Sertainty™ Services Guide

Draft 12

Version: V3.5.0

Table of Contents

List of Tables

List of Figures

1. Overview

The Sertainty Services is a collection of service extensions to the UXP technology object. The Services share metadata and setup; however, each service extension can be utilized independently. The following service extensions are supported:

- Data Service (ds::)
- Delegate Service (dl::)
- Event Service (event::)
- Identity Service (id::)
- UXP Service (sf::)

1.1. Delegate Service

Delegate Service is a proprietary extension of Services providing remote user-access management for UXP Objects. The Delegate Service provides a flexible Delegate Service Management System as well as the statically built Delegate Identity Subscription list for UXP Objects. A Delegate Identity permits external management of authorized Users for a UXP Object.

So, at the time of its creation, a UXP Object may be bundled in different ways:

A *static* bundle builds into the UXP Object the Identity of the User authorized to access it. The advantage of building a static UXP is its simplicity. A static UXP has no need for an external delegate service to resolve the name of authorized users, and therefore it is self-contained and self-reliant. The disadvantage of the static UXP is that the name of the authorized users are built into it, and therefore if the list of authorized users changes, the UXP needs to by unbundled, the list of users edited and then UXP has to be rebuilt. This is not practical in a dynamic environment where there is significant churn in the users list.

An alternative is to use the concept of *delegates* within UXP. A delegate is a pseudo user object that has, as its attribute, the address of a service that can resolve its user access permissions. So a UXP designer can build a UXP with a delegate object and when this UXP is accessed by a user, UXP uses the delegate object to access an external service that can resolve the name of its constituent users. The advantage of this dynamic technique is that the user list can be changed at any time without impacting or having to rebuild the UXP. The disadvantage is that it requires an external service to resolve the constituent user list.

A UXP may have zero, one, or more built-in delegates. As a use case, consider a document that needs to be shared by both the Engineering and Accounting departments with different management structures. The document can be protected by building two delegates into UXP; an Engineering delegate object and an Accounting delegate object. The two delegates can potentially point to different services (one for each department) that can resolve user permissions to access the UXP independently. The Accounting department manages its own users and the Engineering department manages its own through their corresponding services.

For a delegate-based UXP, the UXP must be accessed when the computer is on the grid and can access the user authentication service. However, the designer of the UXP may also turn on the caching option of the delegate, in which case the UXP will cache the latest user access list and will use the cached list when offline. This option can be turned off (by UXP designer) when only on-grid access is to be allowed.

The UXP can also be designed in the *hybrid* mode in which case the UXP may have both the

built-in user list and one or more delegate objects. For example, when there is a fairly static set of users that need to access a document (e.g. department heads) and then there is a dynamic list of other users that may have a need to use the document, a hybrid UXP can be built where the department head identifications are built into UXP statically and then one or more delegates are used to manage the dynamic list of other potential users.

1.2. Event Service

The Event Service provides a repository for UXP Object Events. The UXP Object can optionally record Events at several locations, including a Services instance. When the Event is received by the Services API, the Event is recorded and available for reporting by an authorized agent of the Services.

The Event consists of a set of properties that describe the UXP Object operation, the status of the operation and other environmental elements that permit extensive reporting on data activities. Events can be recorded to a Services server only if there is a matching user registered with the service. If an event is sent to a server having no matching user, the event will be discarded.

1.3. Identity Service

The Identity Service provides all the technology to construct an ID via the web without the need to install sertainty libraries locally. Currently the Identity and associated artifacts are not stored in the Services Database.

The Services provides a base ID Def XML allowing the User to personalize the XML. The user is editing / managing the XML on the local device. The User is responsible for maintaining and securing the XML. The web-enabled Client, using the Services API, passes the XML to the Identity Service within Data Seruvces. DS will add any additional attributes, can remove unnecessary attributes, can validate XML and can publish the iic to a “buffer”; user must “save-as” to capture the *.iic

Assumption:

- **Ability to create / access a UXP**
- **Ability to share**

It utilizes the private Identity Definition XML to construct an *.iic file.

2. Design and Data Flow

Services consists of:

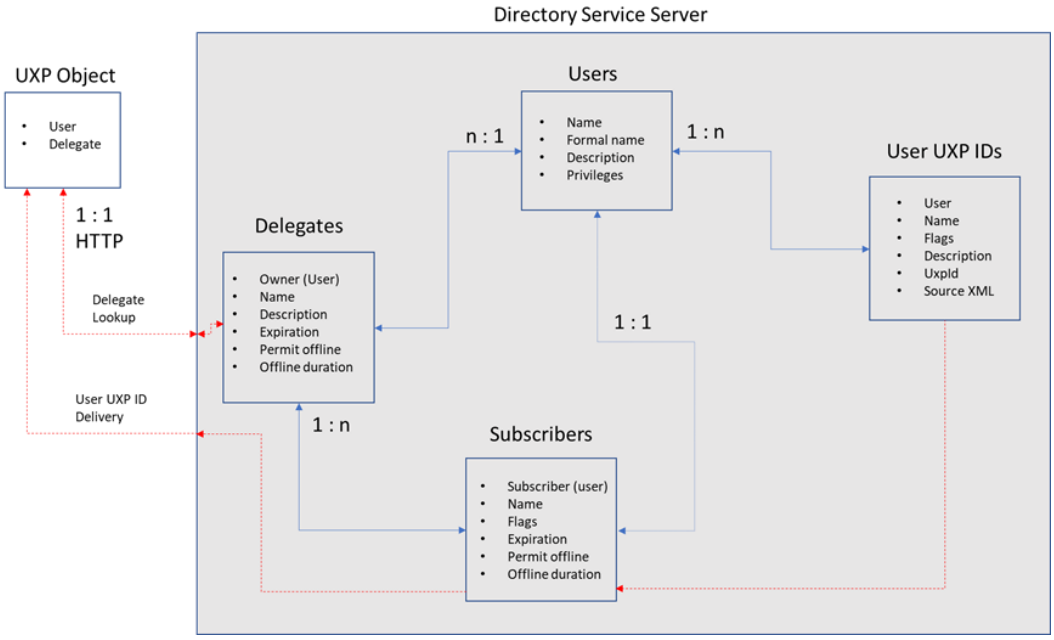
- Server-side shared UXP SQLite Database managed within a UXP Object
    - Services can be extended to utilize MySQL as the primary data store.
- Web service API
- Native C++ API
- Script extensions for local maintenance

The development interface also supports the concept of a local database that can be used without a server. A server can be a default server as provided by Sertainty or a private server. For a private server configuration, Services can be used to set up Sertainty-specific functions.

As noted, the Services Database is a shared UXP SQLite Database. This shared Database is utilized by Services extensions. The UXP SQLite Database has a private API that permits direct access to SQLite artifacts within a conventional read-write UXP Object.

This Database manages the following logical entities:

Figure 1, High Level Sertainty Delegate Service Architecture



Services Server

The URL required for access to the Services Server must be known by the Server. When a Delegate Identity is generated, it must contain the Services Server URL. This URL is required by the UXP Object authentication process to locate the Delegate Service managed within Services.

The Server URL is part of an auto-generated Delegate Identity. Therefore, the Server URL must always match the URL within the Delegate Identity, or the User attempting access will never be able to authenticate.

Users

A User is a logical representation of a UXP Identity. The primary key is a username that must also exist within the UXP Identity. When a Delegate Substitution occurs, the user's UXP ID is used to send to the requesting UXP object for authentication.

Attributes:

- Username – must exactly match a user in the user's UXP IDs.
- Formal name – User-supplied name.
- Description – User-supplied description of user.
- Email address – Email address of user.
- Privileges – A bit-position encoded integer value which encapsulates user privileges.
- Application data – Optional application data that is passed back to the host when a remote user is validate.
- UXP IDs used for delegate substitution, sessions, etc. A user can have many UXP IDs, each serving a different purpose.

Delegate Identity

A Delegate Identity is a generic identity that can be embedded in an existing UXP. Every Delegate Identity is owned by an existing User within the database and can be managed by that user.

When a Delegate Identity is created or updated, a unique UXP ID is generated with the necessary data to permit a UXP Object to contact the Services Database and validate attempted access via the Delegate Identity.

Attributes:

- Owner name – User who owns the Delegate Identity.
- Delegate Identity name – each userUser-supplied name. Must be uniquely named per User.
- Description – User-supplied description of the delegate.
- Generated UXP ID – Auto-generated UXP ID used to embed delegate in Workgroup IDs.
- Private key – Auto-generated key used for protecting user validation messages.
- Checksum – Identity checksum of delegate ID.
- Expiration – Optional expiration time of the delegate.
- Permit offline access – If set to 1, enables offline access via a cached delegate resolution.
- Offline duration – If "Permit offline access" is set to 1, this attribute sets the maximum number of hours that offline access is permitted.
- Access maximum – Sets the maximum number of times a user may access the UXP. A value of zero indicates no limit to the number of accessers.

Delegate Subscribers

A Delegate subscriber is a user who can utilize a delegate. The subscriber owner can manage the list. A user must be a valid delegate service database user.

Attributes:

- Delegate owner – User owner of the delegate identity.
- Delegate name – Name of delegate identity.
- Username as subscriber – User who is subscribed to the delegate identity.

- Expiration - Optional expiration date of delegate.
- Permit offline access – If set to 1, it enables offline access via a cached delegate resolution.
- Offline duration – If offline is permitted, this attribute sets a maximum number of hours that offline access is permitted.
- Access maximum – Sets the maximum number of times a user may access the UXP. A value of zero indicates no limit to the number of accessers. This value overrides the access maximum set at the delegate level.
- Access count – A running count of the number of times a user has successfully accessed the UXP. If this value is equal to or greater than the access maximum, then the remote authentication will be denied. If the access maximum is zero, then the access count is purely informational.

Services Journal

A Date Services Journal is an event / audit record of all activities that utilizes the Services Database. Users with JOURNAL privilege may access and maintain the Services Journal entries.

Attributes:

- Date/Time of activity
- Status of activity (Success or Failure)
- Action keyword to group journal entries.
- Authorized Services Session User
- Username relevant to activity
- Delegate Identity owner if applicable
- Delegate Identity name if applicable
- Brief message describing activity

Services Session

To access and manage Services Database elements, a User must authenticate and start a session using their UXP ID. The session will be authenticated the same way a User is authenticated when opening a UXP or creating a single-sign-on session.

2.1. Delegate Design Summary

A User must be registered with the Services in order to create Delegate Identity and to be a Delegate Subscriber. When the Data Service Database is created, the initial User must be provided in the form of a UXP ID. This User becomes the SYSADMIN for now. It is presumed that there should only be one User per human or machine. A SYSADMIN User is the only Services User who can create additional Users.

A User is a logical data item that can have one to many UXP IDs associated with it. a UXP ID can be designated as:

- Private ID – personal UXP ID that can only be used by the user.
- Public ID – personal UXP ID that may be published in a white pages or similar delegate.
- Session ID – personal UXP ID that is used to authorize a Services Management session.

- Delegate ID – personal UXP ID that is used to resolve an authentication attempt for a UXP containing the delegate.

The above ID types are bit masks, so a UXP ID may be used for more than one purpose.

To manage their user account, create deletes, etc., a User must open a session with the delegate service. The session is authenticated using the UXP ID that has the Session ID flag set.

> ***Notes:*** *By rule, a UXP ID must have the same username as the delegate service username.*

A User can create a Delegate Identity. A Delegate Identity is a placeholder User that can be embedded in UXP Objects as a User. When embedded, the Delegate Identity knows how to contact the home Services in order to validate and resolve an authentication attempt. The Delegate Identity cannot be used to authenticate access; it is only used as a path to the Services. The Delegate Identity resolution is started when authentication is attempted for a UXP Object. If the username is a locally embedded User in the UXP Object, then Services connection is not needed. If the username was not found in the local UXP Object, the UXP Engine will look for a Delegate Identity in the UXP. If a Delegate Identity is found, it will contact the Delegate's home Services, passing the username attempting access. Upon receiving the message from a UXP Object, the Data Service will verify the Delegate Identity that is requesting resolution. If valid, the Services will validate the username, which must be a Delegate Subscriber to that Delegate Identity. A Delegate Subscriber is a direct link to a registered User within the Services, so if a User has been removed or expired, the Delegate Subscription is also invalidated.

If the user passes all tests on the delegate service, the service will deliver to the UXP the actual UXP ID that was marked as a Delegate ID for the user. Once delivered to the UXP, the UXP engine continues with authentication using the user's UXP ID. It is assumed that the user is the only one who can authenticate using the delivered UXP ID. If authentication is successful, the user will be granted temporary access to the UXP contents inheriting rules that were attached to the embedded delegate. Upon UXP closure, the UXP discards all knowledge of the remote user. Future access attempts to the UXP by a user will follow the same remote access procedure.

A delegate has base attributes that can control access, such as expiration and local caching. The subscription has the same attributes that will override the delegate settings. This gives the delegate owner the ability to control access for the entire delegate subscriber list or at the subscriber level. Local caching is a feature that permits offline access for a delegate subscriber. By rule, a delegate must be able to connect to its home delegate service; however, in special cases where the user may need to be off the grid, the delegate owner can specify whether a delegate may resolve using a locally cached data structure. Locally cached data can be set with its own expiration so that a user cannot exploit the delegate system indefinitely. In order to use the cached data, a user must resolve via the delegate service at least once. This will activate local access for a specified access window.

A delegate is owned and manage by its parent user. The delegate owner also manages the subscriber list and its attributes. The subscriber list has no limit. Consider the subscriber list to be similar to an email distribution list where recipients can be added, modified or revoked without having to modify any UXP that utilizes the parent delegate.

> ***Notes:*** *Delegates are linked to a delegate service via a URL. If a URL for the delegate physically changes, then UXP objects containing the delegate will not be able to resolve delegate subscribers.*

E-R (Entity-Relationship) rules:

- One registered delegate service user per human/machine as a guideline

- A user may have many UXP IDs in the delegate service. At least one is required.
- A user's username must be a username for any UXP IDs associated with the account.
- A user must have exactly one UXP ID that has the Delegate ID flag set.
- A user must have exactly one UXP ID that has the Session ID flag set.
- A user can have zero to many delegates. The user owns the delegates that he/she creates.
- A delegate can have zero to many subscribers. A subscriber must be a registered user in the delegate service.

As noted, the delegate service database is a UXP SQL database. UXP SQL has a private API that permits direct access to SQL artifacts within a conventional read-write UXP.

2.2. The process flow

Ordinarily, if a data owner wishes to create a UXP that can be shared with many users, the native UXP requires all users to have identities within the UXP. For a large number of users, or for a highly dynamic group of users, this is impractical.

To solve this problem, a delegate service is set up. The server consists of a UXP SQL database and a listener service that supports a HTTP API. The database sets up the following:

- A local delegate service database is initialized by way of the native API or the script engine. A database cannot be created via the web interface.
- A user is created within the database for each owner and participant of a UXP/delegate relationship. The user must provide a valid UXP ID at creation time.
- A delegate is created to establish the participants in a UXP. A delegate is owned by a user within the database. When a delegate is created, it will create its own unique identity.
- Subscriptions are set up by the delegate owner. A subscription is a list of users from the database that can use the delegate for access into a UXP.
- A validation routine is provided that can validate an attempted UXP access. If the remote user is permitted to access a UXP via the delegate, the validation routine will return the user's actual UXP ID to the calling UXP.

To use a delegate, a data owner must acquire the delegate identity. Typically, the owner will create a workgroup ID that will contain the owner identity as well as the desired delegate identity. Once the workgroup ID is created, it can be used to create a UXP that contains the owner's identity and the delegate identity.

When a user attempts to open and authenticate into the UXP, they will use their username. If the UXP does not find the actual identity associated with the username, it will look for delegate identities. If it finds one, it uses the delegate identity to contact the delegate service that owns the delegate identity via the server validation API. If the delegate service recognizes the delegate identity, it determines if the requested user is actually a member of the delegate server and has been subscribed to the delegate. If so, the user's actually UXP ID is passed back to the UXP and authentication resumes as if the user has an identity within the UXP. If UXP is offline, the server validation API will use a cached copy to determine if the user is subscribed to the delegate, assuming offline access is permitted as described below.

> ***Notes:*** *the username lookup is case insensitive.*

2.3. Offline delegate resolution

A delegate subscription has an option to permit offline validation. This will allow a user to validate remotely, then subsequently validate using a local cached copy of the delegate message. An offline option can be and should be limited by a duration attribute that will automatically disable offline resolution based on an expiration date/time. The duration attribute is specified in hours. For example, a subscriber may be granted access to a UXP for 24 hours, but the subscriber MUST first open the UXP via remote validation *when online* to acquire the cached copy of the delegate. At that time, the expiration is calculated and hard-wired into the cached delegate for that subscriber. This process occurs every time a successful remote validation occurs. If a subscriber does not reconnect to the delegate service by the time the cached delegate expires, access to the UXP will be denied.

Offline settings can be set at both the delegate level and the subscription level. In both cases, the flag `permit_offline` must be set to 1 to enable offline access. Also, at both delegate and subscription levels, a second attribute, `offline_duration`, is used to limit the offline subscription to a specified number of hours.

The delegate rules for offline access are as follows:

- `Permit_offline` can be zero or one. A one enables offline access for all subscriptions. A negative value will translate to a zero value.
- `Offline_duration` can be zero or a positive value. A zero indicates no expiration. A non-zero value represents the number of hours that offline access is permitted. The offline expiration is automatically calculated when the cached delegate is created at the user endpoint.
- `Access_max` is the maximum number of successful UXP authentications for the current user. Once the access count reaches the maximum value, access via Delegate Services will be denied.

For a subscription, the same rules apply to `permit_offline` and `offline_duration` with the following additions:

- `permit_offline` can be -1, which indicates that the value should be retrieved from the delegate.
- `offline_duration` can be -1, which indicates the value should be retrieved from the delegate.
- `Access_max` can be -1, which indicates the value should be retrieved from the delegate.

In all cases, the values specified for the subscription will override the values set at the delegate level. This implies that if a delegate does not permit offline access (default behavior), an exception can be established for an individual user using subscription `permit_offline` value.

2.4. How to Use a Delegate

To use a delegate, both the owner of the delegate and potential subscribers must be registered users within the delegate service. Additionally, the users must have uploaded a valid UXP ID to the server under their respective accounts.

A delegate at the identity server consists of various attributes that are maintained by the owner of the delegate. As noted, the owner is a valid user at the delegate service. The delegate also contains a unique auto-generated ID that will be used by a subscriber. The ID can be by a subscriber via the API or it can be delivered to the user via the registered email

address of the subscriber. The delegate owner has full control over the delegate settings as well as the subscription settings for a user.

Once a delegate ID has been defined and a subscriber list has been built, a user can do the following:

1. Use the delegate ID to create UXP objects. Though feasible, it may give full control to a delegate subscriber, so care should be taken.
2. A more common approach is to create a workgroup with a personal ID and the delegate ID as a participant. This maintains your control as the owner of the data, yet allows the UXP to have dynamic access via the delegate subscriber list.

As the owner of the delegate, a user can build a very powerful access plan to data without requiring local copies of user identities within the UXP. For example, one has a design document that must be accessed by a team. Without a delegate, the team must be present as a potentially large workgroup so that the UXP can be accessed by team members. And, if the team changes, the data owner must redistribute a new UXP containing updated team membership. In a dynamic environment, this is impractical without the use of delegates.

With a delegate, the document owner can create a delegate containing the team members as subscribers. Then a workgroup would only consist of the data owner and the delegate identity. When the UXP is created, the team members in the delegate can access the data just as if the user identity is embedded in the UXP. If the team changes, the delegate owner only has to modify the subscriber list to add, update or remove subscriptions.

2.5. Working with existing private management services

By default, user management is performed by a Sertainty delegate management object against the required delegate service database. The database maintains the registered users, the delegate identities and the delegate subscriptions. It also maintains a journal of activity.

If the current environment has its own delegate services, a simple callback can be specified to link Sertainty Delegate Services when a user validation operation occurs. Sertainty will call the private delegate services first. If it passes, it will verify the Sertainty user.

To maintain a required level of security, a private database must be maintained. To keep a local delegate services in synch with the Sertainty Delegate Services, a simple user management API is defined in the C++ header file `uxpdelegateservice.h`.

2.6. Setting up a Service

If the current implementation already has a server and delegate service subsystem, the steps to use the Sertainty delegate service is as follows:

1. Define and implement a callback function to link any private user-management dependencies. The callback prototype is defined as a C-language function.
2. Create a variable of type `uxp::dataServices`. The variable can be used to initialize a new database, open an existing database and perform other management functions.

Prior to creating any delegates, the server name must be set using the `uxp::dataServices::setServer` call. This call saves the URL that will be called when a UXP object attempts to validate a remote user.

> ***Notes:*** *If the server URL changes after delegates have been defined, they must be recreated, or remote authentication will fail.*

Once a URL has been established, there needs to be an HTTP listener for the URL. For a custom service, any messages from a UXP will be encoded and unreadable by the service. For those messages, pass the message delegate to the `uxp::dataServices:executeWebFunction`. It will process the message, perform the requested action and construct a response. Like the original request, the response buffer is encoded so that only the calling UXP can decode it. The new message should be returned as a response to the original request.

For hosts that use the Sertainty Delegate Service, a supplied HTTP listener can be set up to process incoming requests via the URL.

2.7. Using MySQL as the Data Store

By default, the Services system creates a local UXP SQL database. The local database stores both service administration information as well as data to support users and delegates. This configuration will work without issues for small delegate installations. If, however, the user community can grow into a large collection, perhaps thousands or even millions, Sertainty recommends defining an external MySQL database that will work in concert with the UXP SQL administrative database.

The MySQL database will store all user and delegate data, thus, permitting managed growth of your installation.

To use the MySQL option, an installation must do the following:

- Install MySql locally, preferably MySQL 8 or later. Note the host name and port number used to access the database.
- Create the schema **uxpds** with a strong password of your choice. The Services server will automatically create the necessary tables and indexes with this schema.
- Create a user **uxpds** with full privileges to manage the **uxpds** schema. Services requires this user when storing and accessing necessary elements. Note, this user does not require access to any elements outside the schema **uxpds**.

When initializing the Services server environment, the MySql host name, port number and passsword for user **uxpds** must be provided. The information is then stored in the UXP SQL database and will be used when the server is started. The **uxpds** password can be changed via the Services database parameters API.

To properly work, the MySQL installation must be located by the Sertainty system. The following MySQL client libraries are required by the Sertainty system:

- **Linux**

  libmysqlclient.so

  Example installation: /usr/lib64/mysql/libmysqlclient.so

- **MacOS**

  libmysqlclient.dylib

  Example installation: /usr/local/mysql/lib/libmysqlclient.dylib

- **Windows**

  libmysql.dll

  Example installation: C:\Program Files\MySQL\MySQL Server 8.0\lib\libmysql

2.8. Getting Started with a Local Test Service

Until a remote listener service is established, one can try out delegates with a local delegate service. The UXP engine supports both a remote and a local URL for service

communications. To use a local service, the server URL must be `file://local`. When a delegate is linked to this URL, the UXP engine will talk directly to the delegate service without using a network protocol.

In the examples folder, a UXL script will create a sample local delegate service, create two sample users and set up a sample delegate. To test this, follow these steps:

- Run the script engine command line utility
- At the prompt, execute `file::cd("your-installed-examples-folder");` in which you must provide the path to your copy of the examples folder.
- @sample_delegate.uxl
    - This will create the delegate database, and two users. To finish the process, you will be prompted to enter the credentials for the second user: `SampleUser2@myemail.com.`
    - Next, the script will create a delegate and subscribe user `SampleUser2@myemail.com` to the new delegate.
    - Finally, the new delegate will be published as the file `delegate.iic` in the current folder.
- Exit from the script utility
- Run the Sertainty Assistant
    - If you have an ID library, open it; otherwise, create new ID library.
    - Once open, create a new ID.
    - Drag your personal user to the new ID as a member.
    - Go to the external examples folder and find the file `delegate.iic`**.** Drag that file onto the Assistant over the newly created ID. This will display a window describing the delegate ID from the file. Press OK and the delegate user from the IIC will be placed into your new ID in the library.
    - Now, you have an ID that contains two users: you and a delegate. A delegate user is just a placeholder and cannot be used to directly log into a UXP.
    - Right-mouse click and create a UXP from the ID. This will display a dialog for creating a UXP. Fill in the necessary information such as file name. Add an external file to the new UXP and hit the Create button. This will create a UXP on disk and may open the UXP within the Assistant. Go ahead and close the new UXP.
    - Open the new UXP. At the username prompt, you can center your personal user name or you can invoke the delegate system by entering `SampleUser2@myemail.com`. Since `SampleUser2@myemail.com` is not embedded in the UXP, the engine will attempt resolve the username via the delegate service that was created by the script. If found, it will starting prompting you as if `SampleUser2@mymemail.com` is a local user within the UXP.
- The sample UXL script will recreate the delegate service database every time it runs. This also invalidate the delegate that was embedded in the test UXP object. The existing delegate database can be used, but one will have to write custom UXL scripts to add or modify the existing database.

3. Web Server Communications

Web server communication is performed through an xml Request and Response messages.

Most communications to and from the Services server must be protected. The server is set up to handle clear messages, Sertainty-protected messages and user-protected messages. User-protected messages must have a user-defined access code as defined by the call setAccessCode within the server. Typically, the server administrator would set up the code.

Once a user code is defined, an endpoint that wishes to communicate with the server must encode the web request using the access code. All responses back from the server will be encoded using the same code as the original request.

3.1. Web Server Request

```
<Request>
  <Session>session-id</Session>
  <Function>function-name</Function>
  <ArgList>
    <Argument name="item-1">value</Argument>
    <Argument name="item-2">value</Argument>
    <Argument name="item-n">value</Argument>
    …
  </ArgList>
</Request>
```

Table 1, Request Elements

Element	Requirement	Description
<Request>	Required	The outer most XML tag for the request document.
<Session>	Required	Specifies the required session identifier as assign at API authentication time.
<Function>	Required	Name of the requested function. A function name consists of its service namespace followed :: and the base function name. Ex: dl::newDelegate(...)
<ArgList>	Required for arguments	Indicates a list of one or more arguments are to be passed to the function. If no arguments are required by the function, then tag is optional.
<Argument name="item-n">	Required for arguments	An argument to be passed to the function. The attribute name is used to identify an argument and may be optional.

3.2. Web Server Response

The format of a response is:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="item-1">value</Result>
    <Result name="item-2">value</Result>
    <Result name="item-n">value</Result>
    …
  </ResultList>
</Response>
```

Table 2, Response Elements

Element	Requirement	Description
`<Response>`	Required	The outer most XML tag for the response document.
`<Status>`	Required	Status code indicating the result of the function call.
`<StatusMessage>`	Required	Static message that is associated with the **<Status>** element.
`<ResultList>`	Required for result data	Indicates a list of one or more elements have been returned from the function. If no elements are returned by the function, then tag is optional.
`<Result name="item-n">`	Required for result data	A return data element from the function. The data must be in Base64 format. The attribute `name` is used to identify the result item and may be optional.

4. Deployment

The Services technology is based on UXP technology; therefore, the Sertainty Core system must be installed.

Once Data Sesrvices is installed, the installation folder contains the following developer folders:

- **developer/bin**

  Contains the necessary files for the Services technology.

  Linux Libraries

  libSertaintyDataServices.so (Shared library)

  dsplugins (Folder)

  MacOSX Libraries

  libSertaintyDataServices.dylib (Shared library)

  dsplugins (Folder)

  Windows Libraries

  SertaintyDataServices1.dll (Shared library)

  dsplugins (Folder)

  Note: All **Sertainty Object** libraries are built on MacOSX and Linux using **cdecl** function declarations. For Windows, libraries are built using **stdcall**. A caller must adhere to correct call standard or incorrect behavior may occur.

  All libraries are 64 bit and are compatible with standard compilers.

- **developer/documents**
  - **Sertainty Services Guide**

    The Guide comes in two formats: PDF and HTML. It contains a full description of the Services, include delegate services.
  - **Services Public API**

    The API guide is a reference guide for the public service functions. Like the Developer Guide, it comes in multiple formats.

    The following languages interfaces are supported:
    - C language
    - C++ language that requires the standard C++ environment.

- **developer/examples**

  Contains sample scripts and code.

Module	Description
sample_delegate.c	Demonstrates UXP authentication via Sertainty Delegate Services using C.
sample_delegate.cpp	Demonstrates UXP authentication via Sertainty Delegate Services using C++.

sample_delegate.uxl	Demonstrates UXP authentication via Sertainty Delegate Services using UXL scripting.

5. Web Functions

A data service is defined as a manager of authorized members of a delegate name list. A delegate name is a user that can be defined in a UXP and used to dynamically authenticate a user that does not have embedded credentials within the same UXP. Instead, the UXP will contact the designated delegate service server and attempt to locate the user within the delegate name user list. If found, the user's identity is sent back to the UXP and will be used to authenticate a user for access to the UXP contents.

There will only be one API function that can communicate with a server; however, the single API is overloaded to perform unlimited logical functions. Response values are also overloaded.

A request and its corresponding response are in XML format.

Table 3, Services Shared Web Functions

Function	Description
`ds::authenticate`	Authenticates a Services maintenance session.
`ds::closeSession`	Closes a Services maintenance session.
`ds::deleteJournal`	Deletes old rows from the journal of activities.
`ds::deleteUser`	Deletes a delegate user.
`ds::deleteUserId`	Deletes a UXP ID for a user.
`ds::getJournal`	Gets the current journal of activities.
`ds::getServer`	Gets the current server URL.
`ds::getUser`	Gets delegate user properties.
`ds::getUserId`	Gets a UXP ID for a user.
`ds::getUserIds`	Gets a list of UXP IDs for a user.
`ds::getUsers`	Gets a list of users.
`ds::newUser`	Adds a delegate user.
`ds::newUserId`	Adds a UXP ID to an existing delegate user.
`ds::openDatabase`	Opens the server database.
`ds::openSession`	Opens a new Services maintenance session.
`ds::publishUserId`	Publishes a UXP ID for a user.
`ds::setServer`	Sets the server URL.
`ds::updateUser`	Updates a delegate user.
`ds::updateUserId`	Updates a UXP ID for a delegate user.

Table 4, Delegate Service Web Functions

Function	Description
`dl::deleteDelegate`	Deletes an existing delegate name.
`dl::deleteDelegateAll`	Deletes all delegates by user.
`dl::getDelegate`	Gets the requested delegate.
`dl::getDelegates`	Gets a list of available delegate names for the specified user.
`dl::getPublicDelegate`	Gets the requested public delegate.
`dl::getPublicDelegates`	Gets a list of available public delegate names for the specified user.
`dl::getSubscribers`	Gets a delegate user list.
`dl::getSubscription`	Gets subscription properties.
`dl::getSubscriptions`	Gets a list of delegates to which the user has subscribed.
`dl::newDelegate`	Adds a new delegate name.
`dl::sendDelegateID`	Sends a delegate ID to a registered user.
`dl::sendDelegateIDToAddress`	Sends a delegate ID to an email address.
`dl::subscribe`	Adds a user to a delegate user list.
`dl::unsubscribe`	Removes a user from a delegate user list.
`dl::unsubscribeAll`	Removes a user from all delegate lists.
`dl::updateDelegate`	Updates an existing delegate.
`dl::updateSubscription`	Updates member subscription attributes.
`dl::verifyDelegateID`	Verifies a delegate ID with the Sertainty delegate service.

Table 5, Event Service Web Functions

Function	Description
`event::count`	Returns number of event records.
`event::delete`	Deletes event records.

event::get	Gets requested event records.

Table 6, Identity Service Web Functions

Function	Description
id::addConfigs	Adds a configuration to an ID definition.
id::addUser	Adds a user existing IIC to the ID definition.
id::applyRules	Applies a rule preset to the ID definition.
id::deleteUser	Deletes a user from an ID definition.
id::getUser	Gets the user definition from an ID definition.
id::newDocument	Creates a new ID definition document.
id::newUser	Creates a new user in an ID definition.
id::publish	Publishes an ID definition to an IIC.
id::update	Updates the ID definition properties.
id::updateUser	Updates a user in an ID definition.

5.1. ds::authenticate

Authenticates an open session to the Services maintenance service.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::authenticate</Function>
  <ArgList>
    <Argument name="challenge-1">response-1</Argument>
    <Argument name="challenge-2">response-2</Argument>
    <Argument name="challenge-n">response-n</Argument>
  </ArgList>
</Request>
```

Function Arguments:

challenge-n	Required	Challenge string as retrieved by a prior authenticate call.
response-n	Required	Response value for the

		corresponding challenge string.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="status">value</Result>
    <Result name="challenge-1">value</Result>
    <Result name="challenge-2">value</Result>
    <Result name="challenge-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

`status`	Status of authentication attempt. Possible values: **StatusAuthorized:** Session is ready to be used for scatter operations. **StatusCanceled:** Authentication attempt has been canceled. **StatusChallenged:** User must respond to challenges. Challenges will be included with this result. **StatusNotAuthorized:** Authentication failed.
`challenge-n`	Challenge string to be presented to the user. The number of challenges depends on the level of trust established by the delegate service.

5.2. ds::closeSession

Closes an existing session to the Services maintenance service.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::closeSession</Function>
</Request>
```

Function Arguments:

None

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

5.3. dl::deleteDelegate

Deletes a delegate name.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::deleteDelegate</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
name	Required	Name of the delegate to delete.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.4. dl::deleteDelegateAll

Deletes all delegates for a user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::deleteDelegateAll</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegates.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.5. ds::deleteJournal

Deletes old rows from the current activity journal. Requires **Journal** role to perform this operation.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::deleteJournal</Function>
  <ArgList>
    <Argument name="auth_username">value</Argument>
    <Argument name="username">value</Argument>
    <Argument name="delegateowner">value</Argument>
    <Argument name="delegatename">value</Argument>
    <Argument name="startdate">value</Argument>
    <Argument name="enddate">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

auth_username	Optional	Specifies the authorized user who performed the activity.
username	Optional	Specifies the user associated with the activity.
delegateowner	Optional	Specifies the delegate owner name. If not specified, all delegate owners will be included.
delegatename	Optional	Specifies the delegate name. If not specified, all delegate names will be included.
startdate	Optional	Start date/time prior to which all rows will be purged from the journal. If not specified, the beginning of the journal is used. Date format is ISO.
enddate	Optional	End date/time prior to which all rows will be purged from the journal. If not specified, the current date/time is used. Date format is ISO.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Notes: *This operation requires JOURNAL privileges for the current user.*

5.6. ds::deleteUser

Deletes a user from the delegate service. Current session must have UserAdmin role privilege to delete a user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::deleteUser/Function>
  <ArgList>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the user to delete.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user.*

5.7. ds::deleteUserId

Deletes a UXP ID for a user. Current session must have UserAdmin role privilege to delete an ID.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::deleteUserId/Function>
  <ArgList>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the ID owner.
idname	Required	Name of the UXP ID to delete.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the ID owner is not the same as the current user.*

5.8. event::count

Gets number of event records. Requires **READEVENTS** role to perform this operation.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>event::</Function>
  <ArgList>
    <Argument name="action">value</Argument>
    <Argument name="uxp_name">value</Argument>
    <Argument name="uxp_file">value</Argument>
    <Argument name="event_date_low">value</Argument>
    <Argument name="event_date_high">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

action	Optional	Specifies the action text for the event. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
uxp_name	Optional	Specifies the UXP name. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
uxp_file	Optional	Specifies the UXP file specification. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
event_date_low	Optional	Specifies the earliest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND operator.
event_date_high	Optional	Specifies the latest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND

		operator.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="count">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

count	The number of event records that matched the optional filter criteria.

Notes: *This operation requires READEVENTS privileges.*

5.9. event::delete

Deletes event records. Requires **MANAGEEVENTS** role to perform this operation.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>event::delete</Function>
  <ArgList>
    <Argument name="action">value</Argument>
    <Argument name="uxp_name">value</Argument>
    <Argument name="uxp_file">value</Argument>
    <Argument name="event_date_low">value</Argument>
    <Argument name="event_date_high">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

action	Optional	Specifies the action text for the event. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
uxp_name	Optional	Specifies the UXP name. The filter is applied as an equality and is

		combined with other filter options using the SQL AND operator.
uxp_file	Optional	Specifies the UXP file specification. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
event_date_low	Optional	Specifies the earliest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND operator. The format of the date is ISO.
event_date_high	Optional	Specifies the latest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND operator. The format of the date is ISO.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="count">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

None

Notes: *This operation requires MANAGEEVENTS privileges.*

5.10. event::get

Gets event records. Requires **READEVENTS** role to perform this operation.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>event::count</Function>
  <ArgList>
    <Argument name="action">value</Argument>
    <Argument name="uxp_name">value</Argument>
    <Argument name="uxp_file">value</Argument>
    <Argument name="event_date_low">value</Argument>
    <Argument name="event_date_high">value</Argument>
    <Argument name="start_row">value</Argument>
    <Argument name="count">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

action	Optional	Specifies the action text for the event. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
Uxp_name	Optional	Specifies the UXP name. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
Uxp_file	Optional	Specifies the UXP file specification. The filter is applied as an equality and is combined with other filter options using the SQL AND operator.
Event_date_low	Optional	Specifies the earliest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND operator.
Event_date_high	Optional	Specifies the latest date/time for the event. The filter is applied as a range and is combined with other filter options using the SQL AND operator.
start_row	Optional	Specifies the starting logical row number to return after filters have been applied.
count	Optional	Specifies the number of rows to return after filters and start_row have been applied.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="Row-n">value</Result>
    <Result name="Row-n">value</Result>
    <Result name="Row-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

Row-n	The event record in XML format. The document is encoded in base64.

Notes: *This operation requires READEVENTS privileges.*

5.11. dl::getDelegate

Gets the requested delegate name.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::getDelegate</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
name	Required	Name of the delegate.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name=”owner”>value</Result>
    <Result name=name”>value</Result>
    <Result name=description”>value</Result>
    <Result name=expiration”>value</Result>
    <Result name=permit_offline”>value</Result>
    <Result name=offline_duration”>value</Result>
    <Result name=access_max”>value</Result>
    <Result name=flags”>value</Result>
    <Result name=checksum”>value</Result>
    <Result name=uxpid”>value</Result>
  </ResultList>
</Response>
```

Return Arguments:

owner	Owner name for the new delegate.
Name	Name of the new delegate.
Description	Description of the delegate.
Expiration	Date at which subscription expires. A zero indicates no expiration. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service.
Offline_duration	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.
flags	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user
checksum	UXP ID checksum.
uxpid	UXP ID used as a delegate participant in a UXP. Buffer will be encoded in base64.

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.12. dl::getDelegates

Gets a list of available delegate names for the specified user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::getDelegates</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Specifies the owner of available delegates.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="name-1">value</Result>
    <Result name="name-2">value</Result>
    <Result name="name-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

name-n	Name of the delegate.

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.13. ds::getJournal

Gets the current activity journal. Requires **Journal** role to perform this operation.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getJournal</Function>
  <ArgList>
    <Argument name="action">value</Argument>
    <Argument name="auth_username">value</Argument>
    <Argument name="username">value</Argument>
    <Argument name="delegateowner">value</Argument>
    <Argument name="delegatename">value</Argument>
    <Argument name="startdate">value</Argument>
    <Argument name="enddate">value</Argument>
    <Argument name="startrow">value</Argument>
    <Argument name="maxrows">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

action	Optional	Specifies the action code for the activity.
auth_username	Optional	Specifies the authorized user who performed the activity.
username	Optional	Specifies the user associated with the activity.
delegateowner	Optional	Specifies the delegate owner name. If not specified, all delegate owners will be included.
delegatename	Optional	Specifies the delegate name. If not specified, all delegate names will be included.
startdate	Optional	Specifies the start date/time to filter the journal records. If not specified, the results start at the beginning of the journal. Date format is ISO.
enddate	Optional	Specifies the end date/time to filter the journal records. If not specified, the current date/time is used. Date format is ISO.
startrow	Optional	Specifies the row number to start relative to the result set. For example, if the search finds 100 rows, then by specified row 10, the result set will start returning rows at row 10 instead of row 1.
maxrows	Optional	Specifies the maximum rows to return from adjusted result set. If

		not specified, all rows will be returned.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="row-1">value</Result>
    <Result name="row-2">value</Result>
    <Result name="row-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

row-n	Set of columns separated by the '|' character that represents a journal entry. Columns are: • Date/time of entry in ISO format • Status of operation (0 – success, 1 – error) • Authorized username who performed the activity. • Username associated with the operation (if applicable) • Delegate-Owner (if applicable) • Delegate-Name (if applicable) • Action code for the activity • Message associated with the activity

Notes: *This operation requires JOURNAL privileges.*

5.14. dl::getPublicDelegate

Gets the requested public delegate name.

Format:

```
<Request>
  <Function>dl::getDelegate</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
name	Required	Name of the delegate.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name=”owner”>value</Result>
    <Result name=name”>value</Result>
    <Result name=description”>value</Result>
    <Result name=checksum”>value</Result>
    <Result name=uxpid”>value</Result>
  </ResultList>
</Response>
```

Return Arguments:

owner	Owner name for the new delegate.
Name	Name of the new delegate.
Description	Description of the delegate.
checksum	UXP ID checksum.
uxpid	UXP ID used as a delegate participant in a UXP. Buffer will be encoded in base64.

5.15. dl::getPublicDelegates

Gets a list of available public delegate names for the specified user.

Format:

```
<Request>
  <Function>dl::getDelegates</Function>
  <ArgList>
    <Argument name=“owner”>value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Specifies the owner of available delegates.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="name-1">value</Result>
    <Result name="name-2">value</Result>
    <Result name="name-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

name-n	Name of the delegate.

5.16. ds::getServer

Gets the server URL. The server URL must be set in order to create delegates.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getServer</Function>
</Request>
```

Function Arguments:

None

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="server-url">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

server-url	URL setting for the current server.

5.17. ds::getSystemFlags

Gets the server system flags. System flags modify server behavior.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getSystemFlags</Function>
</Request>
```

Function Arguments:

None

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="flags">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

flags	System flags setting for the current server. Value is a bitmask with the following supported values: 1 – Record all data updates in the journal. 2 – Record delegate lookup in the journal. 4 – Record session authentication in the journal.

5.18. dl::getSubscribers

Gets a delegate user list.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::getSubscribers</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="username">value</Argument>
    <Argument name="startrow">value</Argument>
    <Argument name="maxrows">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Username	Optional	Subscriber username filter. SQL LIKE operator is supported.
Startrow	Optional	Specifies a start row for the matching subscriber list.
Maxrows	Optional	Specifies the maximum number of rows to fetch from the matching subscriber list.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="username-1">value</Result>
    <Result name="username-2">value</Result>
    <Result name="username-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

username-n	Name of a user.

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.19. dl::getSubscription

Gets the subscription attributes. Current session must be the requested user or have UserAdmin role privilege to get user information.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::getSubscription</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="member">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
Name	Required	Name of the delegate.
Member	Required	Name of the user who is subscribed.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="owner">value</Result>
    <Result name="name">value</Result>
    <Result name="member">value</Result>
    <Result name="expiration">value</Result>
    <Result name="permit_offline">value</Result>
    <Result name="offline_duration">value</Result>
    <Result name="access_max">value</Result>
    <Result name="access_count">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

owner	Owner name for the delegate.
Name	Name of the delegate.
Member	Username of the subscriber.
Expiration	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service. A -1 indicates use

	the value from the delegate.
Offline_duration	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained. A -1 indicates use the value from the delegate.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Current successful access count.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.20. dl::getSubscriptions

Gets a list of delegates to which the user has subscribed.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::getSubscriptions</Function>
  <ArgList>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Specifies the user to list.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="row-1">value</Result>
    <Result name="row-2">value</Result>
    <Result name="row-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

row-n	Set of columns separated by the '|' character that represents a delegate. Columns are: • Delegate owner name • Delegate name

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.21. ds::getUser

Fetches a user from the delegate service. Current session must be the requested user or have UserAdmin role privilege to get user information.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getUser</Function>
  <ArgList>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the user to get.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="username">value</Result>
    <Result name="formalname">value</Result>
    <Result name="description">value</Result>
    <Result name="email">value</Result>
    <Result name="privileges">value</Result>
    <Result name="app_data1">value</Result>
    <Result name="app_data2">value</Result>
  <Result></Result>
</Response>
```

Return Arguments:

username	Username of the user.

Formalname	Formal name of the user.
Description	User description.
Email	User email address.
Privileges	Comma-separated list of assigned privileges.
App_data1	Optional application data that is passed back to the application during remote user validation.
App_data2	Optional application data that is passed back to the application during remote user validation.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.22. ds::getUserId

Fetches UXP ID from the delegate service. Current session must be the requested user or have UserAdmin role privilege to get user information.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getUserId</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="idname">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the user who owns the UXP ID.
Idname	Required	Name of the UXP ID to fetch.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name=”username”>value</Result>
    <Result name=”idname”>value</Result>
    <Result name=”flags“>value</Result>
    <Result name=”description“>value</Result>
    <Result name=”uxpid“>value</Result>
    <Result name=”source“>value</Result>
  <Result></Result>
</Response>
```

Return Arguments:

username	User own owns the UXP ID.
Idname	Name of the UXP ID.
Flags	Bitmask indicating the type of UXP ID. Possible bit values: 1 – ID is private to the owner. 2 – ID is public and may be used by others. 4 – ID is used for delegate resolution. 8 – ID is for delegate service session authentication. To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
Description	UXP ID description.
Uxpid	Binary UXP ID in iic format. Buffer will be encoded in base64.
source	Source XML document that defines the ID. This value can only be viewed or changed by the UXP ID owner.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.23. ds::getUsers

Gets a list of users.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getUsers</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="startrow">value</Argument>
    <Argument name="maxrows">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

Username	Optional	Username filter. SQL LIKE operator is supported.
Startrow	Optional	Specifies a start row for the matching user list.
Maxrows	Optional	Specifies the maximum number of rows to fetch from the matching user list.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="username-1">value</Result>
    <Result name="username-2">value</Result>
    <Result name="username-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

username-n	User name

Notes: *This operation requires ADMIN privileges for the current user.*

5.24. ds::getUserIds

Gets a list of UXP IDs for a user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::getUserIds</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="flags">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Specifies the owner of UXP IDs.
Flags	Required	Bitmask as a filter. Possible bit values: 1 – ID is private to the owner. 2 – ID is public and may be used by others. 4 – ID is used for delegate resolution. 8 – ID is for delegate service session authentication.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="idname-1">value</Result>
    <Result name="idname-2">value</Result>
    <Result name="idname-n">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

idname-n	UXP ID name

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.25. id::addConfigs

Adds a configuration to an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::addConfigs</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="configurations">value</Argument>
    <Argument name="user">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
configurations	Required	Configurations in XML format. The buffer must be encoded in base64.
user	Optional	User on which configurations will be added. If omitted, the configurations will be added at the ID level.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.26. id::addUser

Adds a user existing IIC to the ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::addUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="uxpid">value</Argument>
    <Argument name="user">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
uxpid	Required	IIC in binary format. The buffer must be encoded in base64.
user	Required	User to be copied from the IIC.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.27. id::applyRules

Applies a rule preset to the ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::applyRules</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="preset">value</Argument>
    <Argument name="rules">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
preset	Required	Name of the existing rule preset.
rules	Required	Comma-separated list of rule names to apply. An * indicates all rules.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.28. id::deleteUser

Deletes a user from an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::deleteUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="user">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
user	Required	User to be deleted.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.29. id::getRuleParameter

Gets a rule parameter from an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::getRuleValue</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="user">value</Argument>
    <Argument name="rule">value</Argument>
    <Argument name="parameter">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
user	Optional	Optional username from which the specified rule parameter will be read.
rule	Required	Name of the rule set.
parameter	Required	Name of the parameter from the rule set.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="value">value</Result>
  </ResultList>
</Response>
```

5.30. id::getUser

Gets a user from an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::getUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="user">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
user	Optional	User name to fetch.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="name">value</Result>
    <Result name="formalname">value</Result>
    <Result name="email">value</Result>
    <Result name="privileges">value</Result>
    <Result name="expiration">value</Result>
    <Result name="ch-1">response-value</Result>
    <Result name="ch-2">response-value</Result>
    <Result name="ch-...">response-value</Result>
  </ResultList>
</Response>
```

5.31. id::newDocument

Creates a new ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::newDocument</Function>
  <ArgList>
    <Argument name="name">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="personalname1">value</Argument>
    <Argument name="personalname2">value</Argument>
    <Argument name="personalname3">value</Argument>
    <Argument name="address1">value</Argument>
    <Argument name="address2">value</Argument>
    <Argument name="city">value</Argument>
    <Argument name="state">value</Argument>
    <Argument name="zipcode">value</Argument>
    <Argument name="country">value</Argument>
    <Argument name="phone1">value</Argument>
    <Argument name="phonetype1">value</Argument>
    <Argument name="phone2">value</Argument>
    <Argument name="phonetype2">value</Argument>
    <Argument name="phone3">value</Argument>
    <Argument name="phonetype3">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="photo">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

name	Required	Name of the new ID definition.
description	Required	Description of ID.
expiration	Required	Expiration in ISO string format **YYYY-MM-DDTHH:mm:ss**.

personalname1	Required	Personal name or identifier.
personalname2	Required	Personal name or identifier.
personalname3	Required	Personal name or identifier.
address1	Required	Street address.
address2	Required	Street addresss.
city	Required	City name.
state	Required	State name.
zipcode	Required	Postal code.
Country	Required	Country name.
phone1	Required	Phone number.
phonetype1	Required	Descriptive name for phone.
phone2	Required	Phone number.
phonetype2	Required	Descriptive name for phone.
phone3	Required	Phone number.
phonetype3	Required	Descriptive name for phone.
privileges	Required	Comma-separated list of ID privilege names.
photo	Optional	JPEG photo encoded in base64.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name=”iddefinition”>base64-value</Result>
  </ResultList>
</Response>
```

5.32. id::newUser

Creates a new ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::newUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="email">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="formalname">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="prompt-1">value</Argument>
    <Argument name="prompt-2">value</Argument>
    <Argument name="prompt-...">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
name	Required	Name of the new user.
email	Required	Email address.
expiration	Required	Expiration in ISO string format **YYYY-MM-DDTHH:mm:ss**.
formalname	Required	Personal name.
privileges	Required	Comma-separated list of user privilege names.
prompt-1	Required	Response value for prompt.
prompt-2	Required	Response value for prompt.
prompt-...	Required	Response value for prompt.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.33. **id::publish**

Publishes an ID definition to an IIC.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::new</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="options">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

`iddefinition`	Required	ID definition in XML format. The buffer must be encoded base64.
`options`	Optional	Comma-separated list of options. Possible values are: • V2ID Indicates the new ID will use the version 2 architecture. • COMPRESS Reduces the size of a version 2 ID, but also slightly reduces performance when opening a session or creating a UXP. By default, the version 1 ID architecture is used.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="uxpid">base64-value</Result>
  </ResultList>
</Response>
```

5.34. id::setRuleParameter

Sets a rule parameter in an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::deleteUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="user">value</Argument>
    <Argument name="rule">value</Argument>
    <Argument name="parameter">value</Argument>
    <Argument name="value">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
user	Optional	Optional username where the rule parameter will be modified.
rule	Required	Rule set name.
parameter	Required	Rule parameter to modify.
value	Required	New value to be assigned to the rule parameter.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.35. id::update

Updates an ID definition properties.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::update</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="personalname1">value</Argument>
    <Argument name="personalname2">value</Argument>
    <Argument name="personalname3">value</Argument>
    <Argument name="address1">value</Argument>
    <Argument name="address2">value</Argument>
    <Argument name="city">value</Argument>
    <Argument name="state">value</Argument>
    <Argument name="zipcode">value</Argument>
    <Argument name="country">value</Argument>
    <Argument name="phone1">value</Argument>
    <Argument name="phonetype1">value</Argument>
    <Argument name="phone2">value</Argument>
    <Argument name="phonetype2">value</Argument>
    <Argument name="phone3">value</Argument>
    <Argument name="phonetype3">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="photo">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
name	Required	Name of the new ID definition.
description	Required	Description of ID.
expiration	Required	Expiration in ISO string **YYYY-MM-DDTHH:mm:ss**.
personalname1	Required	Personal name or identifier.
personalname2	Required	Personal name or identifier.
personalname3	Required	Personal name or identifier.
address1	Required	Street address.
address2	Required	Street addresss.
city	Required	City name.
state	Required	State name.
zipcode	Required	Postal code.
Country	Required	Country name.
phone1	Required	Phone number.

phonetype1	Required	Descriptive name for phone.
phone2	Required	Phone number.
phonetype2	Required	Descriptive name for phone.
phone3	Required	Phone number.
phonetype3	Required	Descriptive name for phone.
privileges	Required	Comma-separated list of ID privilege names.
photo	Optional	JPEG photo encoded in base64.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.36. **id::updateUser**

Updates a user in an ID definition.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>id::newUser</Function>
  <ArgList>
    <Argument name="iddefinition">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="email">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="formalname">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="prompt-1">value</Argument>
    <Argument name="prompt-2">value</Argument>
    <Argument name="prompt-...">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

iddefinition	Required	ID definition in XML format. The buffer must be encoded base64.
name	Required	Name of the user to update.
email	Required	Email address.
expiration	Required	Expiration in ISO string format **YYYY-MM-DDTHH:mm:ss**.
formalname	Required	Personal name.
privileges	Required	Comma-separated list of user privilege names.
prompt-1	Required	Response value for prompt.
prompt-2	Required	Response value for prompt.
prompt-...	Required	Response value for prompt.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="iddefinition">base64-value</Result>
  </ResultList>
</Response>
```

5.37. dl::newDelegate

Adds a new delegate name.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::newDelegate</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="permit_offline">value</Argument>
    <Argument name="offline_duration">value</Argument>
    <Argument name="access_max">value</Argument>
    <Argument name="flags">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the new delegate.
Name	Required	Name of the new delegate.
Description	Optional	Description of the delegate.
Expiration	Optional	Date at which subscription expires. A zero indicates no expiration. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	Optional	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service.
Offline_duration	Optional	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained.
Access_max	Optional	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.
Flags	Optional	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.38. ds::newUser

Adds a new user to the delegate service.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::newUser</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="formatname">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="email">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="app_data1">value</Argument>
    <Argument name="app_data2">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the new user. Must be the same as a within the UXP Identity.
Formalname	Optional	Formal name of the user.
Description	Optional	User description.
Email	Optional	User email address.
Privileges	Required	Comma-separated list of assigned privileges. Possible values are: • ADMIN • JOURNAL
app_data1	Optional	Optional application data that is passed back to the application during remote user validation.
App_data2	Optional	Optional application data that is passed back to the application during remote user validation.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user.*

5.39. ds::newUserId

Adds a new UXP ID for a user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::newUserId</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="idname">value</Argument>
    <Argument name="flags">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="uxpid">value</Argument>
    <Argument name="source">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the owner of the new UXP ID. Must be the same as a within the UXP Identity.
Idname	Required	Name of the UXP ID.
Flags	Required	Bitmask indicating the type of UXP ID. Possible bit values: 1 – ID is private to the owner. 2 – ID is public and may be used by others. 4 – ID is used for delegate resolution. 8 – ID is for delegate service session authentication. To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
Description	Required	UXP ID description.
Uxpid	Required	Binary UXP ID in iic format. Buffer must be

		encoded in base64.
source	Optional	Source XML document that defines the ID. This value can only be viewed or changed by the UXP ID owner.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the ID owner is not the same as the current user.*

5.40. ds::openDatabase

Opens server database.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::openDatabase</Function>
  <ArgList>
    <Argument name="database">value</Argument>
    <Argument name="readonly">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

database	Required	Complete file specification of the server database.
Readonly	Required	True if opening database in read-only mode.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

5.41. ds::openSession

Opens a session to the Services maintenance service. The session must be authenticated to be operational.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::openSession</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="stop_time">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Username under which to authenticate.
stop_time	Optional	Sets the timeout option. Possible values: 0 - Session will timeout based on the UXP ID idle time settings. 1 - Disables timeouts. N - Stops the session at this absolute time. Number of milliseconds since Epoch. Default: 0

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="session-id">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

session-id	System-assigned number.

5.42. ds::publishUserId

Publishes UXP ID for a user.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::newUserId</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="idname">value</Argument>
    <Argument name="save">value</Argument>
    <Argument name="options">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the owner of the UXP ID.
Idname	Required	Name of the UXP ID.
save	Required	When true, the compiled UXP ID will be saved in the delegate service database.
Options	Optional	Comma-separated list of options. Possible values are: • V2ID Indicates the new ID will use the version 2 architecture. • COMPRESS Reduces the size of a version 2 ID, but also slightly reduces performance when opening a session or creating a UXP. By default, the version 1 ID architecture is used.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="uxpid">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

uxpid	UXP ID as a base-64 string.

Notes: *This operation requires ADMIN privileges for the current user if the ID owner is not the same as the current user.*

5.43. dl::sendDelegateID

Sends a Sertainty delegate ID to a registered user. The user can then the ID to access a delegate list within a UXP.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::sendDelegateID</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Username	Required	User to receive delegate ID. The email address will come from the user's account on the delegate service.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.44. dl::sendDelegateIDToAddress

Sends a Sertainty delegate ID to an email address. The user can then the ID to access a delegate list within a UXP.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::sendDelegateIDToAddress</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="address">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Address	Required	Email address to receive delegate ID.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.45. ds::setServer

Sets the URL for the current server. The URL must be set before creating delegates. If the server URL changes, current delegates will be regenerated. Any delegate already in use by a UXP will no longer permit remote validation.

To enable a private test server, set the URL to file:///localhost. This will call the delegate service via a local dynamic library instead of using the HTTP protocol.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::setServer</Function>
  <ArgList>
    <Argument name="server">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

server	Required	Server URL.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires SYSADMIN privileges for the current user.*

5.46. ds::setSystemFlags

Sets the system flags for the current server.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::setSystemFlags</Function>
  <ArgList>
    <Argument name="flags">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

flags	Required	Flags value as a bitmask. Support values are: 1 – Record all data updates in the journal.

		2 – Record delegate lookup in the journal. 4 – Record session authentication in the journal.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires SYSADMIN privileges for the current user.*

5.47. dl::subscribe

Adds a user to a delegate user list.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::subscribe</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="username">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="permit_offline">value</Argument>
    <Argument name="offline_expiration">value</Argument>
    <Argument name="access_max">value</Argument>
    <Argument name="access_count">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Username	Required	User to be added to the delegate user list. The user must be a valid user in the delegate service database.

Expiration	Optional	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	Optional	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service. A -1 indicates use the value from the delegate.
Offline_duration	Optional	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained. A -1 indicates use the value from the delegate.
Access_max	Optional	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Optional	Sets the currnt access count for the subscriber.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.48. dl::unsubscribe

Deletes a user from a delegate user list.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::unsubscribe</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Username	Required	User to be deleted from the delegate user list.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.49. dl::unsubscribeAll

Deletes a user from all delegate lists.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::unsubscribe</Function>
  <ArgList>
    <Argument name="username">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	User to be deleted from all delegate lists.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user.*

5.50. dl::updateDelegate

Updates an existing delegate.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::updateDelegate</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="permit_offline">value</Argument>
    <Argument name="offline_duration">value</Argument>
    <Argument name="access_max">value</Argument>
    <Argument name="flags">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
Name	Required	Name of the delegate.
Description	Optional	Description of the delegate.
Expiration	Optional	Date at which subscription expires. A zero indicates no expiration. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date

		format **YYYY-MM-DDTHH:mm:ss**.
`Permit_offline`	Optional	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service.
`Offline_duration`	Optional	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained.
`Access_max`	Optional	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.
`Flags`	Optional	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

5.51. dl::updateSubscription

Updates the subscription attributes. Current session must be the requested user or have UserAdmin role privilege to get user information.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>dl::updateSubscription</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="member">value</Argument>
    <Argument name="expiration">value</Argument>
    <Argument name="permit_offline">value</Argument>
    <Argument name="offline_duration">value</Argument>
    <Argument name="access_max">value</Argument>
    <Argument name="access_count">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name for the delegate.
Name	Required	Name of the delegate.
Member	Required	Name of the user who is subscribed.
Expiration	Optional	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	Optional	A 1 value will permit offline access to the UXP object. Otherwise, the delegate must be resolved by contacting the delegate service. A -1 indicates use the value from the delegate.
Offline_duration	Optional	For offline access, specifies the number of hours that offline access will be permitted. The expiration of offline access is calculated based on the last time online access was obtained. A -1 indicates use the value from the delegate.
Access_max	Optional	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Optional	Sets the currnt access count for the subscriber.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.52. ds::updateUser

Updates a user in the delegate service. Current session must have UserAdmin role privilege to update a user. Only specified columns will be updated in the user record.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::updateUser</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="formalname">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="email">value</Argument>
    <Argument name="privileges">value</Argument>
    <Argument name="app_data1">value</Argument>
    <Argument name="app_data2">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the user to update. Must be the same as a within the UXP Identity.
Formalname	Optional	Formal name of the user.
Description	Optional	User description.
Email	Optional	User email address.
Privileges	Required	Comma-separated list of assigned privileges.
App_data1	Optional	Optional application data that is passed back to the

		application during remote user validation.
App_data2	Optional	Optional application data that is passed back to the application during remote user validation.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

5.53. ds::updateUserId

Updates UXP ID for a user. Only specified columns will be updated in the UXP ID record.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>ds::updateUserId</Function>
  <ArgList>
    <Argument name="username">value</Argument>
    <Argument name="idname">value</Argument>
    <Argument name="flags">value</Argument>
    <Argument name="description">value</Argument>
    <Argument name="uxpid">value</Argument>
    <Argument name="source">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

username	Required	Name of the owner of the new UXP ID. Must be the same as a within the UXP Identity
idname	Required	Name of the UXP ID.

`Flags`	Optional	Bitmask indicating the type of UXP ID. Possible bit values: 1 – ID is private to the owner. 2 – ID is public and may be used by others. 4 – ID is used for delegate resolution. 8 – ID is for delegate service session authentication. To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
`Description`	Optional	UXP ID description.
`Uxpid`	Optional	Binary UXP ID in iic format. Buffer must be encoded in base64.
`source`	Optional	Source XML document that defines the ID. This value can only be viewed or changed by the UXP ID owner.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
</Response>
```

Return Arguments:

None

Notes: *This operation requires ADMIN privileges for the current user if the ID owner is not the same as the current user.*

5.54. verifyDelegateID

Verifies a Sertainty delegate ID. The ID must match the ID at the registered delegate service.

Format:

```
<Request>
  <Session>session-id</Session>
  <Function>subscribe</Function>
  <ArgList>
    <Argument name="owner">value</Argument>
    <Argument name="name">value</Argument>
    <Argument name="uxpid">value</Argument>
  </ArgList>
</Request>
```

Function Arguments:

owner	Required	Owner name of the delegate.
Name	Required	Name of the delegate.
Uxpid	Required	The UXP ID to verify. To be valid, the ID must be owned by the specified delegate owner and name. The ID checksum must match as well. Buffer must be encoded in base64.

Returns:

```
<Response>
  <Status>error-code</Status>
  <StatusMessage>error-message</StatusMessage>
  <ResultList>
    <Result name="valid">value</Result>
  </ResultList>
</Response>
```

Return Arguments:

valid	True if the delegate ID is valid and verified.

Notes: *This operation requires ADMIN privileges for the current user if the delegate owner is not the same as the current user.*

6. UXL Scripting Functions

The script engine is a command line interface to the **UXP Technology**. The delegate service has defined a set of external UXL functions that can be accessed via the script engine.

The delegate functions can be loaded using the following UXL function:

```
x::loadPackage("*", "delegate-lib-name")
```

Where **delegate-lib-name** is the delegate service shared library name. Once loaded, the identity delegate functions can be accessed. The library names by platform:

Linux:	libSertaintyDelegate
MacOSX:	libSertaintyDelegate
Windows:	SertaintyDelegate1

Table 9 – Function Summary

Function	Description
dl::deleteDelegate	Deletes an existing delegate name.
dl::deleteDelegateAll	Deletes all delegates for a user.
dl::getDelegate	Gets the requested delegate.
dl::getDelegates	Gets a list of available delegate names for the specified user.
dl::getPublicDelegate	Gets the requested public delegate.
dl::getPublicDelegates	Gets a list of available public delegate names for the specified user.
dl::getSubscribers	Gets a delegate user list.
dl::getSubscription	Gets subscription attributes.
dl::getSubscriptions	Gets a list of delegates to which the user has subscribed.
dl::newDelegate	Adds a new delegate name.
dl::sendDelegateID	Sends a delegate ID to a registered user.
dl::sendDelegateIDToAddress	Sends a delegate ID to an email address.
dl::subscribe	Adds a user to a delegate user list.
dl::unsubscribe	Removes a user from a delegate user list.
dl::unsubscribeAll	Removes a user from all delegate lists.
dl::updateDelegate	Updates an existing delegate.

dl::updateSubscription	Updates subscription attributes.
dl::verifyDelegateID	Verifies a delegate ID with the registered server.
ds::authenticate	Authenticates a delegate maintenance session.
ds::closeDatabase	Closes a delegate service database.
ds::closeSession	Closes a delegate maintenance session.
ds::deleteJournal	Deletes old rows from the journal of activities.
ds::deleteUser	Deletes a delegate user.
ds::deleteUserId	Deletes a UXP ID for a user.
ds::dropDatabase	Deletes local UXP and external entities.
ds::getAccessCode	Gets the current user access code.
ds::getChallenges	Gets the list of challenges as required by the ds::authenticate call.
ds::getDatabaseParameters	Gets the list of external database parameters.
ds::getJournal	Gets the current journal of activities.
ds::getServer	Gets the URL for the current server.
ds::getSystemFlags	Gets the system flags for the current server.
ds::getUser	Gets delegate user properties.
ds::getUserId	Gets a UXP ID for a user.
ds::getUserIds	Gets a list of UXP IDs for a user.
ds::getUsers	Gets a list of users.
ds::initDatabase	Initializes a new delegate service database.
ds::newUser	Adds a delegate user.
ds::newUserId	Adds a UXP ID for a user.
ds::openDatabase	Opens the delegate service database.
ds::openSession	Opens a new delegate maintenance session.
ds::publishUserId	Publishes a UXP ID for a user.
ds::setAccessCode	Sets the user access code.
ds::setDatabaseParameters	Sets the external database parameters.
ds::setResponses	Sets the required responses for the ds::authenticate call.
ds::setServer	Sets the URL for the current server.
ds::setSystemFlags	Sets the system flags for the current server.
ds::updateUser	Updates a delegate user.
ds::updateUserid	Updates a UXP ID for a user.
event::countEvents	Counts UXP event records.
event::deleteEvents	Deletes UXP event records.
event::getEvents	Gets selected UXP event records.

6.1. ds::authenticate

Authenticates the current Services session.

Format:

```
ds::authenticate(db, session)
```

Parameters:

db	Specifies the database handle as returned by ds::openDatabase.
Session	Specifies the Services session handle as returned by ds::openSession.

Returns:

status	Status of authentication attempt. Possible values: **StatusAuthorized** Session is ready to be used for delegate operations. **StatusCanceled** Authentication attempt has been canceled. **StatusChallenged** User must respond to challenges. Challenges must be retrieved using ds::getChallenges. **StatusNotAuthorized** Authentication failed.

6.2. ds::closeDatabase

Closes the specified delegate service database.

Format:

```
ds::closeDatabase (db)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.

Returns:

True if successful.

6.3. ds::closeSession

Closes the specified Services session.

Format:

```
ds::closeSession (session)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Returns:

True if successful.

6.4. ds::deleteJournal

Deletes journal records.

Format:

```
ds::deleteJournal(db, session, filter)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

`Filter`	Specifies an optional table of filters to apply to delete operation. Possible filter columns:
	• **auth_username** – User who performed activity.
	• **username** – User referenced by activity.
	• **delegateowner** – Delegate owner.
	• **delegatename** – Delegate name.
	• **startdate** – Milliseconds since Epoch.
	• **enddate** – Milliseconds since Epoch.
	A filter is a list of value/pair entries.
	Example:
	`list mylist;`
	`ValuePair pair;`
	`pair.name="username";`
	`pair.value="Greg";`
	`appendList(mylist, pair);`

Returns:

True if successful.

Notes: *This operation requires JOURNAL privileges for the current user.*

6.5. dl::deleteDelegate

Deletes a delegate.

Format:

```
dl::deleteDelegate (db, session, owner, name)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase`.
`Session`	Specifies the Services session handle as returned by `ds::openSession`.
`Owner`	Specifies the delegate owner name.
`name`	Specifies the delegate name.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.6. dl::deleteDelegateAll

Deletes all delegates for a user.

Format:

```
dl::deleteDelegateAll (db, session, owner)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Owner`	Specifies the delegate's owner name.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.7. ds::deleteUser

Deletes a user.

Format:

```
ds::deleteUser (db, session, username)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`

`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Username`	Specifies the user name.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user.*

6.8. ds::dropDatabase

Drops a Services database. For external entities, will only drop the SQL artifacts and not the database.

Format:

```
ds::dropDatabase (spec, [ ,dbmod, dbparams])
```

Parameters:

`spec`	Specifies the database file specification.
`Dbmod`	Optional database module that implements an external database. Possible values: • UXP Use the native UXP SQL access module. • MySQL Use the MySQL database module. • Oracle Use the Oracle database module (Under construction). • SQLServer Use the SQL Server database module (Windows-only) (Under construction). Default: UXP

`dbparams`	Optional list of database parameters for external database module. Parameters are a string separated by a '|' character. The parameters must be specified in the correct order based on the database module:
	• UXP ignores the parameters.
	• MySQL uses the following parameters:
	  ○ Database host name.
	  ○ Database port number.
	  ○ Database user password for user uxpds.
	• Oracle Not implemented.

	• SQLServer Not implemented.

Returns:

True if successful.

6.9. ds::getChallenges

Retrieves the current set of challenges for an authentication session.

Format:

```
ds::getChallenges (db, session)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Returns:

Packed challenge list. The format of the list is n**ame|prompt||name2|prompt**.

- Name is the challenge name.
- Prompt is the challenge user prompt.

6.10. ds::getJournal

Fetches journal records.

Format:

```
ds::getJournal(db, session, filter, &list)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by

	`ds::openSession.`

`Filter`	Specifies an optional table of filters to apply to fetch operation. Possible filter columns:
	
	• **action** – Action keyword for journal entry.
	• **auth_username** – User who performed activity.
	• **username** – User referenced by activity.
	• **delegateowner** – Delegate owner.
	• **delegatename** – Delegate name.
	• **startdate** – Milliseconds since Epoch.
	• **enddate** – Milliseconds since Epoch.
	• **startrow** – Starting row number to fetch.
	• **maxrows** – Maximum number of rows to fetch.
	
	A filter is a list of value/pair entries.
	
	Example:
	
	`list mylist;`
	`ValuePair pair;`
	
	`pair.name="username";`
	`pair.value="Greg";`
	`appendList(mylist, pair);`

`list`	A list variable to receive fetched rows. Each row is a string of column values separated by the '|' character. Column values are: • Date/time of entry in ISO format • Status of operation (0 – success, 1 – error) • Authorized username • Username who initiated operation (if applicable) • Message describing operation • Delegate owner (if applicable) • Delegate name (if applicable) • Action keyword

Returns:

True if successful.

Notes: *This operation requires JOURNAL privileges for the current user.*

6.11. ds::getAccessCode

Gets the user access code. The access code is used to protect a message when using the WEB service interface.

Format:

```
ds::getAccessCode(db, session)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Returns:

Current user access code.

Notes: *This operation requires ADMIN.*

6.12. ds::getDatabaseParameters

Fetches the external database parameters. Parameters are specific to the type of external database.

Format:

```
ds::getDatabaseparameters(db, session, &list)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

List	A list variable to receive fetched parameter values.
	The parameters must be specified in the correct order based on the database module:
	• UXP ignores the parameters.
	• MySQL uses the following parameters:
	  ○ Database host name.
	  ○ Database port number.

	○ Database user password for user uxpds. • Oracle Not implemented. • SQLServer Not implemented.

Returns:

True if successful.

Notes: *This operation requires SYSADMIN privileges for the current user.*

6.13. dl::getDelegate

Gets the specified delegate data.

Format:

```
dl::getDelegate(db, session, owner, name, &description,
                &expiration, &permit_offline,
                &offline_duration, &access_max, &flags,
                &checksum, &uxpid)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
description	A sentence describing the delegate.
Expiration	Date at which subscription expires. A zero indicates no expiration. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A value of 1 permits offline delegate resolution for the user.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.

`Flags`	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user
`Checksum`	The UXP ID checksum.
`Uxpid`	The UXP ID that can be embedded in a workgroup ID as a delegate user.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.14. dl::getDelegates

Fetches delegate names.

Format:

```
dl::getDelegates(db, session, owner, &list)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Owner`	Specifies the owner of the requested delegates.
`List`	A list variable to receive fetched names.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.15. dl::getPublicDelegate

Gets the specified public delegate data.

Format:

```
dl::getPublicDelegate(db, owner, name, &description,
                      &checksum, &uxpid)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
description	A sentence describing the delegate.
Checksum	The UXP ID checksum.
Uxpid	The UXP ID that can be embedded in a workgroup ID as a delegate user.

Returns:

True if successful.

6.16. dl::getPublictDelegates

Fetches public delegate names.

Format:

```
dl::getPublicDelegates(db, owner, &list)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Owner	Specifies the owner of the requested delegates.
List	A list variable to receive fetched names.

Returns:

True if successful.

6.17. ds::getServer

Gets the URL for the current server. The server URL value must be set prior to creating delegates.

Format:

```
ds::getServer(db, session)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Returns:

Server URL.

6.18. ds::getSystemFlags

Gets the system flags for the current server.

Format:

```
ds::getSystemFlags(db, session)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Returns:

System flags.

Notes: This operation requires SYSADMIN privileges for the current user.

6.19. dl::getSubscribers

Gets a delegate user list.

Format:

```
dl::getSubscribers(db, session, owner, name, filter, &list)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.

Filter	Specifies an optional table of filters to apply to fetch operation. Possible filter columns:
	• **username** – User name filter. Supports SQL LIKE syntax.
	• **startrow** – Starting row number to fetch.
	• **maxrows** – Maximum number of rows to fetch.
	A filter is a list of value/pair entries.
	Example:
	`list mylist;`
	`ValuePair pair;`
	`pair.name="username";`
	`pair.value="Greg";`
	`appendList(mylist, pair);`

list	A list variable to receive fetched user names.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.20. dl::getSubscription

Gets a user's delegate subscription.

Format:

```
dl::getSubscription(db, session, owner, name, username,
                    &expiration, &permit_offline,
                    &offline_duration, &access_max,
                    &access_count)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession.`
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
username	Specifies the user name.
expiration	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate.
Permit_offline	A value of 1 permits offline delegate resolution for the user. A -1 indicates use the value from the delegate.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution. A -1 indicates use the value from the delegate.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Current successful access count.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.21. dl::getSubscriptions

Gets a user's delegate subscriptions.

Format:

```
dl::getSubscriptions(db, session, username, &list)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user name.
list	A list variable to receive fetched user names. Each row is a value/pair where the first value is the delegate owner and the second value is a delegate name.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.22. ds::getUser

Get the specified user data.

Format:

```
ds::getUser(db, session, username, &formaname,
            &description, &email, &privileges,
            &app_data1, &app_data2)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user to fetch.

Formalname	Formal name of user.
Description	Descriptive sentence for the user.
Email	Email address of user.
Privileges	Privileges for the user. Possible bit values are: ○ 1 – Normal privileges ○ 2 – ADMIN ○ 4 – JOURNAL
app_data1	Optional application data that is passed back to the application during remote user validation.
App_data2	Optional application data that is passed back to the application during remote user validation.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.23. ds::getUserId

Get the specified user UXP ID.

Format:

```
ds::getUserId (db, session, username, idname, &flags,
               &description, &uxpid, &source)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user who owns the UXP ID.
Idname	Specifies the UXP ID to fetch.
Flags	Indicates type of UXP ID. Possible bit values: • 1 – ID is private to the owner. • 2 – ID is public and may be used by others. • 4 – ID is used for delegate resolution. • 8 – ID is for delegate service session authentication.

	To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
`Description`	Descriptive sentence for the ID.
`Uxpid`	Binary UXP ID of user
`source`	Source of the ID in XML format. Can only be viewed or edited by the owner of the UXP ID.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.24. ds::getUsers

Fetches user names.

Format:

```
ds::getUsers(db, session, filter, &list)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Filter`	Specifies an optional table of filters to apply to fetch operation. Possible filter columns: • **username** – User name filter. Supports SQL LIKE syntax. • **startrow** – Starting row number to fetch. • **maxrows** – Maximum number of rows to fetch. A filter is a list of value/pair entries.

	Example:
	
	`list mylist;`
	`ValuePair pair;`
	
	`pair.name="username";`
	`pair.value="Greg";`
	`appendList(mylist, pair);`

`List`	A list variable to receive fetched names.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user.*

6.25. ds::getUserIds

Fetches UXP IDs for a user.

Format:

```
ds::getUserIds(db, session, username, flags, &list)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Usernmae`	User who owns the UXP IDs.
`Flags`	Bitmask as a filter. Possible bit values: 1 – ID is private to the owner. 2 – ID is public and may be used by others. 4 – ID is used for delegate resolution. 8 – ID is for delegate service session authentication.
`List`	A list variable to receive fetched names.

L

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.26. ds::initDatabase

Initializes a new delegate service database.

Format:

```
ds::initDatabase (spec, id, username[ ,dbmod, dbparams])
```

Parameters:

`spec`	Specifies the database file specification. This will replace an existing database of the same name.
`id`	Specifies the UXP ID to be used as the system admin user.
`Username`	Specifies the user to be set up as the system admin user. The user must exist in the specified UXP ID.
`Dbmod`	Optional database module that implements an external database. Possible values: • UXP Use the native UXP SQL access module. • MySQL Use the MySQL database module. • Oracle Use the Oracle database module (Under construction). • SQLServer Use the SQL Server database module (Windows-only) (Under construction). Default: UXP

`dbparams`	Optional list of database parameters for external database module. Parameters are a string separated by a '|' character. The parameters must be specified in the correct order based on the database module:
	• UXP ignores the parameters.
	• MySQL uses the following parameters:
	  ○ Database host name.
	  ○ Database port number.
	  ○ Database user password for user uxpds.
	• Oracle Not implemented.
	• SQLServer Not implemented.

Returns:

True if successful.

6.27. dl::newDelegate

Creates a new delegate.

Format:

```
dl::newDelegate(db, session, owner, name, &description,
                &expiration, &permit_offline,
                &offline_duration, &access_max, &flags)
```

Parameters:

db	Specifies the database handle as returned by ds::openDatabase.
Session	Specifies the Services session handle as returned by ds::openSession.
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
description	A sentence describing the delegate.
Expiration	Date at which subscription expires. A zero indicates no expiration. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A value of 1 permits offline delegate resolution for the user.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.
Flags	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.28. ds::newUser

Creates a new user.

Format:

```
ds::newUser(db, session, username, formalname,
            description, email, privileges.
            App_data1, app_data2)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user name.
formalname	Formal name of user.
Description	Descriptive sentence for the user.
Email	Email address of user.
Privileges	Privileges for the user. Possible bit values are: ○ 1 – Normal privileges ○ 2 – ADMIN ○ 4 – JOURNAL
app_data1	Optional application data that is passed back to the application during remote user validation.
App_data2	Optional application data that is passed back to the application during remote user validation.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user.*

6.29. ds::newUserId

Creates a new UXP ID for a user.

Format:

```
ds::newUserId(db, session, username, idname, flags,
              description, uxpid, source)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user name.
idname	Specifies the new UXP ID name.
flags	Indicates type of UXP ID. Possible bit values: • 1 – ID is private to the owner. • 2 – ID is public and may be used by others. • 4 – ID is used for delegate resolution. • 8 – ID is for delegate service session authentication. To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
Description	Descriptive sentence for the ID.
Uxpid	Binary UXP ID of user
source	Source of the ID in XML format. Can only be viewed or edited by the owner of the UXP ID.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified ID owner is not the same as the current user.*

6.30. ds::openDatabase

Opens a delegate service database.

Format:

```
ds::openDatabase (spec [, readonly])
```

Parameters:

spec	Specifies the database file specification. This will replace an existing database of the same name.
readonly	A value of true indicates delegate services database will be open for reading only.

Returns:

Database handle.

6.31. ds::openSession

Opens a newServices session.

Format:

```
ds::openSession (db, username [, stop_time, usecb])
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Username	Userame to use for authentication. Authentication must be performed using the `ds::authenticate` function.
Stop_time	Sets the timeout option. Possible values: 0 – Session will timeout based on the UXP ID idle time settings. 1 – Disables timeouts. N – Stops the session at this absolute time. Number of milliseconds since Epoch. Default: 0

usecb	Optional boolean to indicate the default script callback should be used to prompt the user. If false or not provided, the caller must manually process challenges and responses.

Returns:

Session handle.

6.32. ds::publishUserId

Publishes a UXP ID for a user.

Format:

```
ds::publishUserId(db, session, username, idname, save
                  [,options])
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
Username	Specifies the user name.
idname	Specifies the UXP ID name.
save	If true, the new iic will be saved in the UXP ID within the delegate database.
options	Set of options separated by the '|' character. Possible values are: • V2ID Indicates the new ID will use the version 2 architecture. • COMPRESS Reduces the size of a version 2 ID, but also slightly reduces performance when opening a session or creating a UXP. By default, the version 1 ID architecture is used.

Returns:

Buffer containing UXP ID in iic format.

Notes: *This operation requires ADMIN privileges for the current user if the specified ID owner is not the same as the current user.*

6.33. dl::sendDelegateID

Sends a delegate ID to a registered user.

Format:

```
dsl:sendDelegateID(db, session, owner, name, username)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase`.
`Session`	Specifies the Services session handle as returned by `ds::openSession`.
`Owner`	Specifies the delegate owner name.
`name`	Specifies the delegate name.
`username`	Specifies the registered user to receive the delegate ID.;

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.34. dl::sendDelegateIDToAddress

Sends a delegate ID to an email address.

Format:

```
dl::sendDelegateIDToAddress(db, session, owner, name,
address)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Owner`	Specifies the delegate owner name.
`name`	Specifies the delegate name.
`address`	Specifies the email address to receive the delegate ID.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.35. ds::setDatabaseParameters

Sets the external database parameters. Parameters are specific to the type of external database.

Format:

```
ds::setDatabaseparameters(db, session, parameters)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`

`Parameters`	List of database parameters for external database module. Parameters are a string separated by a '|' character. The parameters must be specified in the correct order based on the database module:
	• UXP ignores the parameters.
	• MySQL uses the following parameters:
	  ○ Database host name.
	  ○ Database port number.
	  ○ Database user password for user uxpds.
	• Oracle Not implemented.
	• SQLServer Not implemented.

Returns:

True if successful.

Notes: *This operation requires SYSADMIN privileges for the current user.*

6.36. ds::setResponses

Sets the current set of challenge responses for a **Services** authentication session.

Format:

```
ds::setResponses(db, session, responses)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
resposes	Packed list of responses for the current authentication session. The format of the response list is: **Name1|Prompt1|Response1||Name2|Prompt2|Response2 …** Where: • Name is the challenge name. • Prompt is the challenge prompt. Response is the user-supplied response value.

Returns:

True if successful.

6.37. ds::setServer

Sets the URL for the current server. The URL must be set before creating delegates. If the server URL changes, current delegates will be regenerated. Any delegate already in use by a UXP will no longer permit remote validation.

To enable a private test server, set the URL to `file:///localhost`. This will call the delegate service via a local dynamic library instead of using the HTTP protocol.

Format:

```
ds::setServer(db, session, server_url)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
Server_url	Specifies the URL for the current server.

Returns:

True if successful.

Notes: *This operation requires SYSADMIN privileges for the current user.*

6.38. ds::setAccessCode

Sets the user access code for the current server. The access code is used to protect a message when using the WEB interface.

Format:

```
ds::setAccessCode(db, session, code)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
code	Specifies the new user access code for the current server. Code must be 32 characters long. If longer than 32 characters, the code will be truncated. If shorter than 32 characters, the code will be padded with $ characters.

Returns:

True if successful.

Notes: This operation requires SYSADMIN privileges for the current user.

6.39. ds::setSystemFlags

Sets the system flags for the current server.

Format:

```
ds::setSystemFlags(db, session, flags)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
Flags	Specifies the system flags for the current server. Flags are a bitmask with the following supported values: 1 – Record all data updates in the journal. 2 – Record delegate lookup in the journal. 4 – Record session authentication in the journal.

Returns:

True if successful.

Notes: *This operation requires SYSADMIN privileges for the current user.*

6.40. dl::subscribe

Subscribes to the specified delegate.

Format:

```
dl::subscribe(db, session, owner, name, username,
              expiration, permit_offline,
              offline_duration, access_max,
              access_count)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`

Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
username	User who to be subscribed to the delegate membership.
Expiration	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A value of 1 permits offline delegate resolution for the user. A -1 indicates use the value from the delegate.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution. A -1 indicates use the value from the delegate.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Sets the successful access count.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.41. dl::unsubscribe

Unsubscribes from the specified delegate.

Format:

```
dl::unsubscribe(db, session, owner, name, username)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase`.
Session	Specifies the Services session handle as returned by `ds::openSession`.
Owner	Specifies the delegate owner name.

`name`	Specifies the delegate name.
`username`	User who is subscribed to the delegate membership.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.42. dl::unsubscribeAll

Unsubscribes a user from all delegates.

Format:

```
dl::unsubscribe(db, session, username)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`Username`	User who is subscribed to the delegate membership.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user.*

6.43. dl::updateDelegate

Updates a delegate.

Format:

```
dl::updateDelegate(db, session, owner, name, description,
                   expiration, permit_offline,
                   offline_duration, access_max, flags)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
description	A sentence describing the delegate.
Expiration	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A value of 1 permits offline delegate resolution for the user.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit.
Flags	Flags to control delegate visibility and behavior. Possible values: 1 – Delegate can be fetched by anonymous user

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.44. dl::updateSubscription

Updates a user's delegate subscription.

Format:

```
dl::updateSubscription(db, session, owner, name, member,
                       expiration, permit_offline,
                       offline_duration, access_max,
                       access_count)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession`.
Owner	Specifies the delegate owner name.
name	Specifies the delegate name.
member	Specifies the user name.
expiration	Date at which subscription expires. A zero indicates no expiration. A -1 indicates use the value from the delegate. If the date is a positive integer, it is interpreted as the number of milliseconds since epoch. If the date is a string, it must be in the ISO date format **YYYY-MM-DDTHH:mm:ss**.
Permit_offline	A value of 1 permits offline delegate resolution for the user. A -1 indicates use the value from the delegate.
Offline_duration	Number of hours that offline access is permitted. Offline access with expire based on the number of hours since last online delegate resolution. A -1 indicates use the value from the delegate.
Access_max	Maximum number of successful accesses for the subscriber. A value of zero indicates no limit. A value of -1 indicates use the value from the delegate.
Access_count	Sets the successful access count.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.45. ds::updateUser

Updates a user.

Format:

```
ds::updateUser(db, session, username,formalname,
               description, email, privilege.
               app_data1, app_data2)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
Session	Specifies the Services session handle as returned by `ds::openSession.`
Username	Specifies the user name.
formalname	Formal name of user.
Description	Descriptive sentence for the user.
Email	Email address of user.
Privileges	Privileges for the user. Possible bit values are: ○ 1 – Normal privileges ○ 2 – ADMIN ○ 4 – JOURNAL
app_data1	Optional application data that is passed back to the application during remote user validation.
app_data2	Optional application data that is passed back to the application during remote user validation.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified user is not the same as the current user.*

6.46. ds::updateUserId

Updates a UXP ID for a user. Only specified columns will be updated in the UXP ID record.

Format:

```
ds::updateUserId(db, session, username, idname, flags,
                 description, uxpid, source)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`
session	Specifies the Services session handle as returned by `ds::openSession.`
username	Specifies the user name.
idname	Specifies the UXP ID name.
flags	Indicates type of UXP ID. Possible bit values: • 1 – ID is private to the owner. • 2 – ID is public and may be used by others. • 4 – ID is used for delegate resolution. • 8 – ID is for delegate service session authentication. To subscribe to a delegate, a user must have a UXP ID with the flag set for delegate resolution. Additionally, at least one UXP ID must existing having the session flag set; otherwise, a user will not be able to manage their user settings on the delegate service.
description	Descriptive sentence for the ID.
uxpid	Binary UXP ID of user
source	Source of the ID in XML format. Can only be viewed or edited by the owner of the UXP ID.

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified ID owner is not the same as the current user.*

6.47. dl::verifyDelegateID

Verifies a delegate ID with the current server.

Format:

```
dl::verifyDelegateID(db, session, owner, name, id)
```

Parameters:

db	Specifies the database handle as returned by `ds::openDatabase.`

`session`	Specifies the Services session handle as returned by `ds::openSession.`
`owner`	Specifies the delegate owner name.
`name`	Specifies the delegate name.
`id`	A buffer containing the delegate ID to verify.

Returns:

True if valid and verified.

Notes: *This operation requires ADMIN privileges for the current user if the specified delegate owner is not the same as the current user.*

6.48. event::countEvents

Counts UXP event records based on an optional filter.

Format:

```
event::countEvents(db, session, owner, filter)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`owner`	Specifies the UXP owner name.

`Filter`	Specifies an optional table of filters to apply to fetch operation. Possible filter columns:
	• **action** – Action keyword for event entry.
	• **uxp_name** – UXP name that recorded event.
	• **uxp_file** – File specification of UXP.
	• **event_date_low** – Lowest event date to include.
	• **event_date_high** – Highest event date to include.
	A filter is a list of value/pair entries. Dates are strings in ISO format.
	
	Example:
	
	`list mylist;`
	`ValuePair pair;`

	`pair.name="action";`
	`pair.value="New UXP";`
	`appendList(mylist, pair);`

`list`	A list variable to receive fetched rows. Each row is an event record in XML format.

Returns:

Number of matching event records.

Notes: *This operation requires ADMIN privileges for the current user if the specified UXP owner is not the same as the current user.*

6.49. event::deleteEvents

Deletes UXP event records based on an optional filter.

Format:

```
event::deleteEvents(db, session, owner, filter)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`Session`	Specifies the Services session handle as returned by `ds::openSession.`
`owner`	Specifies the UXP owner name.

`Filter`	Specifies an optional table of filters to apply to fetch operation. Possible filter columns:
	• **action** – Action keyword for event entry.
	• **uxp_name** – UXP name that recorded event.
	• **uxp_file** – File specification of UXP.
	• **event_date_low** – Lowest event date to include.
	• **event_date_high** – Highest event date to include.
	A filter is a list of value/pair entries. Dates are strings in ISO format.
	
	Example:
	
	`list mylist;`
	`ValuePair pair;`

	`pair.name="action";`
	`pair.value="New UXP";`
	`appendList(mylist, pair);`

`list`	A list variable to receive fetched rows. Each row is an event record in XML format.

Returns:

True if successfule.

Notes: This operation requires ADMIN privileges for the current user if the specified UXP owner is not the same as the current user.

6.50. event::getEvents

Gets UXP event records based on an optional filter.

Format:

```
event::getEvents(db, session, &outlist, owner, filter)
```

Parameters:

`db`	Specifies the database handle as returned by `ds::openDatabase.`
`session`	Specifies the Services session handle as returned by `ds::openSession.`
`outlist`	A list variable to receive fetched rows. Each row is an event record in XML format.
`owner`	Specifies the UXP owner name.
`filter`	Specifies an optional table of filters to apply to fetch operation. Possible filter columns: • **action** – Action keyword for event entry. • **uxp_name** – UXP name that recorded event. • **uxp_file** – File specification of UXP. • **event_date_low** – Lowest event date to include. • **event_date_high** – Highest event date to include. • **start_row** – Starting row number after filters applied. • **count** – Maximum number of rows to return. A filter is a list of value/pair entries. Dates are strings in ISO format.

	Example:
	
	`list mylist;`
	`ValuePair pair;`
	
	`pair.name=”action”;`
	`pair.value=”New UXP”;`
	`appendList(mylist, pair);`

Returns:

True if successful.

Notes: *This operation requires ADMIN privileges for the current user if the specified UXP owner is not the same as the current user.*

7. Error Codes

The following table lists possible error codes returned by the server API:

Table 7, Error Codes

Code	Description
1	The requested operation was successful.
2	The user must respond to authentication challenges.
3	Access has been denied.
800	A general error has occurred. The corresponding text will describe the general error.
801	The delegate service database is invalid.
802	No privilege for attempted operation.
803	The delegate service database must be opened read/write.
804	The current session is invalid.
805	The specified user was not found.
806	The specified delegate was not found.
807	The user already exists.
808	The delegate already exists.
809	Cannot delete self.
810	The specified user is not a subscriber to the delegate.
811	The specified user is already a subscriber to the delegate.
812	The specified user does not match a user within the UXP ID.
813	The specified journal filter is invalid.
814	The UXP ID already exists.
815	The UXP ID was not found.
816	The data item was not found.
817	The user must have a valid UXP ID.
818	The specified event filter is invalid.
819	The specified user filter is invalid.
820	The specified subscriber filter is invalid.