Access 4 Learning Community

Powered by: SIF

Simple. Secure. Scalable. Standard.

SIF Infrastructure Specification 3.3: Utilities

www.A4L.org

Version 3.3, May 2019

1. Introduction

A SIF 3 Utility Service is a Data Service that *provides* a data object related to the SIF infrastructure, rather than an object which is part of a particular locale-specific data model.

- Utility services support some or all of the identical set of Consumer Requests as Object services (Query, Create, Update and Delete).
- Some Utility services may be subscribed to, and publish object Events when their internal data changes.
- Utility services are accessed by Consumers through the *requestsConnector* and return any delayed Responses and Events into the specified *Consumer Polling Queue*.
- Utility services can be located by Consumers through a Provider entry in the Global Zone section of the Service Providers Registry.
- Utility services are documented with the infrastructure rather than the set of educational Object Services which use that infrastructure (i.e. the data model), and they are present in every release based upon that infrastructure version, independent of locale.
- Utility services may be implemented independently of the core Infrastructure Services or any other Utility Service, and integrated into a site-wide solution in the same manner as implementations of other Object Services.
- Utility services have their formats defined within the Infrastructure namespace (http://sifassociation.org/infrastructure/3.3), rather than in one of the Data Model namespaces which define the payload of Object and Functional Services.

1.1 Required Reading

Please refer to the *Basic Architecture* document for an understanding of the terminology, concepts and global XML element definitions, Service types, operation descriptions and exchange choreographies that will be referred to here. Additionally, the legal notices contained in the Basic Architecture apply to this document.

Please refer to the *Infrastructure Services* document to gain an understanding of the "core" Infrastructure Service components, which these Utility Services supplement.

Some familiarity with both those documents is assumed as a prerequisite, as all Utility Services will employ the Service framework and leverage the specific Infrastructure Services described there.

1.2 Utility Service Types

As noted, all Utility Services are accessed through the *Requests Connector* Service and follow the standard Object Service API in that they support some or all of the defined Object Service Provider interface methods (Query, Create, Update and Delete). There are two additional characteristics that are unique to this set of Services.

1.2.1 Administrative Level Authorization

Particularly in Brokered Architectures, there may be one or more administrative Consumer applications that are used to configure and maintain the Environments of other Consumers. These applications may be granted authorization rights to additional Provider Service CRUD Methods unavailable to non-administrative Consumers.

Where they exist, such extensions are site specific and except in a few cases, will not be explicitly defined in this document.

1.2.2 Scope

All Utility Services span the entire Consumer's Environment. They each have a single entry in the Service Providers Registry, located in the *environment-global* Zone.

This means that in any Consumer Request, the combination of a *serviceType* value of "*UTILITY*" and the *serviceName* provides enough information to allow the Connector in a Brokered Architecture to first determine the identity of the Utility Service Provider, which should receive it, and then construct the exact Service URL to route it to.

As a result, any *zoneId* provided by the Consumer in a Utility Service Request does not affect the ultimate destination of the request, unlike what occurs if the serviceType value is "*OBJECT*" or "*FUNCTIONAL*". In the latter cases, a Consumer-provided *zoneId* may result in the Connector forwarding the Consumer's Request to an entirely different Service Provider URL (ex: an SIS for Special Ed students or one spanning only a specific middle school).

When a Service Consumer issues a Request to a Utility Service Registry however:

- If the Consumer explicitly specified the *environment-global* Zone, a non-qualified Query is assumed and the Utility Service returns all entries in its Registry. If the Consumer specified any other legal Zone (or its default *zoneId* was supplied by the Connector) only those entries applicable to that *zoneId* are returned. Essentially the *zoneId* parameter serves as a query constraint on the entries returned from the Registry, instead of identifying a particular Utility Service instance.

- For the Service Providers Registry, this means a Consumer gets only the Provider entries for its default Zone or, if it specified the *environment-global* Zone, it gets all Service Providers for all Zones.
- For the External Code List Registry (and others), things are somewhat different. The default assumption is that all External Code Lists are applicable to all Zones, and are therefore "Global" in scope, and their entry has a *zoneId* value of *environment-global*.

  Whenever this is not true (for example grade enumerator codes may be different for different school Zones in the District Environment) a Zone-specific External Code List entry of the same name but with the different set of values is created, and tagged with the corresponding *zoneId* value.

  When a Zone-specific query request is issued, each entry in the global list is then examined in turn:

  - If there is no corresponding Code List with that name defined specific to the selected Zone, the Global Code List entry is returned.
  - If there is a specific entry for that Code List in the selected Zone, that entry is returned instead.

The end result is that when a Consumer issues a request to a Utility Service with a *zoneId* other than *environment-global*, only those entries (ex: Providers, External Code Lists) relevant to the specified Zone are returned. If the zoneId specified is *environment-global*, all entries across all the Zones in the Consumer Environment will be returned.

1.2.3 Functionality

The generic functionality provided by each Utility Service in supporting the Environments Provider API is shown in the following table. Unless otherwise noted, no Utility Service supports eXtended Query or Paged Query Requests.

These services will all be described in further detail in the sections that follow.

Utility Registry Services	Functionality	Access
Zones	Contains the name and description of all Zones in the Consumer's Environment (not all of which may be accessible by the Consumer).	Only Query access is enabled for non-Administrative Consumers. Change Requests are not supported.
Providers	Contains entries with the name, service type, vendor information and any supported Contexts of all Service Providers in a given Zone (not all of which may be accessible by the Consumer).	Only Service Providers operating in a Brokered Architecture may create and change the contents of their Registry Entries. All Consumers in both types of Architectures have Query functionality.
Namespaces	Used to retrieve namespace URIs and their corresponding schema location URLs, currently valid within the zone.	Only Query is enabled for non-Administrative Consumers. Change Requests are not supported.
Code Sets	Validates specific code values claimed to be part of an identified named code set, used to reduce breaking changes introduced via normative references to external sources like the NCES Handbook.	Only Query is enabled for non-Administrative Consumers. Change Requests are not supported.
Named eXtended Query	Accepts, validates (per-Consumer) and provides access to previously submitted or pre-defined Query Templates.	All approved Service Providers can use the supplied Token to retrieve the corresponding eXtended Query Script using Query by (Query Template Token) ID. This script is then used as the basis of expanding and executing a Consumer issued eXtended Query Request containing that Token and a set of values corresponding to the parameters in the template. Only Query is enabled for non-Administrative Consumers. Change Requests are not supported.

Other Utility Services	Functionality	Access
Alerts	Accepts and forwards reported alerts (notifications warnings and errors) in accordance with the pre-specified diagnostic and security policies defined by the site administrator.	All Service Consumers can create an Alert, and issue Query by (Alert) ID to access their own previously reported Alerts. Only administrative Consumers can access and change the Alerts issued by other Consumers.
Privacy Obligations	Contains entries for all data protection obligations applicable in a given Zone to data requested by the Consumer, as captured in a Privacy Obligation Document.	Only Query is enabled for non-Administrative Consumers. Change Requests are not supported.

2. Zones Registry

The Zones Registry Utility Service contains the name and description of all Zones visible to a registered Consumer within its Environment. Some Zones may contain services which are not accessible by the Consumer due to authorization restrictions.

2.1 Zones and Consumer Requests

A *Zone*[1] is a collection of Service Providers and associated Utility Registry information within the Consumer's Environment, pre-organized by the site Administrator to correspond to a discrete hierarchical grouping within the owning educational organization such as a school or district, or an alternative grouping such as the set of applications supporting Special Ed students. Zone identifiers are chosen by the administrator and can follow any convention that best meets the needs of the deploying organization.

Each Data Object and Functional 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. As noted above, there is one "special" Zone (*environment-global*) that is reserved for "globally available" (i.e. Architecture-wide) Utility Services.

The Zone in which the Service is to be found always qualifies every Consumer request for any Provider Service. Each Service Consumer is assigned a "*default*" Zone at Registration time, which is used whenever a specific Zone is not explicitly included in one of its Provider Service Requests. If there is no matching registered Service Provider for any Consumer Request, the request must fail.

2.2 Presence and Scope

The presence of a Zones Registry Utility Service is mandatory for all Environments that support Consumer self-provisioning, whether Direct or Brokered. In the simplest case (typically a Direct Architecture provided by an SIS or LMS application), the Zones Registry might consist solely of the Zone ID and Zone Description (with no additional parameters) for two Zones:

- *environment-global*

[1] *Please refer to the SIF 3.0 Base Architecture document for a more complete description of a SIF Zone.*

- XXX

In this case "XXX" would be the "default" Zone assigned to all registered Consumers. Its value might represent the name of the application providing the non-utility services (such as Assessment).

A Consumer does not need to access the Zones Registry if it interacts solely with the set of Service Providers contained in its assigned "default" Zone, and the available Utility Services in the environment-global Zone.

2.3 Supported Operations

The set of Zones is fixed for each Registered Consumer, and does not change while the Consumer is active. Therefore the Zones Registry Service is not required to support change requests or to publish change events.

The only guaranteed operation for non-Administrative Consumers is Bulk Query. There is no support for dynamic Query, and there are no defined Zones Registry Service Paths or eXtended Templates.

2.4 Further Documentation

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:Zone

3. Providers Registry

There is one *Providers Registry Utility* Service per Consumer Environment. The Registry contains a list of Provider Entries, each of which includes:

- A Zone ID
- The type of the provided Service in that Zone (Utility, Data Object, Functional, Service Path or Extended Query).
- The URL identification of the provided Service.
- Any specific Context that Service supports.
- Any Service functionality extensions (such as support for Dynamic Queries or the ability to provide "Total Count")

All potentially accessible Services have an entry in the Providers Registry (including the Providers Registry Utility Service itself), although full or even partial Consumer access to that Service is determined by the access rights currently granted in the Consumer's Environment object[2], and is not guaranteed.

As noted in the Basic Architecture document sections on Service Types, there are 6 types of Services which could be represented in the Provider Service Registry.

Service Type	Defined	URL Service Identifier in Registry Entry (example)	Operations
Data Object	Data Model Schema	students	Query, Create, Update and Delete
Utility Object	Infrastructure Schema	zones	Defined in this document (Query only in this case)
Functional Service	Data Model Schema and Functional Description	studentRecordExchanges	Specific to the Function
Service Path	Data Model Binding for Object Type. Schema identical to schema of object type name in final URL segment	*sections/{}/students* where "{}" indicates the section to report Students for	Read Only

[2] *Please refer to the Environments Service section in the Infrastructure Services document.*

Extended Query	Data Model Binding defines Schema and Script contents	StudentSnapshot	Read Only
Service[3]	Infrastructure Schema	filterrequest	Request, Response

3.1 Supported Operations

In many cases, a Service Consumer will be pre-provisioned to be able to access the set of Service Providers it must rely on to perform its functions. In that case, it does not need to utilize the Providers Registry Service.

However, at sites where the security policy is "authorization on demand", any Service Consumer with the proper authorization rights may be expected to dynamically utilize the Providers Registry to discover available Service Providers in its default Zone or elsewhere. At that point it can dynamically self-Provision itself to issue Requests and (where applicable) to subscribe to Events from one or more of these Providers.

Such usage requires the ability to Query the Providers Registry.

If the Consumer has the proper authorization (and is deployed in a Brokered Architecture), it can also register itself as a Service Provider by creating one or more of its own Provider Entries that will in turn be visible to other Consumers[4]. If the Service Provider implementation supports multiple service options (multiple Zones, multiple Contexts, multiple Data Model version numbers, etc.) it **must** create multiple corresponding entries in the Provider Registry.

The Providers Registry Service must publish change Events when Service Entries are added, updated or deleted. The Providers Registry Service itself does not support dynamic Query, and there are no defined Providers Registry Service Paths or Extended Queries.

Operation	Description
Query	Provides the Consumer with a list of all accessible Service Providers within the specified Zone, or (if access was through the environment-global Zone), a list of all Service Providers of all types in all Zones.

[3] *Initially designed to support calls to the Data Protection Enforce, this type of service may be used for other purposes in the future.*
[4] *This is very similar to when a SIF US 2.6 application provisions itself as an Object Provider.*

Create (Brokered Architectures only)	Allows a Consumer to initially provision (or attempt to provision) itself as a Service Provider. As in any Create Request, multiple Provider Entries may be created with this call (as when the Consumer requests to be a Provider of a given object type for multiple Contexts). Success results in the creation of one or more new Provider Entries.
Delete (Brokered Architectures only)	Removes one or more specified Entries in the Providers Registry. Except for Administrative Tools, this Request must be rejected for all Consumers other than the one which created the entry in the first place. Success results in the deletion of the specified entries from the Providers Registry. At this point these Services become unavailable to all Consumers[5].
Update	Prohibited

3.2 Further Documentation

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:Provider

[5] *Such an Entry Deletion request is generally issued as part of an ordered "unregister" sequence for the Provider, prior to its deleting the Environment. It is not required that a Provider issue this however.*

4. Namespaces Registry

The Namespaces Registry contains the set of XML namespace URIs and their corresponding schema location URLs that are currently valid within the Environment. Both Service Consumers and Providers **SHOULD** stay synchronized with this registry in order to prevent and detect namespace mismatches while exchanging messages.

In terms of guaranteeing interoperability and enforcing element-level privacy restrictions, controlling the Namespaces contained in exchanged messages is as important to Environment Administrators as controlling the XML elements themselves, because Consumer to Provider interoperability requires standardization of the name, value and scope of every element exchanged.

This imposes a Consumer to Provider co-dependency, in that each has to construct messages containing only the namespaces that the partner has the schemas for and can validate. In general, such schema agreement is static and predefined by either the version of the Infrastructure or Data Model supported within the Environment. In this case, neither the Consumer nor Provider need access the Namespaces Registry.

However in Environments where namespaces are subject to change, where customized profile extensions to the core SIF schemas have been made, and especially where the Environments Provider, Data Warehouse or Service Provider is bridging between versions, the Namespaces Registry allows for both synchronization and just in time retrieval of needed schemas, to minimize rejection of messages containing unexpected namespaces.

In particular, with the separation of infrastructure and data model in SIF 3, there is no longer one Namespace that all Servers and Providers within the Environment must conform to. The Namespaces Registry is provided as the only sure way for a Consumer to determine what the Providers within the Environment will expect to see. It allows Service Consumers wanting to place XML on the wire or retrieve data over the wire to dynamically determine whether the namespaces (including version) they rely on are supported within the Environment, before they begin operation.

4.1 Service Implementation Strategy

Each entry in the Namespaces Registry carries only the URL of a single schema file, which by convention leverages the XML Schema Namespaces themselves, other Namespaces present in the Registry, and/or a replicable relative path structure and one or more file references from the targeted location.

The ability to Query the Namespaces Registry is required in all Environments where the Namespace Registry is supported. The Namespaces defining the infrastructure and Data Model versions supported in the Environment can be read, but entries can only be created and deleted by Administrators. There are no Namespaces Registry Events.

4.2 XML Schema Snippet

```
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="imports/xml/xml.xsd"/>
```

4.3 Supported Operations

Request	Direct Architecture	Brokered Architecture
Query	M	M
Create	P	P
Update	P	P
Delete	P	P
Events	P	P

There is no support for dynamic Query, and there are no defined Namespaces Registry Service Paths or Extended Queries.

4.4 Further Documentation

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:Namespace

5. Code Sets Registry

The SIF standard includes multiple normative dependencies on external code sets such as the NCES Handbook. When one of these code sets is revised, the first application utilizing the updated codes will likely not interoperate with previously deployed SIF applications conforming to the earlier version of the code set.

Any such code set revision then represents a potential breaking change which is asynchronous to the SIF standard release cycle. As a result, Consumer / Provider interoperability is impacted even between SIF applications conforming to the same SIF minor release.

Even in the case where all Code Set values are defined directly in the SIF standard, the addition of a single new value to an enumerated code set in a minor release can break interoperability in ways that the addition of a new element cannot. Reception of a new element can be simply ignored by an application conforming to an earlier release, because everything it expected is still present in the arriving message. But reception of a new code set value cannot be ignored, because it provides a value which cannot be determined to be legal for an existing element, that the older application may still need to store.

The Code Set Registry Service provides a way for all legal codes to be defined outside of the SIF Specification, while allowing changes (additions and replacements) of external code set values to be easily verified by the recipient, so as not to break existing Consumer / Provider interoperability.

Codes are arranged (and scoped) under their respective code sets within the Registry. They are specifically not namespace qualified, so they can be built up from multiple sources and remain simple to employ.

5.1 Service Implementation Strategy

Consumers and Providers can use the Code Set Registry to verify and then accept codes that were not yet defined when they were written. Depending upon the implementation, all or part of the Code Sets Registry can be seeded either manually or by having the Registry automatically draw in codes from multiple sources and reconcile them with the manual entries.

Query is a mandatory Code Set Registry operation. All others (*create, update* and *delete* of both code lists and their individual entries) are optional, and may require manual administrative actions to achieve.

However entered, if an unexpected code value is received by a Consumer or Provider, the Code Set Registry **must** be capable of verifying whether the value is in fact legal.

5.2 Alternatives

If a Code Sets Registry Utility Service is not available in the Environment, Consumers and Providers **MUST** employ one or more of the following strategies for dealing with the arrival of a code which does not match the expected enumerated list of values defined when they were created.

- **Blind Trust**: What is received is accepted by the recipient and placed into its data store. This is the simplest approach and removes the need to support or reference the Code Sets Utility Service. However, it may result in inconsistent or erroneous data, since it ignores the possibility of sender error.
- **Other Import Techniques:** Additional code sets can be copied directly from their source on the web or hand entered.

If the code value is not found in the Registry, the remainder of the message can be accepted, but either way an Alert **SHOULD** be issued.

5.3 Supported Operations

Request	Direct Architecture	Brokered Architecture
Query (both Bulk and Paged)	M	M
Create	P	P
Update	P	P
Delete	P	P
Events	M	M

There is no support for dynamic Query, and there are no defined Code Sets Registry Service Paths or Extended Queries. Individual Code Set entries may be returned if Query by ID is requested.

5.4 Further Documentation

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:CodeSet

6. Extended Query Registry

The Extended Query Registry contains the Environment-wide collection of registered Extended Queries (named queries similar to a report or view, that **may** take predefined URL Query Parameters). This collection defines the entire set of Extended Queries that Consumers may legally issue for execution by the related Service Provider. Each Extended Query has an associated unique ID under which it may be referenced.

Many if not all of the queries in a typical Extended Query Registry may have originally been specified as part of the Infrastructure "binding" of the Data Model release, in the form of a profile. For example, an Extended Query called "StudentSnapshot" will, when requested, result in a Response with a predefined format. Such templates are known "in advance" by developers of both Service Consumer and Service Provider applications. A registry serves primarily as a transparency and documentation tool. Critically it provides the infrastructure with an exact name to secure access to the query through security and privacy controls. Exactly how the response is generated remains an implementation detail left to the Service Provider.

To invoke an Extended Query, a Consumer specifies the ID of an Extended Query in a Query Request, along with a set of values for any associated parameters[6].

The Provider **may** already either:

- Have its own logic ready to generate the expected results in the Response. When true, it allows Extended Query in a Request to be serviced by a Provider, which neither needs to know nor support the scripting language used in the registry entry.
- Have its own copy of the script corresponding to the supplied ID. In this case it proceeds to execute the script and return the results as the Response to the Query (either bulk or paged). Providers **should not** be expected to execute scripts ad-hoc, regardless of whether they are defined as XQueries or in some other manner.

If the Provider is unable to support this Extended Query, than a SIF Error Message **must** be returned with an HTTP Error Code of 405 (Method Not Allowed).

6.1 Usages

Extended queries are designed to overcome the limitations of Dynamic Query and Service Paths. Here we include some common scenarios when an Extended Query is the best fit.

[6] *Please refer to the Extended Query section in the Base Architecture Document for further details.*

6.1.1 Formulas

A Formula is an Extended Query that does more than just retrieve data; it may do calculations and other useful transformations to greatly increase efficiency of data transfer (over the wire), as long as security and privacy rules are observed.

6.1.2 Joins

Joins are queries that combine multiple objects. While Service Paths will help you interrelate object efficiently, if you need data from multiple objects in a single operation, or if the join operation is not covered by an existing service path, an Extended Query is in order.

6.1.3 Parameters

While you **may** place requirements in a Dynamic Query or a Query by Example, and **may** retrieve related objects with a Service Path, no explicit provision is made to combine these methods in this standard. Extended Query parameters provide you with the opportunity to have complex logic and give the caller some control over what is being processed and returned.

While named parameters provide the ability to pre-approve Extended Queries where one or more values differ between Requests, Providers must be able to ensure these values specified are not more than a single value. To that end only alphanumeric values (including floating point numbers) with spaces or the empty string are allowed as parameter values. This restriction **may** require what is conceptually one value to be communicated through multiple parameters. For instance: “{:ns:}:{:object:}”

If parameters are defined, they **should** be called out in the parameters section of a registry entry and the list **must** be comprehensive. This holds true, even if the list of parameters is empty.

6.1.4 Access and Ownership

Depending upon Extended Queries functionality and a Consumer’s authorization rights, a Consumer may create a Template, query all Templates in the Registry by either Paged Query or Query by ID (the template token), and delete only those Templates it has created.

6.2 Supported Operations

In most cases the contents of the eXtended Query Template Registry will be fixed, and only the Query operation will be supported.

Request	Direct Architecture	Brokered Architecture
Query	M	M
Create	O	O
Update	P	P
Delete	O	O
Events	O	O

There is no support for dynamic Query and there are no defined Service Paths for this service. Additional no Extended Queries that self-reference this Service may be defined. An individual Extended Query entry may be returned if Query by ID is requested.

6.3 Further Documentation

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:Xquery

7. Alerts

There is a single Alerts Utility Service available in the Environment, located in the *environment-global* Zone. It allows providing authorized Service Consumers with the ability to:

- Log an "alert" (error, warning or status change) by creating an Alert object
- Retrieve specified collections of these Alert objects via standard query mechanisms
- Subscribe to Alert creation Events

The ability to detect when new Alerts are created could be used by an administrative Consumer to monitor the performance of newly installed applications, detect when another Consumer had indicated it was the cause of a problem, or track and flag Alerts above a pre-specified priority level. Such Alerts might include preventative maintenance warnings from a Queue Infrastructure Service that its Consumer has stopped polling for arriving messages, or that the number of messages in the Queue have crossed a predefined threshold.

Ideally an Alert should contain as much identifying information about the problem being reported as possible. However "nesting" an erroneous message inside the Alert can generate unanticipated problems if the error being reported is that the original message format was invalid. For this reason, the original message **MAY** be omitted or described, and when it is included it **MUST** be properly escaped (included as CDATA).

7.1 Supported Operations

The required levels of supported Alerts Utility Service operations (both Requests and Events) available to properly authorized Consumers are indicated in the table below (where M = Mandatory, O = Optional, P = Prohibited).

Operation	Direct Architecture	Brokered Architecture
Query	O	O
Create (single object form only)[7]	O	M
Update	P	P
Delete	P	P
Event	O	M

[7] *Only one Alert object can be specified in each Alert Create Request.*

Support for Query is optional, and when provided for non-Administrative-level applications, returns only Alert objects created the same application that issued the Query. There is no support for dynamic Query, and there are no defined Alert Service Paths or eXtended Query Templates. Individual Alerts may be returned if Query by ID is requested.

7.2 Error Handling

Proper use of the Alerts Service is an essential part of the error logic handling of every Service Consumer and Provider. The following message exchange situations during which errors occur define how SIF 3 components **should** interact with the Alert Utility Service, and are applicable to all Service types (Object, Functional, and Utility).

7.2.1 A Service Provider receives a Consumer operation invocation which it rejects

Actors	Consumer which invokes a Service operation Functional or Data Entity Service Provider supporting that operation Alerts Utility Service
Preconditions	The Consumer invokes a Service operation. It reaches the Service Provider, which rejects it. The reasons could range from: • Invalid XML in the Request payload • Omission of a mandatory element • Incorrect data value (ex: an Object ID which did not correlate to a stored object)
Actions	The Service Provider should then perform the following actions: • Return a *"NAK"* Response to the client, indicating at least the error category, code and description which identify the problem. • Optionally determine whether this error is a duplicate or deserves to generate a unique Alert (i.e. is it identical to a previous error about the same Consumer that has already been logged?). If not, ignore it (or file a different Alert). • If applicable, the reporting Service should issue a *Create Alert* Request to the Alerts Utility Service, identifying itself as the Reporter and the Consumer as

	the Cause, and providing as much information about the error as seems reasonable. This should indicate at least the description, error, category and code which document the problem in the Consumer Request. Along with the appropriate error category and code: o For invalid XML, return the error reported from the parser o For omission of a mandatory element, return the element tag name o For an invalid data value, return the element tag name and the erroneous value • Take any other action (ex: logging the error to a local file) as seems appropriate
Post Conditions	When receiving the rejection of its Service method invocation, the Consumer should then perform only those actions which relate to its own internal logic (i.e. log the problem to a local file, report it to the user, back out of a transaction, adjust its internal data base). However, the Consumer does not normally create an Alert object reporting the problem, as it can rely upon the Service Provider to do that.

7.2.2 The Consumer receives a Service Provider Response which it rejects

Actors	Consumer which invokes a Service Provider operation Functional or Object Service Provider supporting that operation Alerts Utility Service
Preconditions	The Service Provider Response to a previously invoked operation is seen and rejected by the Consumer. The reasons could range from: • Invalid XML in the Response payload • Omission of a mandatory element • Incorrect data value (ex: an enumerated value such as a Country Code which does not correspond to a valid entry as the Consumer understands it, in an External Code List)
Actions	The Consumer has no way to report this problem back to the Service Provider. It should then:

	• Perform only those actions which relate to its own internal logic (i.e. log the problem to a local file, report it to the user, back out of a transaction, adjust its internal data base). This closely parallels the required actions in the case when the Service Provider has rejected the Consumer's Request. • If the Consumer believes the cause of the disconnect rests with the Service Provider, it should also issue a *Create Alert* Request to the Alerts Service, identifying itself as the Reporter and the Service Provider which returned the Response as the Cause, providing as much information about the problem as seems reasonable. This should indicate at least the description, error, category and code which documents the problem in the Response. Along with the appropriate error category and code: o For invalid XML, return the error reported from the parser o For omission of a mandatory element, return the element tag name o For an invalid data value, return the element tag name and the erroneous value
Post Conditions	In rare cases, the Consumer may "re-request" the operation using alternative parameters if it has reason to anticipate that will produce better results.

7.2.3 A Service Provider posts an Event which a Subscribing Consumer rejects

Actors	Consumer Subscriber to Service Provider Events Service Provider which publishes a Change Event Alerts Utility Service
Preconditions	An Event is received and rejected by the Subscriber. The reasons could range from: • Invalid XML in the Event payload • Omission of a mandatory element (in a Create Event only) • Incorrect data value (ex: an enumerated value such as a Country Code which does not correspond to a valid entry as the client understands it, in an External Code List) • Unexpected Event content received (e.g. Create Event with the ID of an object the Consumer thought already existed)

Actions	The Subscriber has no way to report the problem back to the Service. It must ignore the Event in terms of processing it or updating its Data Store. However, it should log the problem to a local file and / or report it to its end user. There is also the possibility that an erroneous Provider is publishing a stream of faulty Events, which if left unchecked, would result in a flood of new Alert Objects from each Subscriber. As a result, it is recommended that before a Subscriber creates an Alert to report an Event error, it first determines that it has never previously (or not in a pre-specified time) reported an error contained in an Event of this type. If the Subscriber believes the cause of the disconnect is a problem with the Publisher, and if it determines it is not redundant to do so, it should also issue a *Create Alert* Request to the Alert Service, identifying itself as the Reporter and the Service which published the Event as the Cause, providing as much information about the original Event as seems reasonable. This should indicate at least the description, error, category and code which documents the problem. Along with the appropriate error category and code: • For invalid XML, return the error reported from the parser • For omission of a mandatory element in a Create Event, return the element tag name • For an invalid data value, return the element tag name and the erroneous value
Post Conditions	The Subscriber should record that it has created an Alert because of an erroneous Event received for this Object type. This will help minimize the number of Alerts reporting the same problem should the cause reside with the Event Publisher.

7.2.4 A Service Provider detects an inactive Consumer

Actors	Stateful Service Provider dependent on additional Consumer operations Alerts Utility Service
Preconditions	The Service Provider detects a Consumer error because an expected Service operation has not been invoked. An example would be when a Polling Queue Instance determines that the preset maximum time limit to wait for the arrival of a GetNextMessage invocation has expired, indicating that the Consumer may be offline.

Actions	The Service has no opportunity to report the problem back to the Consumer (and in fact the Consumer may not even be active). The Provider should then perform the following actions: • Determine whether this error is a duplicate or deserves to generate a unique Alert (i.e. is it identical to a previous error about the same Consumer that has already been logged?). In the example given, if the error is deemed duplicated, it should be ignored. • If applicable, the reporting Service should issue a *Create Alert* Request to the Alerts Utility Service, identifying itself as the Reporter and the Consumer as the Cause, and providing as much information about the error as seems reasonable. This should indicate at least the description, error, category and code which document the "state" problem the Service is having with the Consumer When reporting "*Consumer inactivity*", along with the appropriate error category and code, the Alert object created should contain Alert Level, Error and Description element values which provide indications of: o Time of last client activity o Size (where known) of current Message Queue o Severity Level of Alert (warning that limit exceeded or application offline error) • Take any other action (ex: logging the error to a local file) as seems appropriate

7.3 Further Documentation

Note: There is no "Id" attribute as that is supplied by the Alerts Utility Service Provider when the Alert Object is actually created.

For full data structure and examples please see the Infrastructure data model documentation.

Data Model:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/UtilityServices.html#obj:Alert

8. Privacy Obligations Registry

The Privacy Obligations Registry contains the set of data protection obligations captured in the Privacy Obligations Documents (PODs) which must be enforced. These obligations are ideally expressed twice each. Once in a human readable way, as it relates to a contract or law. Another time in a machine-readable[8] fashion as part of a data access field list, condition list, deletion requirement or other predefined construct. For SIF integrations the data protection obligations are scoped by Consumer, Zone, and Context.

8.1 On the Wire Enforcement

The place where interoperability can play a key role in enforcing privacy is to reduce the data that arrives at a consumer to only what is needed. A POD has two key ways of doing this. First the data access field list **may** be used to express the fields that are permitted to arrive at a given Consumer. Second the condition lists **may** define the records that are allowed to be sent to a Consumer often in conjunction with a scoping mechanism such as a Context.

8.2 Off the Wire Enforcement

When it comes to enforcing the privacy rules expressed in a POD, the Consumer has a role to play. Some of the machine-readable portions, such as data deletion requirements, **must** be handled by the Consumer. Additionally, the human readable sections are most effective when shared with the user or administrator and an interface to view them **may** be provided.

8.3 Service Implementation Strategy

In order to achieve privacy all components (the Provider, Consumer, and where applicable Broker) need access to the PODs that impact them. The below overview is designed to explain how to get the desired POD and where to impact the data transfer (see 8.4 Further Documentation). These steps assume you will be getting the PODs out of this services interface, however if one of these components also provides this Utility Service that **may not** be the case.

[8] Machine readable portions of the POD **may** be configured when consumed or **may** be preconfigured and confirmed when consumed. For instance, a Context may be setup ahead of time to satisfy a condition.

The Consumer **must**…

1. Retrieve its Data Privacy Marker (DPM) when it starts and whenever it receives a 403 error.
2. Retrieve the POD by included its DPM in the where clause of a query to this service.
3. Confirm it can function under the conditions of the POD.
4. Affirm it can either automate the requirements of the POD, or notify the administrator of the tasks they must complete in order to conform to the POD.
5. Include the current dataPrivacyMarker header in all requests it makes.
6. Treat data retrieved or delivered through any mechanism (events, another API, human input, etc.) under the same Benchmarks.

The Provider and/or Broker **must**…

1. Implement or relay request to the POD service.
2. Return the most specific POD based on the DPM.
3. Confirm the DPM is acceptable before responding to requests. Otherwise provide the Consumer with a 403/Forbidden HTTP error code.
4. Run every outgoing payload through the Privacy Enforcer Service (wherever it is implemented) *before* sending it on to the Consumer.
5. Include the dataPrivacyMarker header with the DPM used when the payload was processed.

Note: Because the POD that was applied may be specifically referenced, PODs must be retrievable by their resource ID like any other service.

Note: To ensure the POD(s) attached to a DPM can be retrieved the where clause on the podToken field **must** be supported by the Utility Service Provider.

8.4 Further Documentation

For a full view of how privacy can be achieved using the SIF 3 Infrastructure please see the Privacy Services volume. There you will find detailed documentation of how to architect an integration that uses data responsibly, include full diagrams detailing private data exchanges.

Privacy Volume:
http://specification.sifassociation.org/Implementation/Infrastructure/3.3/PrivacyServices_3-3.pdf

For full data structure and examples please see the Infrastructure data model documentation.

Data Model: http://specification.sifassociation.org/Implementation/Infrastructure/3.3/infrastructures/PrivacyServices.html#obj:Pod

8.5 Supported Operations

Request	Direct Architecture	Brokered Architecture
Query	M	M
Create	P	P
Update	P	P
Delete	P	P
Events	P	P

PODs reflect Privacy Contracts and need to be administered by IT staff.