the value of the requestAction. The Zone, Context and Service name arguments inform a multi-object Service Provider of which object type the Request applies to.	If the Request does not result in a change to the Service Providers internal data, go to step 11.
8	The Request resulted in changes to one or more internally held Service Provider Objects. The Service Provider **may** initialize one or more Event messages, with the appropriate *zoneId, contextId, serviceName,* and *generatorId* values.	
9	For each remaining Data Object whose value was affected by this Request, the Service Provider adds the object changes to the Event messages.	If the next changed object would exceed the Page Size for Events of this object type, go to step 10. When all object changes are reflected in the Event message, go to step 10.
10	One or more Events are ready to be sent. The Service Provider completes the Event messages, by calculating or copying an *authorization* token value and "publishes" the Events in an HTTP Request to the Events Connector.	If more object changes remain be processed, go back to step 8
11	Any change Events resulting from the Request have been posted. The Request must now be responded to. Send the Response back as the HTTP Response equating the following Response header elements with their Request counterparts. * *requestId* (if specified) * *generatorId* (if specified) * responseAction match requestAction	The Service Provider processing is completed at this point. The Response and Event messages still need to be delivered.
12	The Requests Connector receives the HTTP Response.	If the original Consumer Request indicated the Response should be immediate, go to step 16

13	The Consumer requested a delayed Response. The Connector then adds the Response to the back of the FIFO Queue Instance which the Consumer associated with this Delayed Request.	The process sequence suspends until the message moves to the front of the FIFO Queue. When it does, go to step 14
14	The Response reaches at the front of the Consumer's FIFO Queue.	The process sequence is blocked until the Consumer issues a "Get Next Message" operation on the Queue. When it does, go to step 15.
15	The Queue Instance returns the delayed Service Provider Response as the HTTP Response to a "Get Next Message" Request issued by the Consumer to its FIFO Queue.	All Delayed Response processing has been completed. Go to step 17
16	The Response has arrived for an Immediate Request. Send the Response as the HTTP Response to the original HTTP Request, completing the connection.	All Immediate Response processing has been completed. Go to step 17
17	Request processing (immediate or delayed) has been completed.	If the Request did not result in a change to a Service Provider's data, no corresponding Event will be generated. All processing is complete. Otherwise go to step 18
18	The Service Provider has created an Event message corresponding to the changes caused by the Consumer Request, and has sent it to the Events Connector for "publishing".[17]	If there are no subscribers to this Event type, all processing is completed. Otherwise go to step 19

[17] *In this case, the changes to the internal data were caused by successfully processing a Consumer request, but the process is identical to handling changes initiated by the Service Provider itself.*

19	The Events Connector has received the Event published by the Service Provider in response to the original Consumer Request. The Connector must deliver the Event to the Queue of the next Subscribing Consumer.	If there are no further subscribing Consumers who need to get this Event, all processing is completed. Otherwise go to step 20
20	The next destination for the Event message has been determined. The Connector removes the Provider's Authorization Token.	
21	Add the Event to the back of the FIFO Queue that the Consumer associated with its subscription to Events of this type.	Event delivery is now guaranteed whether or not the Consumer is currently active. At this point the processing splits … the Events Connector goes back to step 19. The process sequence stops until the message moves to the front of the Queue. When it does, go to step 22.
22	The Queue Instance returns the Service Provider Event as the HTTP Response to the "Get Next Message" Request issued by the Consumer to the FIFO Queue.	All processing resulting from the issuance of the Consumer Request has been completed.

4.5. Error Handling

There are several points in the above process where the Service Consumer could detect what is (or what it perceives is) an error in an arriving message.

Error Detection Occasions	Possible Causes	Action
The Request Connector rejects a Request posted by the Consumer.	An invalid Destination Service or an Authorization violation (the Consumer was not properly provisioned to request this Service)	Return an immediate SIF Error Object. The Consumer **may**

		create an Alert[18].
The Service Provider rejects a Request from the Consumer.	The Request had an invalid object identifier (ex: A Query for an Object the Provider did not think existed) or the XML payload of the Request was not aligned with what the Service Provider understood to be the object schema.	Return an immediate or delayed SIF Error Object. The Service Provider **should** create an Alert.
An Event or Response is received which is deemed to be invalid by the Consumer	The Event had an invalid identifier (ex: an "update" Event for an object the Consumer did not think existed) or the XML payload of the Response was not aligned with what the Consumer understood to be the object schema.	The Consumer **should** create an Alert.

4.5.1. SIF Error Message

The Service Provider returns a SIF Error Message to a Consumer issuing an erroneous Request.

The actual format of the Error Object is shown below.

Element or @attribute	Char	Description	Type or Value
@id	M	The identity of the Error Object	Format: UUID
Code	M	Corresponds to the value contained in the HTTP Header Status field in which the Error Object is the payload.[19] Its presence allows the Error Object to be self-contained when/if it is persisted.	Format: xs:unsignedInt 400 (Bad Request) 404 (Object not found) See: 4.5.2

[18] *Please refer to the Alerts Service in the Utility Service document for further details.*

[19] *See the HTTP Error Code section below for the complete list of such codes.*

Scope	M	Attempted operation. Ex: “Modify Student”	Format: xs:string
Message	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."	Format: xs:string (xs:maxlength = 1024)
Description	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. Example: "The 'Students' table is opened in exclusive mode by user 'ADM1' (dbm.cpp, line 300)."	Format: xs:string

4.5.2. SIF HTTP Error Codes

An HTTP Error 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:

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 or an unsupported query or an unsupported order clause
401	Unauthorized	Illegal Consumer Authorization token accompanying the request
403	Forbidden	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 No Service Provider has been found to match the parameters (Zone, Context, Service name) in the Request.
405	Method not Allowed	Paged Query Request issued to Object URL rather than Object List URL. Create Request issued to Provider Registry Service in a Direct Zone.
409	State Conflict	An attempt has been made to create an existing object.

		Attempt to create an Environment with an applicationKey for which an environment does already exist (and multiple instances are not configured).
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.

4.6. Success Handling

This class of 2XX and 3XX status codes indicates the action requested by the issuing Service Consumer was received, understood, accepted and processed successfully.[20]

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.

[20] *A Service Provider issuing an Event to the Event Connector is treated as a Consumer issuing a Request.*

304	Not modified	The specific response when a Query asks for objects which have changed, and none have

5. Service Operations

This section provides a more detailed description of the various types of Services and the set of Requests, Responses and Events that each type supports.

There are four (4) types of Service Provider Requests which a SIF 3.1 Service Consumer can issue:

1. Query
2. Create
3. Update
4. Delete
5. Head

Many (but not all) Service Providers will publish Change Events when they detect one or more elements in the data they provide has changed, either as the direct result of a *Create*, *Update* or *Delete* Consumer Request, or due to some Service-internal mechanism (ex: browser-based input).

The *Create, Update* and *Delete* Requests can all contain multiple embedded object-specific sub-requests of the same type. For example a single Modify Request can result in an Object Service Provider changing data element values in multiple objects, and a single Delete Request can include the IDs of multiple objects that are to be deleted.

Change Events can also contain multiple embedded object-specific sub-events of the same type. So in the above case, when a single Consumer-issued Modify Request arrives which contains modifications for N objects, the Service Provider can process all of them, issue a single Response back to the Consumer, and then publish a single Event to its subscribers reflecting all modifications to the affected N objects.

The maximum number of object-specific sub-requests that may be “packaged” by a Consumer into a single Object Request invocation varies by object type and is determined by the Administrator. The same value also applies to the maximum number of objects of that type that may be packaged into an Event.

5.1. Service Types

The following table describes the full range of SIF 3 Service types provided to SIF 3 Consumers. They are distinguished in a Consumer Request by the value of the "serviceType" HTTP Header Field.

Service Type	Description
Data Object	A Data Object is a self-contained collection of XML/JSON data elements. Its format is standardized in an XML Schema that is part of the locale-specific Data Model rather than the SIF 3 Infrastructure. The Data Object Service is the "authoritative source" for all data elements contained in all data objects of a specific type.
Utility Object[21]	A Utility object is also a self-contained collection of XML/JSON data elements. Its format is standardized in an XML Schema that is defined by the SIF 3.0 infrastructure and is independent of any locale-specific Data Model. The Utility Service (and by implication the Environments Provider which often implements it) is the "authoritative source" for all data elements contained in all Utility Service objects of a specific type.
Named XQuery[22]	XQuery technology is used to standardize the way in which Query Responses can be defined to meet important Consumer requirements. When the token representing an Named XQuery is specified in a Query Request, the Response can do some or all of the following: • Contain a subset of expected object elements (ex: no Student Health or Discipline information) • Include calculated aggregates based on the data in multiple objects of the same type • Represent a combination of data elements contained in multiple objects of multiple types The format of the Named XQuery Response is specified in an XML schema such as object in the data model or one specifically defined to match that XQuery. In terms of validation however, all elements of the schema are generally considered optional. In addition, the inclusion of the Template itself in the Data Model binding standardizes how the defined elements of the Response must be produced.

[21] *For further details on individual Utility Services, please refer to the Utility Services document*

[22] *For further details on Named XQuerys, please refer to the following table and the appropriate subsection below.*

Service Path[23]	Service Paths are basically predefined URL segments that are used to optimize Consumer Queries in important use cases. For example, a Query made to a URL containing "sections/1234/students" will return all Students enrolled in Section 1234. Such Service Paths are typically defined as part of the "binding" of a Data Model to the SIF 3 Infrastructure, and are most commonly used to "bridge" object associations. The XML schema defining the formats of the objects contained in the Response payload is determined by the last segment of the Service Path ("students" in this case.).
Functional (Job Object)	A Functional (or Job Object) Service encapsulates stateful process behavior as well as the data exchanged between applications implementing that process. It does this by supporting some or all of the methods of a Data Object Service Provider interface but applying them to educational processes such as *StudentLocator* and *EndOfYearRollover* rather than Object data elements. When a Consumer issues a "Create" to a Functional Service, it results in the creation of a new executing instance of the Function (a "Job Object instance") rather than a new Data Object. From a conceptual point of view, each Job instance contains a set of named "phases", identical to every other Job created by that Function Service. These discrete phases define and encapsulate the sub-actions that need to be done, but they do not explicitly determine the ordering (since the phases defining a Function may be executed in different order, depending upon the implementation and the needs of the site where the Functional Service is deployed). Once created, the Job instance can be queried to find out where in the process it is (what is happening, what is the current status of each completed phase) and the Job may issue Events as its internal phases are completed. Each Job Phase is represented by: • A Phase name • A status (N*otStarted, InProgress, Completed, Failed)* • A defined Object Service corresponding to that Phase (which supports some or all of the set of service operations) The creator of the Job can therefore: • Monitor the status of the Job (through querying the Job instance or by receiving Job level Events)

[23] *For further details on Service Paths, please refer to the following table and the appropriate subsection below*

	• Indirectly impact the Job through creating *individual* phase states. • Receive Events (where supported) from the various Phases of the Job. • Where appropriate Delete the Job.
Infrastructure	Like Utility Services, all Infrastructure Services are independent of any locale-specific Data Model. Each supports a defined Service Interface, which when taken together, define the Environments Provider Interface. However unlike all other Service Providers, the Consumer has a separate URL (returned at Registration time) for each Infrastructure Service, which it **must** use to directly invoke Requests on that Infrastructure Service. Depending upon the Infrastructure Service, each interface supports some or all of the standard four service operations (*Query, Create, Update, Delete*), although no Infrastructure Service posts Events when its internal data changes.

The following table further distinguishes between these service types. It omits Infrastructure Services as they each have their own individual URLs (supplied in the Environment) and do not construct URLs relative to the Connector as shown in the examples below.[24]

All Service types shown below have entries in the Providers Registry Utility Service.[25]

Entry Type	**URL Example**	**Operations**	**Events**	**Dynamic Query *where* clause**
Data Object	students	Query, Create, Update, Delete and Head	YES	Normally YES unless specifically restricted in the Data Model Binding or a Profile.
Utility Object	zones	Defined in Utility Services document	Defined in utility services document	NO
Named XQuery	StudentSnapshot	Read Only	NO	NO
Service Path	sections/{}/students where "{}"	Read Only	NO	Normally NO unless specifically enabled in

[24] For further details on the entire set of Infrastructure Services, please refer to the Infrastructure Services document.

[25] For further details on the Service Providers Registry, please refer to the Utility Services document

	indicates the section to report Students for by ID			the Data Model Binding or a Profile.
Functional Service	studentRecordExchanges	Query, Create, and Delete	YES	NO

5.2. Requests

In terms of the URLs where these Requests are issued, assume an object type called *Student*, with a *Student* Service Provider located at *a URL* ending in *students* and an existing *Student* object with an ID of 12345.

Request	Issued to URL	Effect when successful
Query (non-paged)	../students	Returns data for all Students
Query by Object ID	../students/12345	Returns data for Student 12345
Query (paged)	../students	Returns next page of Student Objects
Query (paged)	../students/12345	Never successful. Returns Error with code 405 (*Method not Allowed)*
Create (single object)	../students/student[26]	Creates a Student and returns its ID
Create (multi-object)	../students	Creates multiple Student objects and returns their IDs
Update (single object)	../students/12345	Updates specified Student
Update (multi-object)	../students	Updates multiple Student objects
Delete (single object)	../students/12345	Deletes specified Student
Delete (multi-object)	../students	Deletes multiple Student objects

[26] *On some REST development platforms, this URL is inconvenient or simply not possible to access. In those cases, the requester should issue a multi-object create to URL "../students", which contains only a single object of type "student".*

5.3. Service Request Identifiers

The following infrastructure identifiers are utilized in support of messages of more than one type. Each has been described earlier.

Identifier	Description
environmentId	An administrator defined globally unique token that identifies the Environment granted to the issuer (whether Consumer or Provider) of this message.
fingerprint	A unique id for an environment that is safe to share, because no operations can be done on the related environment referencing it by the fingerprint. This is useful for tying messages to an environment, without inviting attacks.
zoneId	An Administrator defined name that uniquely identifies a Zone entry in the Zone Registry Service currently operating in the Consumer's Environment. Typically used to scope the Service Provider which receives a given Consumer Request.
contextId	An Administrator or Data Model defined name that uniquely identifies the Context in the Zone in which the message is being exchanged.
messageId	A transport specific ID that accompanies all Requests, Responses and Events. It must be convertible to an RFC 4122 compliant UUID, (minus the *urn:uuid* prefix) with the 32 hex digit SIF Message ID contained in the string "{8 digits}-{4 digits}-{4 digits}-{4 digits}-{12 digits}" with the 13th digit set to a 4 (random) or a 1 (MAC Address). **Note:** The messageId is useful in identifying a specific message, especially in support of error analysis.
[id]	An immutable identifier attached by a Service Provider, to every object it supports. This identifier is unique to the Provider, and opaque to the Consumer. Unless explicitly defined otherwise, when used to identify an Infrastructure or Utility Service object, the ID format is always a UUID. However external Data Models may define the format for their objects as a more general xs:token.
navigationId	Optionally returned in a Paged Query Response, the *navigatorId* 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.

authorization	Present in all Requests and Events, it is based upon the sessionToken assigned to the Consumer after successfully creating its Environment. It is used to authenticate the Consumer and verify that it has been authorized to issue the Request or Event.
generatorId	Optional in change Request and Events. When present it identifies the "generator" of the Request and is carried over into any change Event resulting from servicing that Request. It might take the form of the email address of the administrative clerk who entered in the data.[27]

5.4. Object-level Query

The ability for a Consumer to obtain object information by issuing Queries to identified Service Providers is one of the fundamental cornerstones of application interoperability.

If the provider is not able to understand or implement the query, an error response with code 400 is expected.

The following two HTTP Header fields are unique to all forms of Query Requests and Responses:

Name	Description	Example Value or Format
requestId	Whenever this field is present in the Query Request, it is returned in the Query Response. It is a Consumer specified "token" that **must** uniquely identify every delayed Request issued by the Consumer. It is used to correlate the delayed (asynchronous) Response with the original Request. It could be as simple as a monotonically increasing integer.	*17*
eTag	Optionally returned by a Service Provider within a Query	Format:

[27] *If the SIF Data Model determines the payload of the Request message, the value of any generatorId element may be the object ID of the responsible user.*

	Response. In SIF 3.0 usage such an eTag is equivalent to a "checksum" on all the objects of the type being queried, which are maintained by the Service Provider.[28] If it is returned in a Response, the Consumer **may** include it the next time it issues any Query to that Service Provider. If any data change occurred to any object of the type being queried since the eTag was returned to the Consumer, that Request is processed normally, *whether or not any of the altered objects would be returned in the Response to this Query.* If there were no object data changes since the eTag was created, the Service Provider returns an HTTP Response with a Code of 304 (*Success – no data modified*)	Opaque Token

The SIF 3 functionality described in this section covers "unqualified" object level retrieval, where the Query Responses return all elements from all objects associated with the Query URL.

Subsequent sections will show how this functionality may be extend to include "qualified" retrieval where the Query Responses return only selected elements from a subset of objects meeting specified "where true" conditionals.

5.4.1. Object-level Query Options

The following Query options are supported independently of whether the Query is immediate or delayed, and (for Batch or Immediate Paged) whether the Query is for the entire object collection or further qualified by an Named XQuery or dynamic Where Clause.

Object Query Retrieval Option	Provider Response if successful

[28] *In SIF 3.0 usage, the eTag does **not** serve as a watermark (i.e. it applies to ALL objects of that type supported by the Provider rather than any specific one), so a change in any object will affect the eTag returned to all Consumers. On a Query containing an eTag, the requirement to have an "if-none-match" HTTP Header element accompanying the eTag to detect object changes has been waived.*

By Id	A *Query by Id* is issued directly to the URL corresponding to a specific object *(ex:../students/12345* where "12345" is the unique Identifier of the student). The *Query by Id* Response contains all data for the object corresponding to the specified Id. If successful, every supported element in that object is returned.
Bulk	The *Bulk Query* is made to an object list URL (ex: ../students) with no query-related qualifiers.. The *Bulk Query* Response (whether immediate or delayed) will contain all data for all existing objects of the indicated type supported by the Service provider. If the size of the returned data exceeds the limits of what either the Service Provider or the Environment can support in a single message, an Error with code 413 (*Response too Large*) is returned.
Paged Interactive	A Paged Interactive Query is issued to the URL corresponding to the object list *(ex: ../students*). It always requires an immediate response. It is typically one in a series of successive Consumer queries for the object data. Each such Query Request is for a "page" (a bounded subgroup) of the set of all objects that meet the constraints imposed by the Query Example: A teacher holding a tablet device runs a simple Service Consumer REST application that interactively queries the Assessment System for the first 30 Student scores, which are returned immediately. After they are displayed, the teacher may hit "next" whereupon a new interactive Query will be issued, and the next 30 Student scores will be immediately returned and displayed. Pages will be defined in more detail in a subsequent subsection.
Paged Batch	Similar to the Paged Interactive Query, except here the Consumer issues a single Delayed Page Query (typically for all the objects), and the entire set of Query Response pages are "returned asynchronously to the specified Consumer Queue without any further Query being issued. The Consumer may get the individual Query Responses out of the Queue at its leisure. They are identical in form to the Paged Interactive Responses except that

	they all share the same Request ID (because they were all generated in response to a single Request).[29]

5.4.2. Query Response Pages

This subsection expands on exactly what a Query Page is, and the HTTP header fields that support it.

A "Page" is a bounded collection of objects of the same type from the same Service Provider, returned in response to a "Paged Query" (interactive or batch).

There may be many such pages, each containing a different set of objects, which taken together comprise the full set of objects which satisfy a given Query request. A given Paged Query may be either immediate or delayed and may or may not have associated query qualifiers.

Paged Queries are the primary way in which Consumers can "walk" through large collections of objects in a controlled fashion.

Paged Query Use Case

There are 10000 SIS Student records which a Student Contact System Consumer must acquire to initialize itself to a new installation.

The Consumer assumes the results will be too large to be reported back in a single Response, so it issues a series of Paged Queries, each with a Page Size of 50 (the maximum number of objects it can contain) and an increasing value for the Page number. The first Response contains objects 0-49, the second Response contains objects 50-99 etc.

After 20 such Queries, the Consumer reaches the end of the data for the Query results, and the set of exchanges is complete.

Alternatively the Consumer could have issued a single (delayed) Batch Query Request and the Environments Provider would then issue the 20 Queries, and asynchronously deliver the 20

[29] Support for Batch Queries is the responsibility of the Environments Provider rather than the individual Service Provider, and whether that support is present will be indicated in the created Environment. If supported, the Environments Provider will "break open" the single Paged Batch Query into a set of Page Queries which it will individually forward to the Service Provider until the end of the query results is reached. Each response returned will be inserted into the specified Consumer Queue.

Query Responses to the supplied Queue Instance, where the Consumer could then retrieve them one at a time in synchronous fashion.

From the point of view of the Service Provider, both cases are identical, in that a Paged Query Request is received and a Query Responses is immediately returned, and this happens 20 times.

In the case where a Consumer is issuing a long series of sequential interactive page Queries, one obvious Service Provider optimization is the ability to determine the 10,000 objects once, but then "keep them around" so they can be used to satisfy subsequent Paged Query Requests from that Consumer with very little additional overhead. The ability to keep the set of objects around for reuse is especially important where they satisfy one or more Query Constraints set either in a previous Where Clause or Named XQuery Paged Request (see subsequent sections), so that the object selection does not have to be re-executed with the arrival of each new Paged Query. In order for this strategy to be efficiently implemented:

- Each Paged Query Request after the first one **must** include a Provider-supplied "identifier" to allow it to be linked with the existing set of retrieved objects, without recreating the complete list of objects to be returned. This identifier is the *navigatorId* HTTP header field described below. For Paged Interactive Queries, this responsibility is placed upon the issuing Consumer. For Paged Batch Queries in a Brokered Architecture, this is the responsibility of the Environments Provider, which must take the original Consumer Query and issue a series of Paged Queries to the Service Provider.
- The Service Provider should not save the Consumer-specific state in the first place, unless it has reason to believe that the Consumer will continue to request new object pages using the returned *navigatorId*. The Consumer **should** inform the Service Provider that its state should be saved, by including the *queryIntention* HTTP header field in each Paged Query except the last. Whenever that field is present, a "time to live" indicator associated with the Consumer State **should** be reset. When it is not, the Service Provider can immediately remove the Consumer-related state associated with the *navigationId*.

The second advantage of connecting a Paged Query Request up with previously saved state for that Consumer is that a consistent view of the objects maintained by that Provider might be obtained. For example, if the XQuery script or XPath expression were re-executed with each Request, and a "lower numbered" object was added or deleted between any two Paged Query Requests, the entire object index would be skewed and

the same object would either be reported on two successive Responses, or not reported at all.

Use of the *navigatonId* allows the Service Provider to generate a consistent array of objects in response to the initial Paged Query, and supply all subsequent Paged Requests issued by that Consumer from there. The Consumer can determine if Objects are being added or deleted during the time it is requesting Pages, because these changes will be reported in Events issued by the Service Provider.

The complete set of Paged Query / Response specific HTTP header fields or URL Query parameter are provided in the table below.

Name	Where Found	Description	Value or Format
queryIntention	Requests	If the Consumer intends to follow up with further Paged Queries after this one, this field **must** be included in the Paged Query Request. It is a "hint" to the Service Provider that maintaining Consumer state (and supplying a navigationId) would be advantageous.	• ALL • ONE-OFF • NO-CACHING
navigationId	Response and then Requests	Optionally returned by a Service Provider within a Query Response. If it is, the Consumer **must** include it on all subsequent Paged Query Requests it issues which have the same Query constraints (service and (where present) query qualifiers). It is used to allow the Service Provider to reuse the Query results for an individual Consumer even in cases where multiple Consumers are changing the underlying data between Paged Query invocations.	Opaque Token
navigationPageSize	Request or URL Query Parameter and then	This is included in every Request, and indicates the number of Objects to be returned in the corresponding Response Page. If its value is zero, the Response will contain no objects, but such usage may be valuable if the Consumer wishes to ascertain	Integer

	Response	from the returned navigationCount: • The total number of objects which can be paged through, before beginning Object retrievals • The aggregate number of objects which met the constraints contained in the associated query qualifiers 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. When contained in the Response, it indicates the actual number of objects on the returned Page. This will be equal to the Page Size value in the Request unless this is the last page of results, in which case its value will be the number of remaining objects.	
navigationPage	Request or URL Query Parameter and then 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. Such a No Content Response is particularly important in those cases when the Provider does not support the *itemCount* field, as it allows the Consumer to determine when it has completely paged through the entire set of stored results. **Note:** As inferred by the algorithms in the navigationLastPage entry below, the first navigationPage is one (**not** zero).	Integer
navigationCount	Responses	The total number of objects in the set of results generated by the initial Paged Query that is associated with the returned *navigationId*. If omitted, the Service Provider is indicating that it cannot, or chooses not to calculate the total number of qualifying objects in advance. This might be the case for a Service Provider front-ending a distributed database, where performing this calculation would be an expensive operation.	Integer

navigationLastPage	Responses	This HTTP field only appears in Responses containing Item Count where the Page Size is non-zero. It may be directly derived from the following formula: • (Item Count / Page Size) where there is no remainder • ((Item Count / Page Size) + 1) in all other cases It is included as an aid for the Consumer in detecting when to stop issuing Paged Query Requests. Like with navigationCount, if determining the count is too expensive, this header **may** be omitted.	Integer

5.5. Service Paths

Although the SIF Infrastructure is independent of the Data Model defining the payloads it carries, Service Paths are provided to optimize data retrieval for those Data Models which utilize "Associative" objects. Assume the following three objects: Student, Section and StudentSectionAssociation, the last of which might contain the RefId of a Student, the RefId of a Section and the information related to that Student's association with that Section (such as the student's attendance records and final grade for that section).

A common Consumer use case might be to retrieve all students for a given section who's RefId is known. This might be done by:

- Querying the StudentSectionAssociations collection for all associations with the given section RefId (1 query)
- Examining each returned Association object to obtain its corresponding Student RefId.
- Querying the Students collection for the Student object corresponding to each RefId (1 query per student in section)

In the case where there were 40 students in that section, this would require a total of 41 Queries. In a similar manner, obtaining all the Students in a School might require hundreds of individual Query requests. With Service Paths, the Consumer issues only a single Query in either case.

5.5.1. Service Paths in Query URLs

When used, the Service Path replaces the Service Object Name in the URL of a Query Request, and the Consumer never accesses the intermediate associative object. In the example above, the Service Path in the Query URL would look like:

../sections/1234/students

The Query Response would contain the collection of Student objects corresponding to the set of students enrolled in the section whose RefId was 1234. This single Query then provides the same Student data that would otherwise have required 41 queries to obtain. The exact same approach could be used to satisfy other common use cases. For example:

../students/5678/sections returns all the Section objects in which Student 5678 is enrolled

../schools/9012/students returns all the Student objects attending School 9012

../schools/9012/sections returns all the Section objects offered at School 9012

The following restrictions apply to Service Paths:

- They may only be used in Query Requests. A Service Path **cannot** be used to create, update or delete data.
- They **do not** post Events and cannot be subscribed to. In the above example, when an existing Student is added to a Section, the only Event published will be a Create Event, issued by the StudentSectionAssociations Object Provider.
- They **may** "nest". The following URL is legal and may be defined in a particular Data Model binding to the SIF 3 infrastructure: *../schools/9012/sections/1234/students,* If successful, a query with this Service Path would return all Students for Section 1234 located in school 9012.

In all other respects, they are the equivalent of Service Names. This means a Service Path URL may be qualified by Context and Zone, and be combined with Paged Query HTTP Header fields and where clause URL Query parameters. What explicitly distinguishes a Service Path Query is that:

- The URL segments may include one or more UUIDs, none of which is the last segment
- The value of the "serviceType" HTTP Header Field is set to "SERVICEPATH"

5.5.2. Service Paths in the Provider Registry

If a Service Provider Registry is present, every supported Service Path within a given Environment **must** have an entry in that registry. This takes the form of

owningObjectName/{}/returnedObjectName

The "{}" is a placeholder for the RefId of the owning object that will be indicated in the actual Query URL as issued by the Consumer. To support the examples above would require the following Service Path entries in the Provider Registry:

sections/{}/students

students/{}/sections

schools/{}/students

schools/{}/sections

Any authorized Service Provider may provide a Service Path, although in general it will usually be the provider of the Association object.

5.6. XQuery

This and the following section describe ways to "qualify" the objects returned in the Response to any Query Request in terms of satisfying conditions present in the Query URL.

The responses to Object-level Query Requests include all the elements from all (or the next page) of the objects supported by the Service Provider located at the URL to which the Query Request is dispatched.

This section discusses how to use XQuery-based technology to impose constraints on the object data returned in a Query Response, by:

- Defining the "where" criteria which every returned object has to meet (ex: "*Student must be a senior*")
- Defining the "selection" of exactly which object elements within each qualifying object are to be returned (ex: "*only return Student name and address*")

Rather than imposing a SIF-specific syntax for specifying these constraints (as was done in SIF 2.x with the Query and Extended Query elements) SIF 3 has a light normative dependency on the XQuery 1.0 standard,[30] and uses that industry standard technology for this purpose.

5.6.1. Terminology

There are several related terms, which need to be defined at the outset.

Term	Meaning
XQuery	A query and functional programming language that is designed to query and transform collections of structured and unstructured data, usually in the form of XML.
XQuery Script	A script written in the XQuery "language", that when executed will apply the XQuery constructs and logic to (among other things) manipulate, transform and normalize an indicated collection of data.
Named XQuery	A registered XQuery script that is designed to incorporate externally set template parameter values in its execution.
XQuery Request	A (possibly Paged) SIF 3 Query Request that contains: the identification of a predefined (**static**) Named XQuery entry in the XQuery Template Registry Utility Service. This is optionally accompanied by the Named XQuery parameter values, included as additional Query parameters in the URL path.

5.6.2. Static XQuery Templates

What distinguishes a "static" XQuery Request is the presence of the following two fields:

Name	Where	Description	Value or Format
serviceType	HTTP Header Field	Indicates the type of Service being requested	XQUERYTEMPLATE

[30] *Please see http://www.w3.org/TR/xquery/ for the XQuery 1.0 specification.*

	of Query Request		
xqueryToken	URL Segment	The XQuery entry ID that identifies a Named XQuery stored in the Named XQuery Registry. The query invoked by this Named XQuery and the corresponding URL Query parameter values must: conform to the XQuery script limitations required of all entries in the Named XQuery Registry	xs:token The value must be unique within the integration, and usually corresponds to the name of a predefined XQuery in the Data Model binding or a Profile. Ex: *StudentSnapshot*

A Query Request containing such a Named XQuery token can be dispatched to any Object Service Provider that either:

a. Supports the retrieval, interpretation and execution of XQuery scripts (typically by leveraging the functionality of an XQuery interpretive package).

b. Has been pre-prepared to accept this specific token value, roll in the supplied template parameters and process and return the resultant data. This bypasses the requirement for the Provider Service to have a general XQuery processing capability.[31]

The format restrictions placed upon any Named XQuery that may be interpreted by SIF 3 Object Services Providers are defined in the documentation of the Named XQuery Registry Utility Service, and they may be further restricted by site-specific data security policies.

Issuing a Query Request based upon a static Named XQuery is done by including the name (its ID in the Named XQuery Registry) as a URL segment in the Request, optionally

[31] *In addition to defining Composite Object types, a Data Model can also satisfy the need to optimize the processing of commonly used object qualifiers (ex: return AllStudentsInSection) by explicitly documenting the XQuery Templates that apply those qualifiers as part of the release. This allows the developers of Service Provider software to more readily support XQuery Templates.*

(depending on the script) accompanying it with parameter values used when the script is retrieved and executed at the Service Provider.[32]

Example:

Here is the URL to execute the "*StudentsBySection*" XQuery Template which will return just the students in the specified section, where the ID of that Section is provided in a URL Query parameter:

../ StudentsBySectionId?sectionId=acb66d4b-0140-1000-0006-14109fdcaf83

Note that the equivalent result could have been obtained in this case by (if supported in the Environment) issuing the following Service Path Query:

../ sections/acb66d4b-0140-1000-0006-14109fdcaf83/students

Consumer Process Flow

It is possible for the Consumer to read the contents of the Named XQuery Registry to confirm that the query it wants to use has already been constructed, checked for possible security violations and placed in the Registry by the site administrators under an agreed upon recognized name, which is the assigned value of its xqueryToken.

However in those cases where a specific Named XQuery is particularly useful, its name and content **may** be standardized in the Data Model release, so that Consumer applications can be constructed which rely on it being there in every Environment into which they are deployed (making the read of the XQuery Template Registry unnecessary).

In any case, the Consumer is assumed to understand what a selected Named XQuery does, and how to specify the set of associated URL query parameters that convert it into the XQuery script which when executed by the selected Service Provider, will return (in one Response or a series of Response Pages) the desired set of objects with the desired set of elements.

Service Provider Process Flow

[32] *For a detailed explanation of the XQuery Template Registry Utility Service which support and expand these Template Identifiers, please refer to the Utility Services document.*

Any Service Provider which has registered to support static Named XQuerys must perform the following sequence of operations when it receives a Query Request containing an xqueryToken.

Step	Process	Flow Control
1	The Service Provider receives the Query Request and detects that there is a supplied xqueryToken	If the Service Provider does not have the Named XQuery pre-stored or implemented, and if it does not support interpreting and processing unknown Named XQuerys, reject the Consumer Query with a code of 404 (not found), completing the processing for this Query. If the Service Provider already has the Named XQuery corresponding to that Token value, go to step 3.
2	The Service Provider issues a *Query by ID* to the Named XQuery Registry Utility Service, giving the supplied Token value as the ID.	If an Error is returned, reject the Consumer Query with a code of 404 (not found). Processing for this request is complete.
3	The Named XQuery is known to the Service Provider and can be processed. Execute it as an XQuery Script by supplying the values of the appropriate URL parameters contained in the URL of the Query Request.	If parameter matching fails, or if the script itself exits with an error, reject the Consumer Query with a code of 400 (Bad Request). Processing for this request is complete.
4	The XQuery script or other implementation has successfully executed and an “array” of results (indices to objects, to partial objects, or of the data objects themselves) has resulted.	Save the XQuery Template contents for potential reuse.
5	The processing is over. What happens from this point is determined from the other aspects of the Query Request (Paged or not Paged, Immediate or Delayed).	Before being sent in a Response to the original Query, the collection of objects returned by the XQuery script must be “wrapped” in the corresponding “plural” element (ex: wrap individual “student” objects in the “students” element).

		In general, most of the elements defined in the XML schema for an XQuery Response are optional.

5.7. Dynamic Query

While static Named XQuerys are extremely powerful, and can potentially involve multiple object types, cross Zone boundaries, calculate new element aggregates and make content-based decisions they are “static” because:

- Site security policies may require pre-inspection of all XQuery Scripts to ensure data privacy, as it may not be obvious from what is being returned, whether sensitive data has been compromised (since among other things, a script can return a potentially restricted value under a new element name).
- Many Service Providers may not have a generic XQuery interpreter accessible, and as a result, may not be able to process an arriving Named XQuery unless its content was known in advance so that its support could be “hard coded” either when the Service was first installed at the site or perhaps even before the product shipped.

As a result, a need was seen for a relatively “simple” way for a Consumer to dynamically include some limited qualifiers on any given Query which it could reasonably expect would be interpreted successfully by all Service Providers.

This functionality is achieved by attaching an additional “where” Query Parameter onto the URL specifying the intended Service Provider to which the Query is to be delivered. The value of this parameter qualifies which of the objects controlled by the Service Provider should be returned. It cannot be used if the Service type is an XQuery Token.

A Dynamic Query designed to isolate and return all students who took the introductory course in Computer Science, might contain (depending on the Data Model) the following where clause:

?where=[(studentssections/section=”CIS 101”)]

In a more complex example, the URL Query arguments to the Student Service instructing it to return all students named “John Smith” would be:

```
../students?where=[(name/nameOfRecord/familyName="Smith")and(name/nameOfRecord/givenName="John")]
```

Note: Every condition in the Where Clause is assumed to start just past the root of the object being queried. The above example as a pure XPath would have to (but doesn't) start with:

students/student

All Dynamic Query Where Clause values **must** conform to the simplified standardized XPATH format restricted for data security reasons, the format of which is specified below.[33]

Component	Examples	Comments
Where Clause[34]	where=[*Condition* and *Condition* and *Condition* …] or where=[*Condition* or *Condition* or *Condition* …]	Multiple conditions linked together by either one or more "and" or one or more "or" Boolean operators. Each condition may be optionally enclosed in "()".
Condition	*Element Operator Value* name/nameOfRecord/familyName="Smith"	Object element and possible value, bounded by an operator.
Element	name/nameOfRecord/familyName	XPath to the element (or attribute) being evaluated, relative to the root of the object (ex: students/student)
Operator	= != < > <= >=	The "=" and "!=" equality operators **must** be supported for all elements that support being dynamically queried. Whether support for the other operators is required or optional depends upon the Data Model binding to the SIF 3

[33] *This expression format matches that of a singular form XQuery expression (see Utility Services Document section on the XQuery Template Registry) but mandates neither namespace definitions nor absolute element paths.*

[34] *For purposes of clarity, in this, as in all examples, the required URL encoding of the contents of the "where" clause is assumed rather than explicitly shown.*

		Infrastructure.
Value	“Smith” “true” 17	All non-numeric values **must** be surrounded with double quotes

As a final example, the equivalent to the “Query by ID” URL which returns the data for Student 1234:

../students/1234

is the dynamic query:

../students/?where=[@id=1234]

5.8. Result Set Order

There are many use cases where the ability to specify the order of a result set is a requirement. For example: a user wants to list all students of a school, ordered by last name. If the results could all be returned at once (on a single page), the consumer could easily order the students before being displayed to the end user. However, it is likely that there would be multiple pages of students. In this situation, it is necessary for the ordering to be performed by the provider of students.

A consumer may specify an “order” clause to any Object Query, Dynamic Query, or Service Path by adding a query parameter of the following form:

?order=[primarysortfield=direction;secondarysortfield=direction...]

For example, to order students by last name ascending, the query would be:

../students?order=[name/nameOfRecord/familyName=ascending]

To order students with “last name ascending” as the primary sort order and “first name descending” as the secondary sort order, you would execute:

../students?order=[name/nameOfRecord/familyName=ascending;name/nameOfRecord/givenName=descending]

All “order” sort fields must conform to the simplified standardized XPATH format defined in the previous section.

5.9. Query By Example (QBE)

Another very useful mechanism for a consumer to query data is based on the concept of “Query by Example” (QBE). The concept is based on providing the “Query Executor”, the provider, an example or template of the data to be returned. For example if we want all **female** students with the **legal family name** of “**Jones**” to be returned we would send the following payload to the object provider (we use the SIF AU 1.3 data model in the example):

```
<StudentPersonal>
  <PersonInfo>
    <Name Type="LGL">
      <FamilyName>Jones</FamilyName>
    </Name>
    <Demographics>
      <Sex>2</Sex>
    </Demographics>
  <PersonInfo>
</StudentPersonal>
```

The result of a QBE **must** always be a collection of the same object type as the “example.” In the above example a collection of StudentPersonal objects where the three conditions match. QBE is an alternative to the dynamic query. The Dynamic Query is based on xPath notation to provide the “where-clause” of the query. QBE in contrast provides a partial Data Model Object where the elements that are provided are being used in the “where-clause” on the object provider.

5.9.1. REST Call

Because QBE requires a payload (the query template) the QBE REST call is conveyed as a **HTTP POST**. The HTTP GET cannot be used since payloads are not supported for HTTP GET.[35] To distinguish between a Create Request which is also conveyed as a HTTP POST, a HTTP header called ‘**methodOverride**’ with the value ‘**GET**’ must be provided for QBE functionality. If such a HTTP header is not present or has the value of ‘POST’ then the REST call must be interpreted by the provider as a standard Create Request.

[35] http://stackoverflow.com/questions/978061/http-get-with-request-body

The REST call does support all other features of a standard GET (query), ServicePath and/or Dynamic query. Specifically it supports paging, query intentions in immediate and delayed mode. All standard response HTTP status codes as with a GET (query), ServicePath and/or Dynamic query are supported, specifically if a provider cannot support a QBE then the standard HTTP status code of 400 (Bad Request) **must** be returned by the provider.

Example of QBE Request:

```
POST /.../StudentPersonals
methodOverride: GET
...

<StudentPersonal>
  <PersonInfo>
    <Name Type="LGL">
      <FamilyName>Jones</FamilyName>
    </Name>
    <Demographics>
      <Sex>2</Sex>
    </Demographics>
  <PersonInfo>
</StudentPersonal>
```

5.9.2. QBE Payload & Query Functionality

The QBE HTTP POST call will require a payload. The payload is a **single** standard Data Model Object (i.e. not a collection of objects). The object provided is the query template meaning that all the elements that are provided as part of the object are to be used in the query condition. Each element in the payload forms a specific query condition. The conjunction between the conditions is '**AND**'. There is no 'or' or any other conjunction between conditions supported. This is the nature of the QBE concept. The comparison operator for each condition is a '**LIKE**' as known in SQL syntax. The only supported wildcard in the value of an element is the '**%**' which stands for any number of characters. The following assumptions and constraints apply to the LIKE comparator in QBE:

- A 'LIKE' shall be interpreted as an **EQUAL** if the Data Model Object element type is anything other than a string style type (i.e. dates, numbers etc.).
- For string type elements the 'LIKE' with wildcard has the following meaning:
    - 'ABC': No wildcards in the value is equivalent to an '**EQUAL'**.
    - '%ABC': Wildcard at the start of the value means '**ENDS IN'**. In this case anything that ENDS IN 'ABC'.
    - 'ABC%': Wildcard at the end of the value means '**STARTS WITH'**. In this case anything that STARTS WITH 'ABC'.

  - '%ABC%': Wildcard at the start and end of the value means '**CONTAINS**'. In this case anything that CONTAINS 'ABC'.
  - 'AB%C': Wildcard anywhere in the value means 'STARTS WITH' and 'ENDS IN'. In this case anything that STARTS WITH 'AB' followed by any number of characters and ENDS IN 'C'.
  - Etc.
- Case Sensitivity: No case sensitivity is implied or mandated. It is up to the implementation of the provider if case sensitivity is required/applied or not.

Examples:

All the examples below are Student QBEs and show only a partial payload:

Example 1: Get all students with family name of 'Jones'

```
<FamilyName>Jones</FamilyName>
```

Example 2: Get all students where the family name starts with 'J'

```
<FamilyName>J%</FamilyName>
```

Example 3: Get all students where the family name contains 'one'

```
<FamilyName>%one%</FamilyName>
```

Example 4: Get all students where the family name contains 'one' and the first name is 'Mike'

```
<StudentPersonal>
  <PersonInfo>
    <Name>
      <FirstName>Mike</FirstName>
      <FamilyName>%one%</FamilyName>
    </Name>
  <PersonInfo>
</StudentPersonal>
```

5.9.3. Provider Registry & ACLs

QBEs are considered as standard OBJECT queries and therefore no other service type in the provider registry is required. If the ACL for a particular object service lists the 'QUERY' right as 'APPROVED' then a consumer can issue queries using QBE. The provider may not support them at all or not all combinations. In such a case the provider must return a HTTP status of 400 (Bad Request).

5.10. "Changes Since" Functionality

Events are already supported since the initial release of SIF 3. SIF Events are most commonly supported in Brokered environments but unlikely to be available in DIRECT environments. To enable a consumer to request changes since a "given point" in a DIRECT environment the "Changes Since" functionality is being added to SIF 3.2. The "given point" can be any opaque marker that indicates a point since the last changes have been requested. That opaque marker can be a timestamp, a version number, an offset etc. It is entirely up to the provider to determine what this opaque marker is. The important thing is that it is the provider who will return changed data (payload) and the next valid opaque marker (HTTP Header) to the consumer in the response to a "changes since" request.

IMPORTANT: "Changes Since" functionality is only applicable for OBJECT services. In other words changes since is not supported for any other service type such as SERVICEPATH, FUNCTION, UTILITY etc.

5.10.1. REST Call (Consumer)

For a consumer to retrieve changes since a given point it simply issues the following request to the request connector of the standard Object provider (example for xStudents):

```
GET .../xStudents?changesSinceMarker=<opaque_marker_for_students>
```

Note the "**changesSinceMarker**" URL query parameter. It must be noted that there will be a different opaque marker for each object type. The consumer is expected to track them individually.

"Changes Since" requires this additional URL query parameter to indicate that the consumer requests a "Change Since" rather than a standard query. The actual REST call is conveyed as a **HTTP GET** and supports all other standard HTTP headers. Paging is allowed but **sorting and filtering ("where" clauses as for dynamic queries) is not allowed and must be ignored by the provider if given by the consumer**.

"changesSinceMarker" Management

The changesSinceMarker is given to the consumer in two ways one being the HTTP HEAD method the other being the response to the changes since request. Both of these are detailed below.

HTTP HEAD

The first method to retrieve the “changesSinceMarker” is through the HTTP HEAD method to the Object Provider. If the Object Provider supports the “ChangesSince” functionality for the given object then it is expected to return the “changesSinceMarker” as a HTTP Header in the response. If the provider does not return that HTTP header then the consumer must assume that “Changes Since” functionality is not supported for the given Object Type/Service. It is expected that the consumer calls this method when it first joins the SIF environment and before it performs and initial sync of all data.

Response to “Changes Since” request

Once a consumer has retrieved the “changesSinceMarker” through the HTTP HEAD call, has performed a full sync, it is assumed that the consumer will, at regular intervals, request “changes since” with the REST call listed earlier in this document. As part of the response to such a call the provider **must** return the new “changesSinceMarker” to the consumer as a HTTP Header field in the following two cases:

- If the consumer has ***not provided any paging info***, indicating that the response holds all changes for the given object type, then the provider must return the new “changesSinceMarker” as a HTTP Header field. The consumer will use that new value in the next “Changes Since” call.
- The consumer has ***provided paging info***. In this case the provider must return the new “changesSinceMarker” as a HTTP Header field in the response of the ***first page***. The consumer is expected to store that new value. It will still request the remaining pages with the original “changesSinceMarker”, though. Once all changes have been received it will use the new “changesSinceMarker” in the next “Changes Since” call.

  Note: In the case of ‘paging’ it is expected that the provider will also return the **navigationId** HTTP Header since it is highly likely that the consumer will come back and ask for more pages until there are no more. The consumer would provide that navigationId in each request as a HTTP Header or URL Query parameter.

5.10.2. Payload Interpretation

The response payload to a "changes since" request is always a collection of the same objects (i.e. xStudent). The payload can be empty if there are no more changed data (HTTP Status 204 - No Content). Because objects have no element that indicates the

change type there is a need to interpret the payload as listed below to determine the actual change type:

- If the refId/uuid of a particular object in the response payload does not exist in the consumer's data store it must be assumed to be NEW (i.e. created).
- If the refId/uuid of a particular object in the response payload does exist in the consumer's data store it must be assumed to be UPDATED (i.e. changed). It is important to note that the **full object is returned** not a partial object because there is no way to indicate if the update is partial or full.
- If a particular object in the response payload contains the **refId/uuid only** the consumer must assume that object to be DELETED.

5.10.3. Provider Registry & ACLs

"Changes Since" is considered as standard OBJECT queries and therefore no other service type in the provider registry is required. If the ACL for a particular object service lists the 'QUERY' right as 'APPROVED' then a consumer can issue "Changes Since" requests. If a provider of an object type service does not support the "Changes Since" functionality then it must return the standard HTTP status code 400 (Bad Request). **The "Changes Since" functionality is optional and not mandatory for a provider**. Note that a consumer can "query" the Object provider with the HTTP HEAD call to determine if an Object provider supports the "Changes Since" functionality before issuing "Changes Since" requests (see also 5.15).

5.11. Change Requests and Events Overview

One of the most important scalability features of this release is the ability to package changes to multiple objects within a single change Request and / or change Event. This functionality is especially significant during end of reporting periods when for example many thousands of small attendance record updates may need to be reported, because it allows the publisher to reduce the number of actual Events it issues by two orders of magnitude.

The following information applies to Create, Update and Delete Requests. It will be assumed that the issuing Consumer has supplied an *authorization* value that has been successfully authenticated by the Environments Provider, and it had been previously authorized to issue the Request to the indicated Service Provider. The distinction between immediate and delayed change Requests only determines when and where the results will be delivered, and is

covered in more detail in the description of the Queue Service in the Infrastructure Services document.

5.11.1. Multi-object Requests and Responses

Single object change requests conform completely to the standard REST design patterns, and always result in an immediate Response that also conforms to the standard REST design pattern (reporting success or failure with the appropriate HTTP code).

However when a multi-object formatted Request is issued (even if it contains only a single object) the multi-object form of the SIF Response message **must** be returned. Here each individual object in the Request will have its corresponding change status reported, although its position within the multi-object Response message may not be its position within the request (i.e. ordering is not guaranteed). As is the case with the single object Response, there is no object data (other than the RefId) contained within any multi-object change Response.

All delayed multi-object Responses **must** also contain the additional *relativeServicePath* HTTP header field, which replicates all information contained in the last segment of the Request URL, including the Service name defining the payload format, and any accompanying URL matrix parameters (which could include Context, Zone and XQuery Token). Any URL Query parameters are not included.[36]

In a Brokered Architecture the Environments Provider inserts the value of this field, if the Response is delayed. Its presence is transparent to the Service Provider actually generating the Response (which doesn't know whether the Response it is providing will be delivered back to the Consumer immediately or delayed).

Example:

The value of the relativeServicePath HTTP header field in a typical delayed Response might be:

students;contextId=archived;zoneId=WilsonSchool

[36] *This allows a "stateless" Consumer to reconstruct an outstanding delayed Request from the arriving Response.*

5.11.2. Multi-object Events

All Events are multi-object, even if generated by a single object change Request. All object data changes resulting from any change request **must** be published within one multi-object change Event generated from that Request, although here again, the object ordering within that Event may be different than it was in the original Request[37].

In the case of *create* and *update* Requests, success for an individual object operation implies that the values of all mandatory elements and every specified optional element that is supported by the Service Provider will be set to the values contained in the Request. Even if one or more unsupported optional elements do not have their values changed, the Request will be considered successful.

5.11.3. Partial Failures

These can occur in a "multi-object" change Request (ex: *Create, Update* or *Delete*) for which the Service Provider is processing each object change in turn. For example one or more or all of the object *ID* values in a *Delete* request might not match the *ID* of any existing object of that type.

The top level of the Response sent back by the Service Provider in this case will still indicate "*success*". The individual "status" assigned to each of the proposed changes in the Response will indicate whether or not the change is now reflected in the data associated with that specific Object. Failure of a multi-object request, like failure of a single object request, occurs only when a global error (ex: authorization violation, inability to unmarshal the payload) is detected.

5.11.4. Message Payloads and Data Objects

In general, all "infrastructure elements" such as Zone, Context, Message ID and Message Type are specified as either URL Query or Matrix Parameters or as HTTP Header fields.

As a result, for most forms of change Requests and Events, the HTTP payloads can be umarshaled directly into programming language objects since there are no "outer

[37] *The removal of the requirement to maintain the Request ordering of objects in the generated Response and Event allows Service Provider implementations to crack open a multi-object Request, assign a different thread to each object change, and dynamically build the Response and Event messages from the individual threads as they complete processing an object-specific change.*

wrappers" defined over the object contents, which, to maintain infrastructure independence from the data model, would of necessity have to be of type xs:any.

This greatly simplifies the requirements placed on standard XML to object converters.

5.12. Create

The Create Request is different than the other Change Requests because at the time it is issued, the *id* attribute, which is assigned by the Service Provider, is not yet known. The Consumer must provide a "suggested" or advisory value but the Service Provider is not required to accept that suggestion unless the *mustUseAdvisory* field is set in the HTTP Header of the Create Request. In that specific case, if the Service Provider cannot accept the suggested object ID, it must reject the entire Consumer Request (if for a single object Create) or the sub-request (if for a multiple object Create) with HTTP status 404 (not found). [38]

The Consumer's suggested ID will be returned in the "advisoryId" element of each sub-result in the Create Response (whether successful or not) so that the consumer can match the result to the original sub-request. This is required because the ordering of sub-results in a Response message need not match the ordering in the Request which generated that Response.

The format expected for the *object id* returned by an application service provider is a token (although a given Data Model can define its unique object identifiers however it choses, including UUID format). The formats for the *object id* in the various Utility Services are defined in the Utility Services document.

Message Type	Single Object Payload (on success)	Multi-Object Payload (on success)
Create Request	A "Student" object element, containing at a minimum, all mandatory elements, with an empty or "suggested" value for the "id" XML attribute.	A "Students" element, which consists of multiple "Student" object elements of the form described for the single object Request.
Create Response	An HTTP Response with a status of 201 (object created). The payload consists	An HTTP Response with a status of 200 and a payload consisting of a single "*createResponse*" element. This

[38] *The Consumer in this case is requesting that the object be created. The actual "creator" of the object is the Service Provider, and that is the component that assigns the globally unique object identifier (and issues the corresponding change Event and services subsequent queries, updates and deletes on the object when the identifier the Provider assigned to it is specified in the request).*

	solely of the new Student object.	includes, for each requested object successfully created, an internal status of 201 and the service-assigned refId. Any requested object that failed to be created is indicated by an error statusCode and a corresponding error payload Such a sample *createResponse* payload is shown below.
Create Event	N/A. Events are always reported with multi-object payloads, even if they contain only one object	A "Students" element containing the collection of all successfully created Student objects, each of which contains all the elements of the Student as known to the Provider (including the "id")

Example: Create Request

The following XML instance fragment illustrates the payload of a multi-object Student Create Request sent to the Service Provider URL ending in *students.* The ID "suggestions" are not globally unique, and will almost certainly be replaced by the Service Provider.

```
<students>
  <student id="1">
    … student data …
  </student>
  <student id="2">
    … student data …
  </student>
  <student id="3">
    … student data …
  </student>
</students>
```

Example: Create Response

The Response to a "create" Request of 3 objects, where the middle object was a duplicate of an existing one.

```
<createResponse>
  <creates>
    <create id="df789e1c-dfe7-4c18-8ef0-d907b81ea61e" advisoryId="3"
statusCode="201"/>
    <create advisoryId="1" statusCode="409">
```

```
        <error id="6f789e1c-dfe7-4c18-8ef0-d907b81ea61e">
          <code>409</code>
          <scope>StateConflict</scope>
          <message>Student already exists!</message>
        </error>
      </create>
      <create id="ff789e1c-dfe7-4c18-8ef0-d907b81ea61e" advisoryId="2"
statusCode="201"/>
    </creates>
</createResponse>
```

Note that the error is embedded in the second create, and the results do not correspond directly to the order of the objects in the original Request. Inclusion of the suggested object ID value in the advisoryId element allows the Consumer to correlate the error to the actual sub-request which failed.

The two successful create operations are reported back with their corresponding assigned id attributes.

5.13. Update

It is unnecessary for the Consumer issuing an update Request to have recently (or ever) issued a Query Request on the object being updated. This allows updating of a single element in an object by a Consumer who is aware of only that element. An example would be a Library System that is updating the Library Card number assigned to a Student.

As a result, the *update* Request is required to contain only those elements to which new values are being assigned, implying the payload of such a Request might contain objects missing one or more mandatory object elements. The Consumer is not constrained however and may send back additional information, up to and including the entire object.

In the corresponding *update* Event, the Service Provider is required to send back only the elements that actually changed. There are implementations that find this difficult to do however, and (whatever elements were actually modified) find it easier to send back the entire resulting object in the *update* Event.

This can be indicated through the following HTTP Header Field in the Update Event.

HTTP Header Element	Char	Description	Value
replacement	O	The Object data contained in an Update Event may reflect either:	One of:

		• Only the actual elements updated as a result of an Update Request (along with the object id) • The latest value of **all** object elements (in other words it completely duplicates the object "after" changes have been applied, rather than just including only the elements which were actually changed). [39]	"FULL", "PARTIAL" where "PARTIAL" is default and will be assumed if this field is not present

The actual payloads in update messages are described below.

Message Type	Single Object Payload (on success)	Multi-Object Payload (on success)
Update Request	A "Student" object element containing at least all elements that are being changed, plus the "id" identifying which Student the changes apply to. As noted, other elements may be included, up to the complete contents of the Student Object after the changes.	A "Students element, which consists of multiple, "Student" object elements, each subject to the restrictions of the Student element in the single object payload.
Update Response	An HTTP Response with a status of 204 (no content). There is no Payload.	An HTTP Response with a status of 200 (success) and a location corresponding to the URL of the first successfully modified object. It contains a payload consisting of a single "*updateResponse*" element of the form shown below.
Update Event	N/A. Events are always reported with multi-object payloads, even if they contain only one object	A "Students" element containing the collection of all successfully modified Student objects. The actual contents of each Student

[39] *Note that depending on the Data Model, some Data Object Services may be required to always send full object representations within every Update Event. From the infrastructure point of view, this is optional behavior and cannot be verified.*

		object is determined by the replacement HTTP field defined above.

Example:

The Response to an "update" Request of 3 objects, where the middle object did not previously exist.

```
<updateResponse>
  <updates>
    <update id="df789e1c-dfe7-4c18-8ef0-d907b81ea61e" statusCode="200"/>
    <update id="41953aaa-2811-11e6-b67b-9e71128cae77" statusCode="404">
      <error id="6f789e1c-dfe7-4c18-8ef0-d907b81ea61e">
        <code>404</code>
        <scope>Not Found</scope>
        <message>Student does not exist!</message>
      </error>
    </update>
    <update id="ff789e1c-dfe7-4c18-8ef0-d907b81ea61e" statusCode="200"/>
  </updates>
</updateResponse>
```

Note that the error is embedded in the second update, and the results correspond directly to the order of the objects in the original Request. This allows the Consumer to correlate the error to the actual sub request which failed.

The two successful update operations are reported back with their identifying id attributes.

5.14. Delete

Message Type	Single Object Payload (on success)	Multi-Object Payload (on success)
Delete Request	There is no payload. The URL to which the Delete Request is issued indicates which object is being deleted (i.e. the "id" identifying the Student is the last segment of the Request's URL, in conformance with standard REST conventions).	A "deleteRequest" element, which consists of multiple, "delete" object ids. Since REST conventions do not support payloads on HTTP DELETE messages, all multi-object Delete Requests are conveyed via an HTTP PUT message containing an additional HTTP Header Field value of *methodOverride* set to *DELETE.*

Delete Response	An HTTP Response with a status of 204 (no content). There is no Payload.	An HTTP Response with a status of 200 (success). It contains a payload consisting of a single "*deleteResponse*" element of the form shown below.
Delete Event	N/A. Events are always reported with multi-object payloads, even if they contain only one object	A "Students" element containing the collection of all successfully deleted Student objects. The actual contents representing each Student being deleted consists solely of the id element of the deleted Student.

Request Example Multi-Object Delete:

```
<deleteRequest xmlns="http://www.sifassociation.org/infrastructure/3.2.1">
    <deletes>
        <delete id="df789e1c-dfe7-4c18-8ef0-d907b81ea61e"/>
        <delete id="abc89e1c-34e7-4cde-908a-d9abc81ea09a"/>
        <delete id="ff789e1c-dfe7-4c18-8ef0-d907b81ea61e"/>
    </deletes>
</deleteRequest>
```

Response Example Multi-Object Delete:

```
<deleteResponse xmlns="http://www.sifassociation.org/infrastructure/3.1">
  <deletes>
    <delete id="df789e1c-dfe7-4c18-8ef0-d907b81ea61e" statusCode="200"/>
    <delete id="abc89e1c-34e7-4cde-908a-d9abc81ea09a" statusCode="404">
      <error id="6f789e1c-dfe7-4c18-8ef0-d907b81ea61e">
        <code>404</code>
        <scope>Not Found</scope>
        <message>Student does not exist!</message>
      </error>
    </delete>
    <delete id="ff789e1c-dfe7-4c18-8ef0-d907b81ea61e" statusCode="200"/>
  </deletes>
</deleteResponse>
```

5.15. Head

Head is an abbreviation for headers taken from the HTTP method HEAD. HEAD is identical to the GET method except it only returns the headers that would be present for the same

request and not the body. This can be more efficient and therefore useful in many situations. In order to realize these efficiencies, retrieving just the headers is supported on the Request Connector.

Example Uses of Head:

- Initialize “Changes Since” for the first time.
- Serve as a service level ping to check availability.
- Ascertain the number of pages a query will (likely) result in.
- Check if data has changed or if your copy is current.

For each supported GET call a provider **should** support the corresponding HEAD operation. If the provider is unable to support the HEAD method for a particular service an error of **405 (Method Not Allowed)** must be **returned. The consumer** should **then fallback to the corresponding GET operation. At this point the consumer may need to be built such that it ignores the body of in this request’s return.**

It is important to keep in mind that while a properly used HEAD is more efficient than a GET, that efficiency is most likely only realized on the wire. In most cases it will require nearly as many server resources to service a HEAD request as it would to handle an otherwise identical GET request. As always be judicious in your use of requests.

5.16. Functional Services

A Functional (or Job Object) Service encapsulates stateful process behavior as well as the data exchanged between applications implementing that process.

It does this by supporting all four methods of a Data Object Service Provider interface, but applying them to Object Services rather than Object data elements.

When a Consumer issues a “Create” to a Functional Service, it results in the creation of a new executing instance of the Function (a “Job”) rather than a new Data Object.

From a conceptual point of view, each Job instance contains a set of named “phases”, identical to every other Job created by that Function Service. These discrete phases define and encapsulate the sub-actions that need to be done, but they do not explicitly determine the ordering (since the phases defining a Function may be executed in different order, depending upon the implementation and the needs of the site where the Functional Service is deployed).

Once created, the Job instance can be queried to find out where in the process it is (what is happening, what is the current status of each completed phase) and the Job may issue Events as its internal phases are completed.

Each Job Phase is represented by:

- A Phase name
- A status (*NotStarted, InProgress, Completed, Failed)*
- A defined Object Service corresponding to that Phase (which supports some or all of the set of service operations)

The creator of the Job can therefore:

- Monitor the status of the Job (through querying the Job instance or by receiving Job level Events)
- Interact with the Job at any phase by issuing Query, Create, Modify and Delete requests (the meaning of which are determined by the Functional Service itself).
- Receive Events from the various Phases of the Job

Examples of such Functional Services might include *StudentLocator and EndOfYearRollover.*[40]

[40] *Further details of Functional Service functionality will be made public after the input from a SIF 3.0 Proof of Concept project is received*

