Sertainty

UXP Developer Guide

Version: V3.6.0

Table of Contents

1 UXP Engine

The **Sertainty UXP Technology** implements the Unbreakable Exchange Protocol (**UXP**) to provide a methodology and means by which one can protect and control access to private data. Unlike existing approaches, the **UXP Engine** embeds active technology within a protected entity to prevent unauthorized access no matter where the document resides. The **UXP Engine**, as part of the **UXP Technology** libraries, is external to the **UXP Object**.

1.1 UXP Identity Data

UXP Identity (UXP ID) Definition consists of several components to form a unique digital fingerprint of the client. The **UXP ID Definition** can represent a human, a machine or a process.

UXP ID Definition consists of the following attributes:

- **User Definition**
  The User Definition consists of a unique identifier and private **UXP Identity artifacts**. Though an account must be created and managed by a person, it can represent inanimate objects such as web sites, documents, devices or even locations.

  A **UXP Object** must contain at least one active **User Definition**; otherwise, the **UXP Object** will be permanently locked. The **User Definition** can be presented at **UXP Object** creation time via native **KCL Code** or via a **UXP ID Definition**.

  UXP ID-based User Definitions are created and stored as a **Special Purpose UXP Object** called a **UXP Identity**. The username can be a simple name or a validated email address. With a simple valid email, the user has gained implied trust. To achieve certified trust, the user and email address must be validated and confirmed by a third-party entity, such as a government agency. Trusted validation is not yet implemented.

- **Environment**
  When a User is authenticated, the system automatically acquires a configuration from the client application containing location and device. That data is transformed into the current configuration for the User and is compared to known configurations.

  A configuration formally contains a configuration header, network, location and device data. A configuration can be acquired by the Sertainty system or one can fabricate a configuration using the **ID Definition XML** template.

  Note: a manually designed configuration must still match an automatically acquired configuration at authentication time.

1.2 User Data

User data can be any block of data. The **UXP Engine** accepts serializable objects, string data, binary data and files to produce a protected entity within the **UXP Object**. To the **UXP Engine**, data is always treated as a series of unsigned bytes that can contain any value.

The **UXP Engine** can protect a single data artifact or many artifacts concurrently. In fact, the **UXP Engine** has a built-in virtual file system that will permit a complete file hierarchy to be protected within a single instance of a **UXP Object**.

1.3 KCL Code Module

Every **UXP Object** a contains **KCL Code** executable. The **KCL Code** is a proprietary program that is used to control access and policies on behalf of the data owner. The **KCL Code** can be constructed two ways:

- **Custom KCL**
  KCL is a proprietary C-like language that allows for a flexible way of constructing an engine. It does, however, require skills in programming and can be difficult to implement. A benefit of using custom **KCL Code** is that a designer can implement decisions that are unique to the implementation.

  See **Sertainty KCL Guide** for more information.

- **UXP ID**
  The preferred and easiest method for constructing **KCL Code** is by way of the **UXP Identity**. When using the **UXP Identity**, a pre-designed **KCL Code** is constructed using various identity and policy artifacts from the **UXP Identity**. The benefit is that the user can use the power of **KCL** without coding the **KCL Code** module.

1.3.1 KCL Code Module Control using a UXP ID

An alternative to manual **KCL Code** development is the **UXP ID**. A **UXP Identity** can generate the necessary **KCL Code** components and allow a user to develop a complex identity and governance without any manual coding.

The **UXP Identity** is actually a self-protecting Special Purpose **UXP Object** that contains pre-built **KCL Code** . It requires no programming skills and makes it very easy to build **UXP objects**.

A **UXP Identity** can be created by several methods:

- From an **ID Definition XML** template that sets pre-defined policy attributes and user access.
- From the *Sertainty Assistant, ID Editor*. The *ID Editor* allows the user to design a **UXP Identity** via a graphical user interface.

See the **Sertainty Workflow Guide** for detailed information on **UXP Identity** construction and usage.

Note: Sertainty recommends using the **UXP Identity** method for setting up **UXP Objects**.

1.4 UXP Object

The **UXP Object** consists of several component layers:

- **Virtual Header**

  Every **UXP Object** contains a header that allows the **UXP Engine** to identify the **UXP Object** as valid entity. If a header cannot be found or read, then the object is not considered a valid **UXP Object**.

  The header is uniquely identified by a domain, which consists of multiple tokens. By default, the **UXP Engine** will provide a default set of tokens, which means any **UXP Object** created will be recognized by all other installations of **UXP Technology**.

  If a particular installation desires a private **UXP Object** format, a domain can be defined and used when creating and accessing Objects. If the correct domain information is not provided when opening the **UXP Object**, the **UXP Engine** will not be able to access the virtual header information and will terminate the operation.

  A domain has two forms:

  - **Dynamic**

    A dynamic domain is private to the user of the technology. The domain information, consisting of a two-part key, must be managed and protected by the user. Before a **UXP Object** can be opened or created, the user must set the domain information.

  - **Licensed**

    A licensed domain is embedded in the license and must be set up by Sertainty. Since the license manages the domain, the user only has to indicate the desired domain when creating a **UXP Object**. To open a **UXP Object** that used a licensed domain, the **UXP Engine** will automatically identify the license domain and use that information to access the **UXP Object**.

- **UXP Object Metadata (Metadata)**

  The **Metadata** contains all the virtual file system data. All other user-specific data is stored as virtual data files.

- **Virtual File System**

  The virtual file system stores external data objects as files within the **UXP Object**.

  Private cloaking keys protect a virtual file.

  The following virtual files are supported:

  - **Directory**

    A directory can contain other virtual files as well as virtual sub-directories. This feature may require additional license.

- **Signature (TBD)**

  A signature file is an X509 certificate that can be used for document signing within the **UXP Object**. This is not yet implemented.

- **User data**

  User data can be any user-specified object. The virtual file can originate from either an existing file or an in-memory buffer. It can contain anything that can normally go into a conventional file and is only limited in size by the operation system itself.

- **KCL Code**

  The core of every **UXP Object** is a small p-code executable that operates very similar to a conventional computer. The **KCL Code** is fully self-contained and is machine independent; meaning that it can run the same compiled **KCL Code** on all supported platforms without modification or re-compilation. The **KCL Code** is discussed in more depth in the **Sertainty KCL Guide**.

  The primary role of the **KCL Code** is to provide user definition and policy decisions. When a user attempts to authenticate, the **KCL Code** must evaluate all relevant information as provided by the data owner as well as environmental information collected by the system. If the **KCL Code** denies access, the potential user will never access the protected data within the **UXP Object**.

1.5 Multi-User Access

A **UXP Object** can be opened in four modes:

- **Share None**

  This mode locks the **UXP Object** for exclusive access. Though faster because less locking is performed, a second user must wait until the **UXP Object** is closed before attempting to open it.

- **Share All**

  This mode is the most flexible method of opening a **UXP Object**. It permits multiple concurrent accesses; however, overall performance is slightly less than **Share None**.

- **Share Anon**

  This mode permits any user to write a new virtual file to the **UXP Object**. The **UXP Object** must be open, but not authenticated. Reads and updates are not permitted while in this mode.

- **Share Read Only**

  This mode is useful for high performance shared read access. No locks are placed on the file, however; it's up to the **UXP Object** to permit this. In some situations, the operating environment may force read-only mode.

  The following restrictions exist when a **UXP Object** is open for read-only operations:

  - Local event logging is disabled. Object activity will not be recorded within the **UXP Object.**

- Device and location changes will not be saved, meaning that the system will always consider unknown devices and locations as untrusted. Normally, a **UXP Object** may attempt to remember where it has been authenticated. Future attempts to authenticate may benefit from the knowledge of a prior access.
- **UXP Object** statistics will not be gathered
- All changes to the **UXP Object** will be disabled, including adding, deleting and altering files.

During authentication, the owner can check the **KCL Code variable Read Only** to determine the status of the **UXP Object**. If the **UXP Object** is physically read-only, the authentication can reject the connection and the user will not be permitted to open the **UXP Object**. If the owner does not have a problem with the read-only restrictions, the read-only status can be ignored, and the user will be permitted to open the **UXP Object** upon successful authentication.

When the **UXP Object** is an in-memory buffer, the only available mode is **Share None**.

1.6 Log Files

The **UXP Engine** attempts to create a log file when logging is enabled by the **UXP Object**-aware application. Logging is used to record various activities within the environment as a record. It is nothing more than a ledger or journal. The file will be named using a unique time code and can be found at the following location:

{Sertainty-install-folder}/log

If an alternate location is preferred, set the system environment variable **UXPLOGPATH** to the path.

A new log file is created at startup time of the **UXP Engine**. If the logging level is zero, then no log file will be created for the specific application.

1.7 Virtual Files

All artifacts, whether it is private metadata or user data, are stored as virtual files within the **UXP Object**. When a virtual file is created within a **UXP Object,** the data is managed similar to a conventional file system. The system can maintain a hierarchy of directory structures, where each may contain virtual files and sub-directories. A virtual file is identified by its full directory path and name.

In a virtual file path, a directory separator is the forward slash. The top-most directory is **/**. For example, a simple virtual file name **TMP** at the top level would have a full path name of **/TMP**. The same virtual file within a sub-directory **MyDir** would have a full path name of **/MyDir/TMP**.

With directory support, it is possible to protect an entire directory hierarchy in a single **UXP Object**.

Virtual names must be unique within their current directory and must be at least one non-blank character in length.

A virtual file is very similar to a typical file on disk, except that it's encapsulated within the defined **UXP Object**. The virtual file, content-wise, is identical to its physical counterpart; however, several transparent permutations may occur while transferring data into the protected **UXP Object**:

- Data is divided into fixed-length pages that are indexed. The page size can be specified at import time or it can be defaulted. In both cases, the page size value is always aligned to the nearest power of two that is greater than or equal to the specified value. The default value is 2MB.
- Every virtual file can have a data cache associated with it. At creation time, the cache size can be set to any value between 0 and 20. A zero value will disable the cache. Each cache buffer is large enough to hold a virtual file page. The purpose of the cache is most valuable for sequential access of large data sets.

  For example, if a video file has a page size of 2MB, yet the user application is fetching 512 byte buffers, then the caching system will avoid a hard disk fetch by reading the requested data from an in-memory buffer. For random access, the caching provides little benefit.

- At creation time, the virtual file can optionally be compressed. Depending on the file contents, compression can reduce virtual file sizes up to 90%.
- A proprietary cloaking algorithm based on industry standard encryption and other cloaking techniques protects all data. In fact, because of the virtual file system, most data artifacts are protected multiple times via a recursive I/O system.
- Files can be hidden by prefixing the virtual file or directory name with a "." Character. Only users with owner privileges can access hidden files.

Like physical files, a virtual file can be accessed as a series of bytes. A calling routine can specify the starting location within the virtual file as well as the number of bytes. The **UXP Engine** will then locate the desired data block within the virtual file, decompress, if necessary and de-cloak only the data to be returned.

All unrequested data remains protected within the UXP Object.

Like similar file access routines in coding languages like C or C++, the **UXP Engine** supports full random access seeks and fetches. Further, the user application can access the data as if the data is clear. Translation of the user data to and from its protected state within the **UXP Object** is automatically managed.

When a virtual file is opened, the **UXP Engine** will call the rule **Authentication::fileAccess** to determine if the user can access the file. **UXP Object**-level privileges do not necessarily grant file access.

1.8 User Definitions

User Definitions represent the entities that can authenticate and access **UXP Object** content. By design, every **UXP Object** must have at least one **User Definition**, but can have many **User Definitions**.

Each **User Definition** consists of a public username and a set of challenge pairs, prompt / responses, that are private to that user (human or machine). Additionally, there may be policies associated with the **User Definition** such as schedules, acceptable location and hardware settings.

User Definition can be defined and managed by two methods:

- Custom **KCL Code** (see **Sertainty KCL Guide** for more detail)

- **UXP ID Definition XML**

Using either development method, the outcome is the same: **User Definitions** become a **KCL Code** executable that is accessed during authentication and data access operations.

Challenge Pairs, Prompt/Response, are a primary contributor to the security of the **UXP Object**. Similar to conventional username/password, the **UXP Engine** will use the data contained in the **KCL Code** to validate user access.

Where the **User Definition** becomes far superior to the conventional approach is its algorithm to randomize Challenge Prompt presentment and to assign relative trust. During an authentication attempt, the **KCL Code** will determine how trustworthy the user is and recommend a number of challenge prompts to be presented. A trust value of 100% may allow the user to authenticate with a very small number of challenge prompts, or even no challenge prompts, while an untrustworthy access may present three, four or more challenge prompts. In all cases, challenge prompts are randomly chosen, and any unacceptable responses are kept secret. This inhibits random guesses by the user.

As noted, Challenge, Prompt/Response, pairs are private to a user. When constructing the pairs as a human, Sertainty promotes a method of utilizing a person's life history as opposed to conventional known random passwords or hints that can be socially reverse engineered. It attempts to capture the person's true identity that is based on facts, not beliefs.

For a machine **UXP Identity**, the challenge pairs are auto generated as well randomly selecting machine specific attributes. These will make up the **UXP ID Definition** for a machine.

For users who are insert, such as applications or machines, the **User Definition** supports fully randomized prompt/response pairs that can be safely embedded within an application by way of **Sertainty Secure Strings**. This technique protects the source data on disk and in memory, preventing pattern matching by binary data scanners.

For all types of users, prompt/response pairs can either be optional or required. When a challenge pair is optional, it becomes part of the pool of challenges use for choosing random challenges to present during authentication. When a challenge is required, the **KCL Code** will force the challenge to be met for every authentication. Typically, a machine-based user would be design with all challenges being required.

1.8.1 Response Value Restrictions

A challenge response must adhere to the following restrictions:

- A value must be of minimum length. The default length is 3 characters but can be modified using the system preference **minChallengeLength**.
- A value cannot contain the prompt, regardless of the prompt and response case.
- A value cannot be repeated as a response value to other challenges more than 20% of the challenge total.

- When complex response is enabled, a value must contain at least 1 uppercase, 1 lowercase, 1 digit and 1 special character. Complex response values are enabled by the system preference **useComplexChallengeRules**.

1.8.2 Response Data Masking

The **UXP Engine** provides options for masking user input. By masking data, the owner of the **UXP Object** can require dynamic responses for any or every user challenge.

For example, the following challenge is issued:

Enter your code seven: XXXXXX

where **XXXXXX** is the stored response. Without masking, the value provided by the user must exactly match **XXXXXX** every time the user is challenged in this form. With masking, the user may be required to enter **XXXXXX** plus a dynamic time attribute, such as current day, current year, prior day, etc. If the challenge requires the masking and the mask value for the user is the following:

MaskLastYear,MaskUserData,MaskNextMonth

the required input would be **2011XXXXXX8** if the current date is July, 2012. Without changing the **UXP Object**, the response in January 2013 for the same challenge would be **2012XXXXXX2**.

Each masking code is represented as a **KCL** variable with a constant value. The mask can be defined for each challenge response as a list of masking codes separated by commas. When masking is enabled, the same masking must be applied to each challenge response, except the original username challenge.

The following masking codes are supported in the **KCL Code**:

Table 6 – Masking Codes

Code	Description
MaskAmPm	A constant containing the internal mask value. The mask value entered by the user must be either AM or PM.
MaskDay	A constant containing the internal mask value. The user must enter the day of the month.
MaskHour12	A constant containing the internal mask value. The user must enter the current hour in 12-hour format where midnight to 1AM is 0 and noon to 1PM is also 0.
MaskHour24	A constant containing the internal mask value. The user must enter the current hour in 24-hour format where midnight to 1AM s 0 and 11PM to midnight is 23.
MaskLastMonth	A constant containing the internal mask value. A constant containing the internal mask value. The user must enter the month number for last month where January is month 1 and December is month 12.
MaskLastYear	A constant containing the internal mask value. The user must enter the four-digit year for last year.
MaskMonth	A constant containing the internal mask value. The user must enter the current month number where January is month 1 and December is month 12.
MaskNextMonth	A constant containing the internal mask value. The user must enter the month number for next month where January is month 1 and December is month 12.
MaskNextYear	A constant containing the internal mask value. The user must enter the four-digit year for next year.

MaskToday	A constant containing the internal mask value. The user must enter the day of the month.
MaskTomorrow	A constant containing the internal mask value. The user must enter the day of the month for tomorrow. If today is the last day of the month, then tomorrow will be the first day of the next month.
MaskUserData	A constant containing the internal mask value. The user must enter the actual challenge value.
MaskYear	A constant containing the internal mask value. The user must enter the current four-digit year.
MaskYesterday	A constant containing the internal mask value. The user must enter the day of the month for yesterday. If today is the first day of the month, then yesterday will be the last day of last month.

1.8.3 Authenticating a UXP Object

The UXP Engine implements the following workflow for a user:

1. Retrieve the username. A User Definition can be found only if the User provides a valid username.
2. If the current User Definition has a schedule defined, check to make sure the current time falls within the specified schedule window.
3. Check the current device, network and location configuration.
4. Check any responses provided by the User.
5. Repeat these steps, if necessary.

The **UXP Engine** continues to follow the above steps until either the user is denied access with a status of **StatusNotAuthorized** or granted access with a status of **StatusAuthorized**.

The status values returned by any of the above four steps are always returned to the **KCL Code** Authentication**::main** routine. Within that routine, a call to **validateUser** is made, which performs the four steps. The indicated status is returned via this call.

The **Authentication::main** procedure is ultimately responsible for setting the final status. It alone decides if the user is permitted access, denied or challenged for more identity proof. The routine is given access to failure counts, access counts, the current working user definition and various tools to notify users of failures and successes. If the procedure returns the status **StatusNotAuthorized**, the system exits and denies access to the protected entity. If the procedure returns the status **StatusAuthorized**, access is granted with the designated privileges. All other status values trigger a user challenge for additional proof of identity.

1.8.4 Multi-factor Authentication

Multi-factor authentication is implemented using external factors and approvals.

External factors are similar to simple challenge pairs except that they are sent via E-mail or SMS to a target address. When the User is prompted for the external factor, the User must have access to the external factor recipient address or be in direct contact with the recipient.

If more than one external factor is issued, the User must get **all** external responses to obtain access authorization. Alternatively, an owner may send the same external response to multiple external recipients. In this scenario, the user must gain access to the response code from at least one of the recipients.

Approvals are similar to external factors but indicate an approval response is necessary to gain authorization for a third party. The validation rules are the same as external factors.

To indicate multiple recipients for an external factor or approval, enter the recipient addresses separated by a semi-colon character.

Example:

challenge.address = foo@bar.com;foo2@bar.com

challenge.address = 4233335567;4442225678;5535553456;4448885555@vnet.com

1.9 Events

The owner of the **UXP Object** controls all event logging. Logging will only occur if the correct event options are set when the **UXP Object** is opened.

The following table determines when an event entry is recorded:

Table 1 – KCL Event Option Keywords

Event Option	Description
EventAccess	Indicates an event entry will be recorded upon any access.
EventEmail	Indicates an event entry will be sent to the email address as specified in variable **EventEmailAddress**. Event data is sent in clear form.
EventEmailAddress	Specifies the email address to use when **EventEmail** option is selected.
EventExternal	Indicates an event entry will be recorded by calling the applications external event callback. The callback function may read event data using the structured API only if the caller knows the event key as set by the owner. If the callback function is invalid, the event recorder will automatically attempt to record the event within the **UXP Object**.
EventFailure	Indicates an event entry will be recorded for any failed authentication attempt.
EventFile	Indicates an event entry will be recorded in a local file. **EventFileSpec** must contain the file specification. If the **EventFileSpec** ends with **.uxp**, then the event will be recorded in an existing UXP as an anonymous write. If the **EventFileSpec** is the word **console:**, then the event will be recorded to stdout.
EventFileSpec	Specifies the local file specification or a trusted server URL. Additionally, the keyword console: can be used as the output file. Event data will be sent to the current console standard output device.

Event Option	Description
	Example: $(HOME)/$(UXP_FILENAME).log would become /home/myhome/foo.log
EventFtp	Indicates an event entry will be recorded at a remote FTP site (Not yet implemented). **EventFtpURL** must contain the server URL.
EventFtpURL	Indicates the remote ftp server URL.
EventLocal	Indicates an event entry will be recorded within the **UXP Object**. An event recorded locally is protected by the UXP. Only the owner and users with **ReadEvents** privilege may read event data. Event data is immutable. It cannot be changed or deleted by anyone.
EventMessages	Indicates an event entry will be recorded for any external E-mail and SMS messages that are sent.
EventRemote	Indicates an event entry will be recorded at a remote UXP entity http server (Not yet been implemented). **EventURL** must contain the server URL.
EventRepeats	Indicates an event entry will be recorded for repeated failed authentication attempts.
EventSecurelKey	Specifies a key that must be used by an external callback to read event data. This option is used in conjunction with the **EventExternal** option.
EventSMS	Indicates an event entry will be sent to the SMS phone number or address as specified in the variable **EventSmsAddress**. Event data is sent in clear form.
EventSmsAddress	Specifies an email address that will be use to deliver an SMS message.
EventURL	Specifies the remote http server URL.

All options are maintained as bit switches, so any combination of options can be set using the single **KCL** variable **EventOptions.**

For example, to set event logging for failures using the event callback, the following setting is required within the KCL **Authentication::setup** routine:

```
EventOptions = EventFailure | EventExternal | EventFile;

EventFileSpec = "$(UXP_FILENAME).log";
```

Event records critical information about the monitored activity. Each event record contains the following data:

Table 2 – Event Data

Item	Description
User	The user definition name of the current user.
Timestamp	The date and time of the entry, including time zone and UTC offset.

Status	A status value indicating success or failure.
Action	An action value that indicates the attempted operation.
Message	A text message describing the event entry.
Device	Device data from the user's computing device.
Location	Location data describing the network and location of the user, if available.

For external and remote event logging, the data owner can specify a security key. The key is then used to validate the event data before it is passed to the external agent. If no key is specified, the data is not protected.

1.10 Deferred Message Delivery

If a **UXP Object** triggers or initiates an external electronic message, such as an SMS message, an E-mail message or an FTP delivery, the **UXP Engine** may decide to queue up the attempted delivery. If the object can communicate with the Internet, delivery may be immediate. If the Internet is unavailable or is untrusted, the attempted delivery will be saved in its own **UXP Object** and will be delivered at the next trusted opportunity.

The actual queue data is stored in a protected file in the following location:

{Sertainty-install-folder}/data/msg

1.11 E-mail Services

An E-mail service must be defined from within the **KCL Code f**or each **UXP Object**. This means that each **UXP Object** can have a unique E-mail service to which alerts and challenges are relayed. The service settings must be defined when the **UXP Engine** calls the **KCL** routine **Authentication::setup**.

The following **KCL** variables are provided for E-mail setup:

Table 3 – E-mail KCL Variables

Variable	Description
EmailAuthentication	Specifies whether SMTP user authentication is performed. Possible values are: • False • True
EmailPort	Specifies the SMTP port number.
EmailPwd	Specifies the account password used to connect to the SMTP server.
EmailReplyTo	Specifies the sender's E-mail address. This will become the reply address when a message is sent. To add an optional name, use the following format for your address: name <email-address>

EmailSecurity	Specifies the SMTP security setting. Possible values are: • SSL • TSL • NONE
EmailServer	Specifies the SMTP server address used for E-mail relay.
EmailUser	Specifies the account used to connect to the SMTP server.

1.12 SMS Services

SMS delivery is performed using a mobile service provider's SMS via E-mail option. This requires the user to provide a phone number for the supported service in the form:

xxxxxxxxxxx@service.com

The actual delivery will utilize the same e-mail configuration as defined in E-mail Services.

Unlike an E-mail message, an SMS message contains a subset of data elements for challenges and alerts. If a **UXP Object** owner wishes full disclosure, it is recommended that an alert be send to a trusted E-mail client; otherwise, complete device and location data will not be sent.

1.13 Location Services

As part of authentication, the **UXP Engine** attempts to find the physical location of the object at creation and access time. To do that on non-cellular devices, a remote server must be accessed to get the correct IP address as well as other ancillary artifacts such as street address and estimate geophysical coordinates.

By default, a built-in web service callout is provided with the **UXP Engine.**

Table 4 – Location Tags

Item	Description
CityName	Returns the city name.
CountryCode	Returns the country abbreviation or code.
CountryName	Returns the country name.
IP	Returns the IP address. This is a required tag.
RegionCode	Returns the state or region abbreviation.
RegionName	Returns the state or region name.
StreetAddress	Returns an approximate street address.
Zipcode	Returns the approximate zip code.

For mobile devices with wireless support, **UXP Object** may determine a more accurate location than it does with a simple IP address. Similar to the way a mapping application works, **UXP Object** can determine the physical location to within 100 feet. Once the wireless triangulation has been established, the data is sent to a remote server where the location data is mapped to physical address.

Because a callout is dependent on Internet routing and performance, a location call can be rather time consuming. If your application prefers not to wait on location services, a preference is provided to control behavior. Using the preference **locationScanOption**, one can set the following values to control behavior:

Table 5 – Location Scan Options

Option	Description
uxpLocationScanNone	Disables location scanning,
uxpLocationScanLevel1	Performs the fastest scan by retrieving an approximate location.
uxpLocationScanLevel2	Performs a mid-level scan that will attempt to utilize wireless triangulation.
uxpLocationScanLevel3	Performs the most thorough and trusted location scan; however, a scan may take up to 10 seconds at application startup.

A preference is set using a static call within the **UXP Object** library.

1.14 Home Directory

The **UXP Object** runtime requires a home directory to operate. Within the directory, a distributed file called **UXP sertainty.rsf** must reside.

The directory name **home** will be located using the following search:

- Path found in the boot.ini file. Boot.ini is located in the same folder as the SertaintyCore shared library.
- Private storage path. (IOS and Android)
- UserDocumentsPath/Sertainty.
- The directory containing the running application binary.
- The current directory of the running application.
- In a location as defined in the system environment variable **UXPHOME.**
- UserHomePath/Sertainty.
- In a location as defined in a **UXP Object** preference **uxphome**.
- Shared path as defined at installation time.

1.15 Common Home Directory

For installations that wish to control client behavior, a common home is supported. The common home location will be searched first for the requested resource. If not found there, a normal search of the standard home directory will occur.

If a resource is found in common home, the resource will be copied to the standard home directory. This will keep the user's environment in sync, even if the user machine is not connected to a network.

To activate a common home for a user, the environment variable UXPCOMMONHOME must be set to a valid directory.

1.16 License Support

A licensing sub-system is built into the **UXP Object** core library. It is used as a mechanism to control access to the **UXP Object** library but can also be utilized by a third party to construct unbreakable license files for custom applications.

A license is a Special Purpose **UXP Object** that can only be opened by the **UXP Engine**. It is not meant to be accessed by the user as a simple protected data object. Like other **UXP Objects**, it is validated when opened and can protect itself from illegal use.

The core **UXP Object** library requires a valid license for various activities. The license is loaded as part of the **initializeLibrary** call that must be made prior to any other call to the library backend.

When calling **initializeLibrary**, the caller must pass the name of the license file. A license file can be located in any directory, or the system will look in the valid **UXP Object** home directory. If the license file name is an asterisk, the system will search for sertainty.lic in the Sertainty home folder. If the file name is a complete specification, including the full volume specification, the system will attempt to find the license using that specification. If the license name is a relative path and name, then it will prepend the Sertainty home folder and attempt to search for the license.

Licensing not only controls access to the executable library, but also to **UXP Objects**. For example, a **UXP Object** owner specifies the authenticated user can write the **UXP Object**; however, the current license only permits reading. The license privilege will always override a **UXP Object** privilege if the **UXP Object** privilege is greater than the license privilege.

1.17 File specification Tokens

When passing file specification into the Sertainty UXP system, supported substitution tokens may be used to permit evaluation at runtime. For example, when writing scripts that may be ported to other physical machines, one should use the token $(HOME) to represent the current user's home folder. This would permit shared scripts, even across operating system platforms.

> **Example**: $(HOME)/myid.iic would be replaced with /Users/Greg/myid.iic, where /Users/Tom may be the home folder for a user on a MacOS system.

Table 7 – Supported Tokens

Name	Description
$(APPDATA)	Replaces the token with current user's application data path.
$(COMMONHOME)	Replaces the token with the Sertainty common home path.

Name	Description
$(DATE)	Replaces the token with the current date using the format YYYYMMDD.
$(DATETIME)	Replaces the token with the current date and time using the format YYYYMMDDhhmm.
$(DAY)	Replaces the token with the ISO day number. **Ex:** file$(DAY).txt would generate file01.txt when the current day of the month is the 1st.
$(DESKTOP)	Replaces the token with the user's desktop path.
$(DOCUMENTS)	Replaces the token with the user's documents path.
$(DOWNLOADS)	Replaces the token with the user's downloads path.
$(EXE)	Replaces the token with the path to the current Sertainty shared library.
$(HOME)	Replaces the token with the current home directory path.
$(LOG)	Replaces token with Sertainty log folder.
$(MONTH)	Replaces the token with the month number. **Ex:** file$(MONTH).txt would generate file07.txt if the month is July.
$(SAMPLE)	Replaces the token with the current path to the developer examples folder.
$(SEQUENCE)	Replaces the token with a unique identifier that makes the specification unique. If the filename portion of the path contains $(SEQUENCE), a unique timestamp and sequence will be included and cause a new file to be created for every automator event. If a single file is desired for all events within an automator, exclude the $(SEQUENCE) tag. **Ex:** mylog$(SEQUENCE).log
$(SERTAINTYHOME)	Replaces the token with the current Sertainty home directory path.
$(SHARED)	Replaces the token with the current shared Sertainty directory path.
$(TEMP)	Replaces the token with a temporary folder path.
$(UXP_FILENAME)	Replaces the token with the current **UXP Object** name. If there is no current **UXP**

Name	Description
	Object, the token will be replaced with the token unknown.
$(UXP_FOLDER)	Replaces the token with the current **UXP Object** location. If there is no current **UXP Object**, the token will be replaced with the current working folder.
$(WORKFLOWCONFIG)	Replaces the token with the workflow home data path.
$(WORKFLOWHOME)	Replaces the token with the active workflow configuration data path.
$(WORKFLOWHOST)	Replaces token with current workflow host folder.
$(YEAR)	Replaces the token with the year number. **Ex:** file$(YEAR).txt would generate file2018.txt

1.18 Preferences

Preferences are used to control component behavior. There are two levels of preferences: system and user preferences. System preferences affect all **UXP Object** users while user preferences are local to the current user.

Preferences are set and retrieved using function calls within the **Sertainty API**.

Preferences are located in the **InitPath** as defined in boot.ini. See the section on deployment for detailed information on the boot.ini file.

The naming convention is as follows:

User preferences:
sf_<platform>_<os_user>.ini
System preferences:
sf_<platform>.ini

Note: The INI files described here may not be compatible with generic INI parsers due to the use of the '\' character. Sertainty considers a '\' as an escape character. To embed a '\' as a literal value, the reverseslashes must be doubled, i.e. \\. The preferences processor will correctly interpret double '\' characters as a single '\'.

The following preferences are supported:

Table 8- System Preferences

Preference	Data Type	Description
agentInterval	**Integer**	When the **Agent** is running, specifies the number of seconds between attempted message deliveries. **Default:** 60
agentLogging	**Integer**	Sets the logging level for the **UXP Object Agent**. Possible values: **0. Logging is disabled** **1. Log only fatal errors** **2. Log severe errors** **3. Log all errors and warnings** **4. Log everything** **5. Log debugging information** **Default:** 0
agentEnvInterval	**Integer**	When the agent is running, specifies the number of minutes between hardware/network/location scan operations. **Default:** 15
agentLIcenseInterval	**Integer**	When the agent is running, specifies the number of minutes between license refresh operations. **Default:** 720
agentRefreshInterval	**Integer**	When the agent is running, specifies the number of seconds between refresh operations. **Default:** 600
CommonHome	**URL**	Specifies a network accessible location from which home **UXP Object** library data may be found. This is typically used to enforce corporate rules and application features. **Default:** Empty
disableStringEncoding	**Boolean**	Disables string encoding for external logs and other text-based files. Typically, this should only be set when troubleshooting Sertainty application behavior. **Default:** false
driveLogging	**Integer**	Sets the logging level for a drive-mounted **UXP Object**. Possible values: **0. Logging is disabled** **1. Log only fatal errors** **2. Log severe errors** **3. Log all errors and warnings** **4. Log everything** **5. Log debugging information** **Default:** 0

Preference	Data Type	Description
fileWatchInterval	**Integer**	For file watch operations, specifies the number of milliseconds between file change detection operations. To quickly detect file changes, set the value low. For less CPU consumption, but slower change detection, set the value higher. **Default**: 10000
installerLogging	**Integer**	Sets the logging level for the installer utility. Possible values: **0. Logging is disabled** **1. Log only fatal errors** **2. Log severe errors** **3. Log all errors and warnings** **4. Log everything** **5. Log debugging information** **Default:** 0
locationScanOption	**Integer**	Specifies the type of location scanning invoked by the **UXP Engine**. Possible values: **0. Scanning is disabled** **1. Use only Sertainty trusted server** **2. Use Sertainty trusted server plus other available GPS and location resources.** **Default:** 2
machineChallenges	**Integer**	Specifies the default number of challenges when generating a machine ID user. **Default:** 10
maxChallenges	**Integer**	Specifies the maximum number of challenges per user. The absolute maximum is 999. **Default:** 30
maxdeliveryattempts	**Integer**	When attempting to send a message, this specifies the maximum number of attempts before permanently discarding the message. **Default:** 1
maxSplits	**Integer**	Specifies the maximum number of splits that will be created when a UXP split operation is performed. **Default**: 100
minChallengeLength	**Integer**	Specifies the minimum length for a challenge name and value. **Default:** 3

Preference	Data Type	Description
minChallenges	**Integer**	Specifies the minimum number of challenges for an identity. Three is the absolute minimum. **Default:** 10
quickProtectLogging	**Integer**	Sets the logging level for a QuickProject operations. Possible values: **0. Logging is disabled** **1. Log only fatal errors** **2. Log severe errors** **3. Log all errors and warnings** **4. Log everything** **5. Log debugging information** **Default:** 0
scriptLogging	**Integer**	Sets the logging level for script utility operations. Possible values: **0. Logging is disabled** **1. Log only fatal errors** **2. Log severe errors** **3. Log all errors and warnings** **4. Log everything** **5. Log debugging information** **Default:** 0
smtpConnectionTimeout	**Integer**	When attempting to connect to an SMTP server on behalf of message delivery, this specifies the number of milliseconds to wait for a valid connection. **Default:** 3000
smtpResponseTimeout	**Integer**	When attempting to deliver a message, this specifies the number of milliseconds to wait for message delivery confirmation. **Default:** 20000
startAgent	**Boolean**	When True, the **UXP Object** core library will attempt to start an **Agent** if one is not already running. **Default:** True
taskWatchInterval	**Integer**	For workflow change task operations, specifies the number of milliseconds between file change detection operations. To quickly detect file changes, set the value low. For less CPU consumption, but slower change detection, set the value higher. **Default**: 10000
useComplexChallengeRules	**Boolean**	When True, challenge responses must contain at least 1 uppercase, 1 lowercase, 1 digit and 1 special character.

Preference	Data Type	Description
		Warning: When set, the response values become conventional passwords and minimize the strength of the Sertainty authentication model. **Default**: False
vfsStartClientDelay	**Integer**	Specifies the wait time in milliseconds when mounting a Drive device. **Default**: 2000

Table 1 – User Preferences

Preference	Data Type	Description
mountpath	**Directory**	When mounting a **UXP Object** as a Drive device, this specifies the mount point for the device. **Default:** ${Home}/Sertainty Drives
PreferenceBuffers	**Interger**	Specifies the default number of cache buffers for **UXP Object** virtual file when using **QuickProtec**t or **QuickMoun**t. **Default**: 0
PreferenceCompanyName	**String**	Specifies the company name when creating **UXP Objects** when using **QuickProtec**t. **Default:** Empty
PreferenceCompanyWebPage	**String**	Specifies the company web page when creating **UXP Objects** when using **QuickProtect**. Default: Empty
PreferenceDefaultDomain	**String**	Specifies the default license domain when using **QuickProtect**. **Default:** Public
PreferenceDefaultID	**String**	Specifies the default **UXP ID** to use when using **QuickProtect**.
PreferencePageSize	**Integer**	Specifies the default page size for a **UXP Object** virtual file when using **QuickProtec**t or **QuickMount**. **Default:** 0
PreferenceUseReadWriteArchitecture	**Integer**	When True, the default **UXP Object** architecture is read-write when using **QuickProtect** or **QuickMount**. **Default:** False
uxphome	**Directory**	Specifies the folder containing required data for the **UXP Object** libraries. Typically, this would not be used unless alternate folder will be used as the home folder. **Default**: Empty

Note: all preference names are case sensitive.

1.19 Environment Variables

The following system environment variables can be set to modify the **UXP** library behavior:

Table 10 – Environment Variables

Variable	Data Type	Description
UXPDATAPATH	**String**	Specifies an alternate folder at which cached data will be managed. If empty, the standard data location will be used. **Default:** Empty
UXPHOME	**String**	Specifies an alternate folder for the home location. If empty, the standard home location will be used. **Default:** Empty
UXPINITPATH	**String**	Specifies an alternate folder at **UXP** init files will be managed. If empty, the standard init location will be used. **Default:** Empty
UXPLOGPATH	**String**	Specifies an alternate folder at which log files will be written. If empty, the standard log file location will be used. **Default:** Empty

2 UXP Object Construction and Access

2.1 Building your Application

Presently, the supported interfaces are C, C++ and C#; mobile – Android and iOS . Other languages interface wrappers are planned for a future release of the tool kit.

After installing the toolkit, several folders will contain the necessary components to compile and link your applications.

The installation folder contains the following developer folders:

- **developer/bin**

  Contains the necessary files for the **UXP Technology**.

  Linux Libraries

  libboost_program_options.so
  libboost_atomic.so
  libboost_thread.so
  libboost_date_time.so
  libboost_filesystem.so
  libboost_system.so
  libSertaintyCore.so
  boot.ini

  MacOSX Libraries

  libSertaintyCore.2.dylib
  libosxfuse_i64.2.dylib
  boot.ini

Windows Libraries

ssleay32.dll

libeay32.dll

SertaintyCore2.dll

boot.ini

Note: All **UXP 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 Data Services Guide**

    The **Data Services Guide** describes a collection of service extensions to the **UXP Technology** object. The Data Services share metadata and setup; however, each service extension can be utilized independently.
  - **Sertainty Developer Guide**

    The Developer Guide contains a full description of the core **UXP Technology**.
  - **Sertainty KCL Guide**

    The **KCL Guide** describes the **KCL** environment and how to use it for custom **KCL** identities.
  - **Sertainty Workflow Guide**

    The **Workflow Guide** contains details focused on Workflow enablement using the Sertainty Agent, UXL Scripting and UXP Managed Identity creation
  - **Public API**

    The API guide is a reference guide for the public **UXP** 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.
    - C# language
- **developer/examples**

  Contains sample scripts and code.

Table 2 – Code samples

Sample	Description
Bytearrays.c, c++, c#, uxl	Demonstrates the use of uxl ByteArray which is a Sertainty Data Structure that is used to handle binary and ascii data. The sample demonstrates writing data to and reading data from the uxl ByteArray as well as loading contents of a file read from the file system into the uxl ByteArray -

	using C, C++ and C# language interfaces and **UXL** scripting language
1-Delegate_create_db.c, c++,c#, uxl	This sample demonstrates how to create a new Data Services Database using an existing ID that will be designated as the DB owner ID. The DataServices db stores and manages Delegate IDs, Users and Delegate subscribers.
2-Delegate_open_session.c, c++,c#, uxl	Demonstrates opening an existing Data Services (DS) Database. This sample allows you to open an existing DS Database and authenticate into it to create a DS Session.
3-Delegate_set_server.c, c++,c#, uxl	This sample sets the Delegate Server location in the DataServices database. The server URL will be linked and embedded in each auto-generated delegate user identity - the proxy identity. Once activated by a UXP Object containing the delegate identity, the URL will be contacted.
4-Delegate_add_users.c, c++,c#, uxl	Demonstrate adding a new user to Data Services (DS) Database. A user is only added if it doesn't already exist in the DB.
5-Delegate_add_delegates.c, c++,c#, uxl	This demonstrates within the Data Services database: (1) creating a new Delegate Identity (2) creating new User2 (3) adding User2 as a Subscriber to the Delegate ID subscription list The Delegate Identity, new User 2 and Subscriber are only added if it doesn't already exist in the Data Services db.
6-delegate_get_info.c, c++, c#	Displays detailed information from the Data Services Database as follows: (1) Delegate IDs (2) Delegate Subscribers (from User IDs in the Data Services db) (3) User IDs A Delegate ID is a proxy ID with a subscription group in Data Services db. Users in the Data Services db can be added as Delegate Subscribers to the Subscription List for the Delegate ID. A User can an owner of a Delegate ID. A Delegate owner (User ID) is not required to be a subscriber to a Delegate ID. All User IDs are listed in the Data Services Database.
Helloworld.c, c++, c#, ..uxl	Demonstrates the initialize library call necessary before any UXP ID or UXP functions can be called - using C, C++ and C# language interfaces; and uxl scripting

Id_from_xml.c, c++, c#. .uxl	This sample demonstrates constructing a UXP Identity file (*.iic) from an ID Definition XML Source file (*.xml). The generated UXP Identity file (*.iic) can be used to generate a UXP Object.
Id_session.c, c++, c#, .uxl	This sample demonstrates how to authenticate into an ID session and use it to auto-authenticate, also known as, Single Sign-On, into a UXP Object. A UXP Object requires authentication of the prospective user (process or person). ßAccess will only be granted after a successful authentication. Having to individually authenticate into multiple UXP Objects is time-consuming. Also time-consuming is authenticating into a single UXP Object multiple times. As a convenience, a UXP Identity can be used for a Single Sign-On session. A Single Sign-On session allows automatic authentication into UXP Objects that were created using that same UXP Identity. For authentication, there are two approaches to programmatically seeking authorization. The first approach is to declare a function callback that is called when the system presents challenges. The callback function is given the list of challenges that it must process and return. The callback function is called until resolution is reached. The second uses a looping process to allow the program to handle the challenge list manually. Responses are then given back to the system and the authentication loop continues until a resolution is reached.
MappingsCAPI.cs	C# function mappings.
Mappings.cs	C# function mappings.
Open_uxp_auth.c, c++, c#, .ux;	Demonstrates how to open and interactively authenticate a UXP file using known challenge/response pairs and read contents of virtual files protected within a UXP file. using C, C++ and C# language interfaces
Sample_config.c, c++, c#, .uxl	Demonstrates fetching current machine's location information using the Sertainty SDK; Similarly, the machine's network information can also be fetched from the machine's configuration. This sample only demonstrates location information for brevity - using C, C++ and C# language interfaces
Sample_sm_exchange.c, c#	Sample program using C language interface and SMEX. Uses XML to create ID

	(1) Using either an existing sender UXP ID or create an on-demand, one-time temporary source ID (2) Starts a SSO login session - existing or new (3) Select existing destination UXP ID or create an on-demand, one-time temporary destination ID
sample_workflow_auto_auth.c, c++, c#	Demonstrates a mini workflow, resulting in automatic UXP authentication/ SSO; (1) create ID from XML (2) Create UXP from ID & adds data (3) Closes the UXP (4) Auto authenticates success / fail (5) on success, extracts the data (6) closes UXP
sample_auto_id.xml	Example **UXP ID** xml source definition used to demonstrate automatic UXP authentication for sample_workflow_auto_auth.c, c++ or c#
sample_auto_text.xml	Secure text used to demonstrate automatic UXP authentication utilized by sample_workflow_auto_auth.c, c++ or c# sample
Sample_auto_text.c, .h	Two files are generated from the secure text file sample_auto_text.xml; demonstrates secure, in-memory, text strings, utilized by sample_workflow_auto_auth.c, c++ or c# sample
Sample_workflow_create_id_uxp.c, c++,c#, uxl	Demonstrates a mini workflow, resulting in UXP authentication and data extraction (1) create ID from XML (2) Create UXP from ID & adds data (3) Closes the UXP (4) Authenticates success / fail (5) on success, extracts the data (6) closes UXP
Sample_workflow_delegate.c, c++, .uxl	A delegate can be used to allow a user to access a UXP object without having an identity embedded within the UXP. Access is still controlled by the data owner, but is managed via the delegate database– using **UXL** scripting language This sample exercises various Delegate Service operations using SDK. 1. Users, IDs, and Delegates are added to the Data Services database. 2. Users subscribe to User Delegates 3. UXP is created with Delegate

	4. Upon access to UXP by a Subscriber of the Delegate, subscription metadata like max access, and access count are observed. A UXP identity database is a UXP object that permits SQL access. The database is used to manage users and delegate definitions. A Delegate Identity can be used to allow a user to access a UXP Object without having an identity embedded within the UXP Object. Access is still controlled by the data owner, but is managed via the delegate database.
Sample_workflow_kcl.c, c++	Demonstrates creation of an ID from KCL ID configuration – using KCL language Demonstrates a mini workflow: (1) create ID from KCL, (2) Create UXP from ID & adds data, (3) Closes the UXP, (4) Authenticates success / fail, (5) on success, extracts the data, (6) closes UXP
Sample.kcl	Demonstrates creation of an ID from KCL ID configuration – using KCL language
Sample_workflow_sm.c, c++, UXL	Sertainty SmartMessage sample program using C language interface. New ID is used as a single sign-on object SmartMessage created linking ID. (1) Create ID from XML (2) New ID is used to in a single sign-on session (3) A SmartMessage is creating linking the ID (4) Closes the UXP (5) Authenticates success / fail (6) on success, extracts the data (7) closes UXP
Sample_workflow_sql.c, c++, .sql	Sample workflow using UXP as a SQL database within a UXP Object. Uses XML to create ID (1) Creates ID from xml (2) Creates UXP from ID (3) Creates SQL tables within UXP object (4) Close / Re-opens UXP (5) Accesses SQL / data

Uxp_from_id.c, c++, c#, .uxl	This sample demonstrates how to create a UXP Object using a UXP Identity and add data files into the created UXP Object.
Sample_workflow_sql.c, c++, .sql	Sample workflow using UXP as a SQL database within a UXP Object. Uses XML to create ID (1) Creates ID from xml (2) Creates UXP from ID (3) Creates SQL tables within UXP object (4) Close / Re-opens UXP (5) Accesses SQL / data

Sample scripts are also provided to demonstrate the **UXL Script Engine**. Contains the header files for C and C++ languages.

2.1.1 Deployment

To deploy applications that utilize **UXP Technology** the following guidelines apply:

2. Place the necessary libraries and files in your preferred bin folder:
   - **Linux Libraries**
     - libboost_program_options.so
     - libboost_atomic.so
     - libboost_thread.so
     - libboost_date_time.so
     - libboost_filesystem.so
     - libboost_system.so
     - libSertaintyCore.so
     - boot.ini
   - **MacOSX Libraries**
     - libSertaintyCore.2.dylib
     - libosxfuse_i64.2.dylib
     - boot.ini
   - **Windows Libraries**
     - ssleay32.dll
     - libeay32.dll
     - SertaintyCore2.dll
     - boot.ini
3. Edit boot.ini

   This file contains the various folders used by **Sertainty UXP Technology**. Change the locations to your preferred locations prior to running **UXP Technology.**

   Note: boot.ini must be in the same folder as the SertaintyCore shared library.

The following settings are found in boot.ini:

SharedPath=C:\kit

HomePath=C:\kit\home

InitPath=C:\kit\init

DataPath=C:\kit\data

LogPath=C:\kit\log

WorkoadPath=C:\kit\home

DeveloperPath=C:\kit\developer

4. The following files must be placed in the designated **Sertainty home folder**:
   - **emailpresets.xml**

     This file contains email presets for defining email delivery servers.
   - **filetypes.xml**

     This file contains simple file type associations for **UXP** operations.
   - **init.xml**

     This file contains application settings that can be managed by an administrator.
   - **rulepresets.xml**

     This file contains rule data that can be applied when constructing identities.
   - **sertainty.lic**

     The **Sertainty License** is necessary for all operations. It controls what functions are available to the application.
   - **sertainty*.rsf**

     This file contains critical data utilized by the **UXP Technology**.

2.2 Sample Native Construction Flow using C Language

Included in the SDK example folder are source files that demonstrate how to create a **UXP Object** from C, C++, and C#. The following provides a step-by-step walk-through of building a **UXP Object**. The examples utilize the native C-language interface. C++ and C# interfaces are also available.

a. Validate the library license

When a UXP object-aware application starts, it must enable the **UXP Engine** with a license validation call. The call checks the current runtime license and enables subsequent calls to library functions. Without a valid license, an application cannot call any other **UXP Object** function successfully.
The call is:

```
if (!uxpsys_initializeLibrary(error-buffer,
                              argc,
                              argv,
                              "*",
                              "Sertainty"))
 {
```

```
    handle error
  }
```

b. Create a UXP Identity

A **UXP Identity** can be created from a source definition XML document or via the API. The **Sertainty Assistant** and the **UXL Script** utility have methods for creating a **UXP Identity**.

Note: **The Assistant** contains the required **KCL Code** that is necessary to construct a **UXP Object**.

```
uxpsys_fileReadAll(callstatus, "myid.xml", buffer);

uxpid_publishToFile(callstatus,
                    "myid.iic",
                     uxpba_getData(buffer),
                     ModifierReplace);
```

We now have a binary **UXP Identity** called myid.iic.

c. Create a UXP Object

```
uxpFileHandle myUxp = uxpfile_newHandle(argc, argv)
```

In this example, we create a handle for a **UXP Object** called myUxp. At this point, myUxp is not attached to usable physical **UXP Object**.

```
Uxpfile_openNewFile(myUxp, "myuxp.uxp", "myid.iic", IdFile,
                    ModifierReplace, 0)
```

This will physically create a new **UXP Object** on disk called myuxp.uxp and will replace any existing file by that name.

d. Set optional UXP Object attributes

```
uxpfile_setName(myUxp,"My Data");
uxpfile_setDescription(myUxp,"This is my data");
uxpfile_setCompanyName(myUxp,"ABC Corp");
uxpfile_setOwnerName(myUxp,"Greg Jones");
```

In this example, we have set the common **UXP Object** attributes using the API.

e. Add virtual files to the UXP Object

```
uxpfile_addVirtualFromFile(myUxp,"MyFile1", "mydata.dat",
                           -1,-1, ModifierCompress)
uxpfile_addVirtualFromFile(myUxp,"MyFile2", "mydata2.dat",
                           -1, -1, 0)
```

The specified file is copied to the **UXP Object**. The copy operation will also protect the data from further unauthorized access with the **UXP Object**.

f. Save and close the UXP Object

```
uxpfile_close(myUxp);
```

The close operation saves any remaining metadata and marks the **UXP Object** as valid.

g. Open the UXP Object for access

```
uxpfile_openFile(myUxp, "myuxp.uxp", ShareAll)
```

This will open the file myuxp.uxp as a **UXP Object**. Immediately, the **UXP Engine** will attempt to validate the environment and execute **KCL Code** to determine who is trying to access the **UXP Object**.

```
Status = uxpfile_authenticate(myUxp)
```

If the internal **KCL Code** requires, the status will indicate challenges must be met. In that case, the API grants access to the challenges to be presented to the user. Once the user has responded to the challenges, they will be sent back to the **UXP Engine** for validation.

If all challenges were correctly met, the User is granted access to the **UXP Object** with the privileges defined for the matching User Definition.

If the User incorrectly responded, the **KCL Code** can decide to further challenge the User with additional prompts or deny access.

h. Checking for errors

For all activities, the caller can check for possible error conditions using the following routines:

```
uxpsys_hasError(myUxp)
```

This routine will return a 1 if an error has been detected for the prior call to the **UXP Engine**.

```
Char *bufptr = uxpfile_getErrorMessage(myUxp)
```

This routine will retrieve the error message from the prior call to the **UXP Engine**.

2.3 Required User Definition Elements

When a **User Definition** is defined, there are typically many challenge definitions associated with the User. However, as a rule, each **User Definition** must have the following challenge types to allow any user to access the protected object:

- **USERNAME**

  The username is a keyword that uniquely identifies the user. It can be any number of printable characters. As a recommendation, the length should be a minimum of eight characters.

The above must be created as required challenges.

2.4 Pre-Loading Responses

For automatic authentication, it is possible to push known responses into the **UXP Engine** prior to opening a **UXP Object**. The advantage of this approach is to avoid prompting the caller with challenges to the user identity.

For example, an application keeps sensitive data inside a **UXP Object** that is required to allow the application to operate. The application will attempt to open the object and access data; however, the **KCL Code** rules require proof of identity. Without pushing responses into the engine, the **UXP Engine** will attempt to prompt the caller for the necessary proof.

Pre-loading responses **only works with required challenges** and must contain responses for all required challenges for the User. If any challenge is missing in the pre-loaded responses, the **UXP Engine** will attempt to prompt for the required data.

For example, a **UXP Object** has three required challenges: USERNAME, PASSPHRASE and CODE1. To effectively avoid prompting, one must push a response for the username, the pass phrase and code1 challenges. If any are missing or incorrectly specified, the **UXP Engine** will reject the responses and prompt the user for more proof.

Additionally, pre-loading responses does not guarantee unprompted access. If the **KCL Code** authentication rules detect an anomaly, the u
0ser may still be prompted for additional proof of identity.

> **Note:** If the **KCL Code** variable **isWorkflow** is set to **true**, the system will never prompt the user for responses. This means that a workflow **UXP Object** can only be authenticated using pre-loaded responses.

2.4.1 Opening an Existing UXP Object

Prior to any activity, the **UXP Object** will invoke the **KCL Code** routine **Authentication::userSetup**. This routine typically sets the global variable settings that define the E-mail information and **UXP Object** options. It is only called once.

After the setup routine returns, the **UXP Engine** must now identify the current user. In a loop, the **UXP Engine** will call the routine **Authentication::main** until either the user is granted or denied access. All other events are considered challenges that must be met by the user prior to the continuing the loop. The user cannot access any resource within this **UXP Object** until access is granted by **Authentication::main**.

When developing a custom authentication routine, it must be noted that the **UXP Engine** responds to the **Authentication::main** routine by way of the **setAuthentication** procedure. The **setAuthentication** procedure sets the current user status and privileges.

Only two status values are interesting to the engine:

StatusAuthenticated

StatusNotAuthenticated

All other status values are translated into **StatusChallenged**. When Authentication::main exits, the **UXP Engine** immediately checks the status. If it is not **StatusAuthenticated** or **StatusNotAuthenticated**, it assumes that **StatusChallenged** is the current state. Given that, if you **Authentication::main** routine does not explicitly set the status value, the **UXP Engine** may loop infinitely.

To avoid an infinite loop, always set the status and check the **SessionFailureCount** variable. There should always be an escape mechanism that sets the status to **StatusNotAuthenticated** when **SessionFailureCount** exceeds a reasonable value.

2.4.2 Adding a New Document to the UXP Object

To save a document, the **UXP Object** will generate random cloaking data that drives the cloaking algorithm.

Once the cloaking parameters are set, the specified user document is copied into the **UXP Object** and is protected from any possible unauthorized access.

2.4.3 Accessing an Existing Document in the UXP Object

To access a protected document within the **UXP Object**, the **UXP Engine** first calls the **KCL Code** routine **Authentication::fileAccess**. Within that routine, the owner can determine if the authenticated user can access the specified file. Though not required, the **Authentication::fileAccess** gives the owner fine grain control over **UXP Object** access. For example, the **UXP Object** may contain a file for two users to read; however, user two may only read the document after user one has read and signed it.

2.5 UXP Callbacks

The **UXP** system uses callback functions to communicate with the caller. The following callback functions can be defined:

- **Display a simple message**

  This will be utilized for simple messages to be displayed to the user.

- **Prompt the user for simple input**

  Prompts the user for a single response.

- **Prompt the user for all required challenge input**

  The callback will contain a list of challenges that must be presented to the user. When completed, the callback will return the responses to the **UXP** system for validation.

  Note: A challenge callback is not required if the alternate authentication method is used. Instead, a list of challenges will be returned directly with a get challenges call. Sertainty recommends using the returned challenge list for processing challenges.

- **Progress message**

  Displays a simple progress message for long operations.

- **Progress indicator**

  Provides the caller with progress metrics.

- **Record external event entry**

  Provides a method of recording **UXP Object** activity to a place other than the **UXP Object**.

- **Record external log activity**

  Provides a method of recoding optional log data for support and advance problem resolution.

- **Timeout indicator**

  Provides a mechanism to catch a **UXP Object** timeout. The owner of the **UXP Object** can specify the amount of idle time that can pass before a **UXP Object** will automatically be closed. The callback can catch the notification and can indicate to the **UXP Engine** to ignore the timeout and continue processing.

Each callback accepts a pointer to a user-supplied data structure that will allow the caller to pass control and necessary data back to the caller when the callback is invoked. For example, in a window manager, the callback may need to invoke specific window controls to display a dialog box. The user-supplied data could be a structure containing the necessary methods for invoking the dialog.

2.6 Sample Callback to Prompt the User for Challenges

```
/**
 * Gets challenges from the user
 * @param handle UXP entity handle
 * @param userData User data to pass through on callback
 * @return True if accepted.  False if canceled
 */
static int privatePromptChallenges(uxpUXP entityHandle handle, void *userData)
{
  uxpDataBuffer *buf;
  char value[1000];
  int I,cnt;
  uxpListHandle chList;
  uxpChallengeHandle ch;

  printf("%s\n","To verify your identity, please provide the following:");

  chList = uxpGetChallenges(handle);
  cnt = uxplistCount(chList);

  buf = uxpNewBuffer();

  for (I = 0; I < cnt; i++)
  {
    ch = uxplistGetChallenge(chList, i);

    uxpchStartTimer(ch);

    for (;;)
    {
      printf("%s> ", uxpchGetPrompt(ch, buf));
      gets(value);

      if (strlen(value) > 0)
```

```
            break;
        }

        uxpchEndTimer(ch);
        uxpchSetValueString(ch, value);

        uxpAddResponse(handle, ch);
    }

    return 1;
}
```

2.7 Application Example Scenarios

The following are typical scenarios that may benefit from **UXP Identity (UXP ID) Technology**:

1. **Protecting files in the directory without requiring a UXP Object as a container**. When utilized, this approach is easier to understand and manage for non-technical users. For example, if one has a computer folder containing valuable documents, a personal **UXP Identity** can take each document and produce a protected counterpart in the same folder. The original could then be deleted. To access the protected document, the owner would open the personal **UXP Identity** and request that the document be unprotected, again, in the same folder.

   Note: In all cases, the protected document can only be accessed by the owner of the **UXP Identity**.

2. **Sending lightweight messages between trusted end points.** For example, a power plant contains a controller with a digital interface. The controller is responsible for managing power availability to various geographic areas and can be managed remotely by a trusted console. In this case, the trust is two-way in that the controller trusts that the commands are coming from the correct console, and the console assumes that messages are being received and interpreted by the trusted controller.

   With a **UXP Identity**, the controller and the console can send messages without fear. If constructed appropriately, messages sent from the trusted console can only be constructed by that device located in the correct geographic location. Plus, the user of the console is the only person that can authenticate the console **UXP Identity**. On the controller side, a message can only be read and trusted if it was protected according to the console **UXP Identity**. It cannot be read by any entity other than the console user (which can be a machine) residing at the power plant.

3 Advanced Technologies

The **UXP** system has a number of advanced functions that can be used to not only increase the protection of a **UXP Object** but allow a developer to increase security within the application.

3.1 SQL Engine

A **UXP Object** has two on-disk structure types used to manage its proprietary protection.

The first, and simplest form is the Write-Once architecture. This variant permits storing and accessing data; however, it does not support updating existing entities within the **UXP Object**.

The second form allows full read-write capability within the **UXP Object**. This format also introduces an embedded SQL engine that is used to manage the extended features.

The **UXP SQL** engine can be accessed by a licensed user to store highly structured data the same way one would utilize standard commercial SQL database systems. The fundamental difference with the **UXP SQL** engine is that it utilizes UXP protection and authentication protocols to manage the security of the data.

Once a Read-Write **UXP Object** has been created, a user may issue standard SQL statements to define metadata and manage user data. Though the **UXP SQL** data is not viewed as virtual files like the unstructured data within the **UXP Object**, it is protected and governed by the same UXP protocol.

Note: The **UXP SQL** engine cannot interact with existing unstructured data included as virtual files in the **UXP Object**. The SQL data can co-exist with a virtual file within a **UXP Object**; it just cannot access or modify the data. A future release may remove this restriction.

3.1.1 Features

The **UXP SQL** engine is based on a subset of SQLite. It supports the most common features of SQLite such as:

- BEGIN TRANSACTION
- Built-in SQL functions
- COMMIT TRANSACTION
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW
- INSERT
- ROLLBACK TRANSACTION
- SELECT
- UPDATE
- UPSERT

Unsupported:

- ATTACH database command.
- DETACH database command.
- Metadata table access.
- PRAGMA command.
- SAVEPOINT command.

Example:

```
uxp::sql::query q(&uxpobject);

q.prepare("select c1,c2 from t1 where c3 = ?");
q.bindValue(0,"greg");
q.exec();

while (q.next())
{
```

```
    Std::cout << "c1: " << q.valueText(0) << std::endl;
    Std::cout << "c2: " << q.valueInt64(1) << std::endl;
}

q.finish();
```

3.2 Secure Variables

No matter how well on-disk data is protected, in-memory copies found in common variables represent a significant risk. Simple reverse engineering of code by dumping memory can produce clear versions of protected data.

The **UXP** system provides a simple library of secure variable management for common data types. The following types are supported:

- Integers
- Double precision floating point numbers
- Strings
- Array of bytes

Each of the types masks the data and additionally adds tamper-resistant measures. This creates a significant deterrent to kernel debuggers and memory scrapers.

Table 3 – Secure variable C++ classes

Variable Class	Description
uxl::secureInt	Manages secure 64-bit integers.
uxl::secureDouble	Manages double precision numbers.
uxl::secureString	Manages secure variable-length strings.
uxl::secureByteArray	Manages secure variable-length byte array.

3.3 Secure String Constants

Most systems deploy static text for use within an application. Unless measures are taken, static text of a sensitive nature can be discovered by scanning binary code and memory dumps. This is especially perilous for a system like the **UXP Engine**. What good is a data protection scheme, if an unauthorized agent can easily reverse engineer embedded hints to authentication and protection?

The **UXP** error messaging system provides an API to load and access secure strings. Secure strings are stored as an external binary file that is created from a source XML document. Once compiled, the strings can be loaded at runtime by an authorized agent and used by application for any purpose that requires static text.

Examples of strings that would benefit from secure string technology are passwords, context-sensitive test, internal structure names, etc.

The format of a secure string source file is:

```
<TextBundle>
    <Text key="TXT_T_PASSWORD">fasdfasdfasdf</Text>
    <Text key="TXT_T_SPECIAL">This is a special string</Text>
 </TextBUndle>
```

Table 4 – Secure String XML tags and attributes

Tag	Description
TextBundle	The main tag for the bundle.
Text	Defines a new string.
key	Defines the unique key that can be used to fetch the string content. When the bundle is compiled, the **key** is stored in a generated file as **a #define**.

To compile a secure string source document, run the **SertaintyString** utility. It is a console application that must be run in a terminal shell.

SertaintyString command-line-arguments

Supported commands:

```
Sertainty Secure String Utility:
  -h [ --help ]         Produce help message
  -s [ --source ] arg   Source file defining secure strings..
  -o [ --outfile ] arg  File to store secure strings.
  -c [ --codefile ] arg Generated code file.
  -i [ --incfile ] arg  Generated code header file.
  -d [ --domain ] arg   Domain name for secure strings.
  -b [ --buffer ]       Write messages to embedded buffer instead of message file.
  -r [ --replace ]      Replace existing output files.
  --k1 arg              Security key.  Must be 32 characters.
  --k2 arg              Security key.  Must be 32 characters.
  --k3 arg              Security key.  Must be number between 1 and 32.
  --k4 arg              Security key.  Must be number between 1 and 32.
```

Table 5 – Sertainty String Utility command-line arguments

Option	Description
-h	Generates a help listing.
-s source-file	Specifies the source XML document containing <TextBundle> artifacts. **Required**

Option	Description
-o output-file	Specifies the name of the compiled text string file. It must only be a file name as the system requires all compiled message files be placed in the Sertainty home folder. **Required if the file option is used. Not required for buffer option.**
-b	Specifies the compiled messages will be encoded directly in a native C or C++ file instead of a .msg file. The benefit is that there is no external file to manage.
-c code-file	Specifies the generated code file that contains necessary components to load the compiled string bundle. This file contains a secret key that must be protected from unauthorized viewing. The module also contains an initialization routine that must be called by the application to load the compiled strings. If the file type is **cpp,** the generated file is a C++ source module. If the file type is c, the generated file is a C source module. **Required**
-i include-file	Specifies the header file for the compiled strings. The compiled string keys from the source document are defined here so that an application may use them to fetch strings. **Required**
-d domain-name	Specifies a name for the compiled message bundle. The name is used when loading and fetching secure strings. **Required**
-r	Replace existing files, if they exist. **Optional**
--k1 value	Specifies a 32-character key used for string compilation. **Required**
--k2 value	Specifies a 32-character key used for string compilation. **Required**
--k3 value	Specifies an integer between 1 and 32 used for string compilation. **Required**
--k4 value	Specifies an integer between 1 and 32 used for string compilation. **Required**

To load secure text for access, use the init function found in the generated header file. The actual **init** function name is formed by taking the designated message output file and appending **"_init"**. The function must be called once prior to fetching any secure strings from the message file.

To fetch a secure string, use the uxl::message class in C++ or the uxlmsg_ C library.

For C++:

```
    /**
     * @brief Gets a secure text string.  Secure strings are protected on disk
     *        and in memory, meaning that a memory dump will expose nothing.
     * @param domain Secure string domain
     * @param id Message identifier
     * @return Text string
     */
    const std::string uxl::message::getSecureText(const std::string &domain,
                                                  uxp_int16 id);
```

For C:

```
    /**
     * @brief Gets a secure text string.  Secure strings are protected on disk
     *        and in memory, meaning that a memory dump will expose nothing.
     * @param status Status handle
     * @param outbuf Buffer to receive string
     * @param domain Secure string domain
     * @param id Message identifier
     */
    void uxlmsg_getSecureText(uxpCallStatusHandle *status,
                              uxlByteArray *outbuf,
                              const char *domain,
                              uxp_int16 id);
```

3.4 Custom error and text messages

The **UXP** error handling system utilizes two forms of static text. When creating an error message, a registered error message array can be utilized to signal very detailed messages. Because the messages are registered, a single error message can be use many times without duplicating static in-memory test.

A second feature of the messaging system is simple text messages. This is a variant of the error messaging system that uses the static registered message text to define a static string one time, thus, avoiding duplication. Additionally, by storing text in a messaging array, the static text cannot be used as attack point by unwelcomed adversaries. If one embeds static text, a code dump may reveal proximity of critical code, thus, providing a starting point for reverse-engineering private code.

To define text or error arrays, one must create two modules: a header file of fixed numeric constants, and a source module that implements the static text as an array.

Sample error message header file:

```
#include "uxlcommon.h"

   /* Error message identifiers */

   typedef enum
   {
     My_message_01      = 1001,
     My_message_02      = 1002,
     My_message_03      = 1003,
```

```
    My_message_04      = 1004,
    My_message_05      = 1005,
    My_message_06      = 1006,
    My_message_07      = 1007
  } my_message_id_t;

  /**
   * Simple class to register errors
   */

  namespace mySpace
  {

    class UXPLB_HIDDEN MyErrors
    {
    public:
      static void registerErrors(void);
    } ;
  }
```

Sample error message code file:

```
  #include "uxlmessage.h"
  #include "uxlerrors.h"

  static uxlMessageItem gl_msg_array[] = {
    { My_message_01, "The day %1 is invalid"},
    { My_message_02, "The procedure %1 was not found"},
    { My_message_03, "The day is wrong: %1 "},
    { My_message_04, "The sky is blue exists"},
    { My_message_05, "Invalid operation"},
    { My_message_06, "Invalid name"},
    { My_message_07, "Invalid type"},
    {
      0, ""
    }
  };

  void mySpace::MyErrors::registerErrors(void)
  {
    uxl::message::loadMessages(gl_msg_array);
  }
```

The above modules should be compiled and included in your application. The class method mySpace::MyErrors::registerErrors() should be called one time prior to using the error messages.

Error messages support argument substitution. In the example, several messages have a percent character followed by a number. The token represents a substitution value where the number is the relative argument number to inserted into the final message. When logging an error with a message requiring an argument, one must instantiate a message object with the designated error message identifier. Then, an argument must be added to the message object. Next, the message can be passed through the exception system or by other means of delivering a message. Upon reaching its destination, a message can be extracted, fully formatted and ready to present to a user.

For simple text, the same process of defining text message must be followed. The main difference from error messages, however, is that text messages are simple text that can be retrieved directly from the message array.

Sample text message header file:

```
#include "uxlcommon.h"

/* Text message identifiers */

typedef enum
{
  text_0001 = 2001,
  text_0002 = 2002,
  text_0003 = 2003,
  text_0004 = 2004,
  text_0005 = 2005,
} text_id_t;

/**
 * Simple class to register text messages
 */
namespace mySpace
{

  class UXPLB_HIDDEN text
  {
  public:
    static void registerText(void);
  } ;
}
```

Sample text message code file:

```
#include "uxlmessage.h"
#include "text.h"

static uxlMessageItem gl_msg_array[] = {
  { text_0001, "Member foobar expires within 3 days." },
  { text_0002, "Member \"%1\" has expired." },
  { text_0003, "Member \"%1\" expires today." },
  { text_0004, "Member \"%1\" expires tomorrow." },
  { text_0005, "Member \"%1\" has no privileges." },
  {
    0, ""
  }
};

void mySpace::text::registerText(void)
{
  uxl::message::loadText(gl_msg_array);
}
```

Like error message, the above text modules should be compiled and included in your application. The class method mySpace::text::registerText() should be called one time prior to using the text messages.

To use a text message, call the uxl::message::getText(id) where id is the desired text identifier.

Example:

```
Std::string t = uxl::message::getText(text_0001);
std::cout << “Text: “ << t << std::endl;
Member foobar expires within 3 days.
```

Note: all user-defined error and text message identifiers must be greater than 1000. If less than 1000, custom identifiers may conflict with intrinsic UXP errors and text messages. The above samples assume C++ as the native language; however, a developer may use the C language since the error and text message construction will be the same.

3.5 Building native UXL functions

If the **UXL** language system requires additional functionality, it's possible to write custom functions that can be called from **UXL** scripts, including the *Workflow Data Protector*.

Native functions can be built using any native language; however, the main function definition must be done in C++. Once defined, the new functions are placed inside a dynamic shared library, which can be loaded by the **UXL** language system.

3.5.1 Getting started

Two modules must be built based on the C++ base classes uxp::scriptPlugin and uxp::scriptFunction. The base classes are found in module **uxpscript.h**.

The class uxp::scriptPlugin defines the linkage that must occur between the **UXL** engine and the native **UXL** function. It contains two required functions that are used by the UXL engine to recognize the native functions. Only one instance of uxp::scriptPlugin can exist within a shared library and must be visible externally as the name uxp_**scriptplugin**. When the shared library is loaded, the UXL engine attempts to locate the uxp_scriptpugin object and query the object for defined functions.

A native function is derived class that extends the class uxp::scriptFunction. The following table describes the uxp::scriptFunction class.

Table 6 – Native function methods

Function	Description
Constructor(package, name, min, max)	Constructor for new object. It defines the package name, function name, minimum and maximum arguments for the new native function.
execute()	A pure virtual function that must implemented in the new function class. When a UXL engine calls the native function, the method is called and permits execution of custom native code. **Must be implemented by developer**.
getName()	Returns the name of the native function. A function is called within UXL using the concatenation of the package name and the function name. Example: special::myFunc()
getPackageName()	Returns the package name of the function.

Function	Description
getArgCount()	Returns the number of arguments that have been passed into the function at execution time. A function is defined as having a minimum and maximum number of arguments. If a caller fails to provide an acceptable number of arguments, an exception will be thrown and execution will stop.
getArgument(offset)	Returns the request argument. The offset is zero-based and can be a value ranging from the minimum number of arguments minus one to the number returned by the getArgCount() minus one. The returned value will be stored in a uxl::variant object.
setReturn(value)	Allows the custom function to provide a return argument to the caller.

See the plugin sample modules sampleplugin.cpp and sampleplugin.h for help in building a native function.

To use the native functions, call the **x::loadPackage UXL** function. This will load the shared library and set up the desired native functions within the library. The shared library must be within the normal search path for shared libraries; otherwise, the load will fail.

3.6 SmartMessage

SmartMessage is a low-level protocol that can protect data in flight. Rather than sending a entire **UXP Object**, with all its overhead, over a connection, **SmartMessage** permits a similar, yet light weight, alternative.

A message is protected similar to the contents of a **UXP Object**; however, it does not contain the **KCL Code** or a virtual file system. There is a small header module, but for the most part, the message is much smaller than the **UXP Object** counterpart.

A **SmartMessage** contains the following:

Table 14 – SmartMessage Properties

Property	Optional	Description
Created	Yes	Contains the creation date and time.
Data size	No	Contains the native size of the user data.
Flags	No	Contains the original flags used to create the message.
Internal header	No	Contains private data necessary to managed message content.
ID.DeviceID	Yes	Contains the physical device identifier.
ID.ExchangeIdentity	Yes	Contains the target ID unique identifier.
ID.Identity	Yes	Contains the ID unique identifier.
ID.IP	Yes	Contains the network IP address.
ID.Location	Yes	Contains physical address.

Property	Optional	Description
ID.LocationID	Yes	Contains the physical location identifier.
ID.Name	Yes	Contains the ID name.
ID.Personal1	Yes	Contains the ID personal name.
ID.Trusted	Yes	Contains a Boolean indicating whether the ID is trusted.
ID.User	Yes	Contains the current session user name.
User data	No	Zero to 'N' bytes of user data.

The **SmartMessage** can exclude properties marked as optional at creation time. By removing optional properties, the size of the **SmartMessage** is reduced by several hundred bytes.

Required workflow:

- A user must establish a session using a **UXP Identity**. The **UXP Identity** must have **SmartMessage** privileges in order to create the **SmartMessage**.
- Choose a target **UXP Identity** for protecting the data. The target **UXP Identity** controls the access to the new **SmartMessage**.
- Create a **SmartMessage** in a buffer or a file using the target **UXP Identity**. An optional modifier **ModifierMinSize** can be set to exclude the optional properties.
- Append data to **SmartMessage**. An optional flag to compress data can be utilized.
- Close the new **SmartMessage**. From the point on, only a session using the target **UXP Identity** can open and access the **SmartMessage**.

Sample Use-Case:

Secure Chat

- Two users wish to have a private, secure chat. Both user A and B must have valid **UXP IDs**.
- A chat server has the **UXP Identities** for both user A and B.
- User A logs into chat system using personal **UXP Identity**.
- User B logs into chat system using personal **UXP Identity**.
- User A locates user B in chat system.
- User A sends a **SmartMessage** to user B using user B's **UXP Identity**. User B is the only entity that can open and read the **SmartMessage**.
- User B replies to user A using user A's **UXP Identity**. Again, only user A can open and read the **SmartMessage**.
- Exchanges can be repeated as in a typical conversation.
- To end the chat conversation, user A and user B terminate their chat server session.

In the above use-case, even the chat server will not be able to read the messages that are being exchanged.

3.7 Anonymous SmartMessage Exchange (SMEX)

SmartMessage Exchange was engineered to facilitate secure communication and data interchange between trusted endpoints and to provide protection for data in-flight. SMEX utilizes **Just in Time (JIT**) temporary **SmartID Technology**, which enables secure data exchange using the lightweight **SmartMessage**. The **JIT Smart UXP Identities** are short lived and are only valid for a single exchange session. It is impractical to break into a **JIT Smart UXP Identity**, as the challenges are randomized during its creation, which results in every new session being created with a different **Smart UXP Identity**.

In use cases where know **Smart UXP Identities** are being used for **SmartMessaging**, **SMEX** could further obfuscate and protect the data exchange.

Workflow:

The following steps are required in order to establish and use **SMEX** to setup a secure communication channel.

- **SMEX Handshake**
- **Initialize SMEX**

  Both parties involved in an exchange session initialize **SMEX** locally. This will result in creation and local activation of **JIT Smart UXP Identities**. **JIT Smart UXP Identities** will now be available for exchange.

- **Exchange JIT Smart UXP Identities**

  The parties exchange the **JIT Smart UXP Identities** in order to complete the final step of the handshake.

- **Start SMEX Session**

  Once the **JIT Smart UXP ID** is received from remote peer, it can be used to start **SMEX** session on an already initialized **SMEX**. This will result in a **SmartMessage Session** being created between the two **JIT Smart UXP Identities**.

  SMEX is now ready to protect further exchanges between the peers.

- **Secure Data Exchange**
- **Encode**

  Use the encode routine to protect data before transmitting to the remote peer in an **SMEX Session.**

- **Transmit**

  Once encoded, the data can be safely transmitted to the remote peer.

- **Decode**

Use the decode routine in an **SMEX Session** to reveal the encoded data transmission received from the remote peer.

- **Alternate Handshake Workflow**

  - Initialize **SMEX** on Peer A
  - Transmit Peer A **JIT Smart UXP Identity** to Peer B
  - Initialize **SMEX** on Peer B and Start **SMEX Session** using Peer A's **JIT Smart UXP Identity** on Peer B
  - Transmit Peer B **JIT Smart UXP Identity** d to Peer A
  - Start **SMEX Session** using Peer B's **JIT Smart UXP Identity** on Peer A

4 Sertainty UXP Identity (UXP ID)

A **UXP ID** is a special purpose **UXP Object** that contains a secure, portable identity. With a UXP **ID**, a **UXP Object-aware application** can perform the following:

- Create a **UXP Object** without manually writing any **UXL** modules. A **UXP Object** created using this technique will be governed by the rules within the **UXP ID**.

  Example: Person A can provide a **UXP ID** that will allow person B to protect data that only person A can access.

- Create a **SmartMessage**. A **SmartMessage** is a protected data entity that does not reside within a **UXP Object**. The data can be an in-memory buffer or a persistent on-disk file.

  Example: A **SmartMessage Object** would be useful for communications among secure peers, such as utilities infra-structure or person-to-person links.

- Create a **UXP Single-Sign-On**. This **UXP ID** has only a single identity user and nothing more. This would be useful for protecting web-sites, machine entry points, databases, etc.

A **UXP ID** contains the following public information:

- Name and address of **UXP ID** owner
- Unique **UXP** I**D** identifier
- Optional photo and business contact information

Native APIs are supported in C++ and C; other language interfaces are planned. The **Workflow Assistant** fully implements the **UXP ID** technology, which can then be used by the **UXP Technology**.

4.1 UXP ID Construction

UXP Identity (UXP ID) data consists of several components to form a unique digital fingerprint of the client. The basic components necessary to build a valid **UXP ID** are:

- **Users Definitions**

  Each user definition contains the following components:

  - **Challenge pairs**

    A challenge is a setting that can only be met by the authentic user. The number of challenges required to permit access is set by user preferences.

    Many challenges are local prompt / response pairs. Others can be potentially be complex responses in the form of biometric input.

    In some cases, a challenge may require other trusted users to respond with a code that was sent at authentication time.

  - **Device and location configurations**

    The **KCL Code** attempts to determine the state of its operating environment, including device data, location data, software conditions and time. The user can set preferences that make decisions based on trust and the environment.

    > **Example:** if a device is not recognized, a preference can indicate the user must respond to an extra three challenges.

  - **Preferences or Rules**

    Preferences are various settings that determine the **KCL Code** behavior. Some preferences are unique to the user, and some can be overridden by global preferences within the **ID**. The **KCL Code** chooses the most restrictive setting when deciding between a user preference and a global preference.

  - **Schedule**

    A schedule is a flexible calendar that determines when data can be accessed. It can be as simple as picking a date to allow access, or it can be a perpetual setting such as allowing access on Monday through Friday, 8AM to 5PM. Like preferences, a schedule can exist at the user level and at the global level.

- **UXP ID Definitions**

  A definition is the basis of **UXP ID**. It contains public information and protected information. The public information is meant to be visible and should be considered when determining the validity of a **UXP ID**.

  - **Global device and location configurations**

    The **KCL Code** attempts to determine the state of its operating environment, including device data, location data, software conditions and time. The user can set preferences that make decisions based on trust and the environment.

    > **Example:** if a device is not recognized, a preference can indicate the user must respond to an extra three challenges.

  - **Global preferences or rules**

    Preferences are various settings that determine the **KCL Code** behavior. Some preferences are unique to the user, and some can be overridden by global preferences within the **UXP ID**. The **KCL Code** chooses the most restrictive setting when deciding between a user preference and a global preference.

  - **Global schedule**

    A schedule is a flexible calendar that determines when data can be accessed. It can be as simple as picking a date to allow access, or it can be a perpetual setting such as allowing access on Monday through Friday, 8AM to 5PM. Like preferences, a schedule can exist at the user level and at the global level.

  - **Users**

Users are required to authenticate the **UXP ID**. Each user within a **UXP ID** must also be a **User Definition.** And, for a **UXP ID** to be valid, there must be at least one valid user.

A user can also be imported for another **UXP ID**. In this scenario, an imported user cannot be changed, nor can any of the private user data be viewed. One would use an imported user to construct a shared **UXP ID** that contains multiple users.

- ## Managed ID Interface (MID)

The **MID interface** is a simple API that enables a user to fully implement a **UXP ID** using an XML document. The document schema defines all the required and optional artifacts that can contribute to a valid **UXP ID**. The API is implemented in the **UXP ID** modules under **UXP ID** definitions.

4.1.1.1 Example of Full ID XML Schema, complete with one valid user

The following represents a full **UXP ID** XML schema, complete with one valid user:

```
<?xml version="1.0"?>
<!---->
<!-- ID Definition: My Personal -->
<!-- Date:          2018-03-15T10:17:10 -->
<!---->
<ID name="My Personal">
  <Description type="string"></Description>
  <Expiration type="string">2019-01-01T05:59:00</Expiration>
  <PersonalName1 type="string"></PersonalName1>
  <PersonalName2 type="string"></PersonalName2>
  <PersonalName3 type="string"></PersonalName3>
  <Address1 type="string"></Address1>
  <Address2 type="string"></Address2>
  <City type="string">Madison</City>
  <State type="string"></State>
  <Zipcode type="string"></Zipcode>
  <Phone1 type="string"></Phone1>
  <PhoneType1 type="string"></PhoneType1>
  <Phone2 type="string"></Phone2>
  <PhoneType2 type="string"></PhoneType2>
  <Phone3 type="string"></Phone3>
  <PhoneType3 type="string"></PhoneType3>
  <Photo type="string"></Photo>
  <Privileges type="string">Files,Messages,SSO,Imports</Privileges>
  <!---->
  <Rules>
    <Rule name="GeneralSetup">
      <AdvancedDataLogging type="bool">false</AdvancedDataLogging>
      <AlternateReality type="string"></AlternateReality>
      <Compliance type="date">03/14/2218 20:00</Compliance>
      <MaximumAccesses type="int">0</MaximumAccesses>
      <MaximumCycleFailures type="int">4</MaximumCycleFailures>
      <MaximumIdleTime type="int">500</MaximumIdleTime>
      <MaximumTotalFailures type="int">0</MaximumTotalFailures>
      <UseLocalTime type="bool">false</UseLocalTime>
      <Workflow type="bool">false</Workflow>
    </Rule>
    <!---->
    <Rule name="RestrictionsSetup">
      <ConfigurationAltReality type="bool">false</ConfigurationAltReality>
      <ConfigurationApproval type="bool">false</ConfigurationApproval>
      <ConfigurationDeny type="bool">false</ConfigurationDeny>
      <ConfigurationDestroy type="bool">false</ConfigurationDestroy>
      <ConfigurationPrompts type="int">0</ConfigurationPrompts>
      <EveryAuthenticationApproval type="bool">true</EveryAuthenticationApproval>
      <EveryAuthenticationPrompts type="int">4</EveryAuthenticationPrompts>
      <HardwareAltReality type="bool">false</HardwareAltReality>
      <HardwareApproval type="bool">false</HardwareApproval>
```

```
  <HardwareDeny type="bool">false</HardwareDeny>
  <HardwareDestroy type="bool">false</HardwareDestroy>
  <HardwarePrompts type="int">0</HardwarePrompts>
  <MovementAltReality type="bool">false</MovementAltReality>
  <MovementApproval type="bool">false</MovementApproval>
  <MovementDeny type="bool">false</MovementDeny>
  <MovementDestroy type="bool">false</MovementDestroy>
  <MovementPrompts type="int">0</MovementPrompts>
  <NetworkAltReality type="bool">false</NetworkAltReality>
  <NetworkApproval type="bool">false</NetworkApproval>
  <NetworkDeny type="bool">false</NetworkDeny>
  <NetworkDestroy type="bool">false</NetworkDestroy>
  <NetworkPrompts type="int">0</NetworkPrompts>
  <Preset type="string"></Preset>
  <ReadOnlyAltReality type="bool">false</ReadOnlyAltReality>
  <ReadOnlyApproval type="bool">false</ReadOnlyApproval>
  <ReadOnlyDeny type="bool">false</ReadOnlyDeny>
  <ReadOnlyPrompts type="int">0</ReadOnlyPrompts>
  <ScheduleAltReality type="bool">false</ScheduleAltReality>
  <ScheduleApproval type="bool">false</ScheduleApproval>
  <ScheduleDeny type="bool">false</ScheduleDeny>
  <ScheduleDestroy type="bool">false</ScheduleDestroy>
  <SchedulePrompts type="int">0</SchedulePrompts>
  <UntrustedSystemAltReality type="bool">false</UntrustedSystemAltReality>
  <UntrustedSystemApproval type="bool">false</UntrustedSystemApproval>
  <UntrustedSystemDeny type="bool">false</UntrustedSystemDeny>
  <UntrustedSystemDestroy type="bool">false</UntrustedSystemDestroy>
  <UntrustedSystemPrompts type="int">0</UntrustedSystemPrompts>
  <UntrustedTimeAltReality type="bool">false</UntrustedTimeAltReality>
  <UntrustedTimeApproval type="bool">false</UntrustedTimeApproval>
  <UntrustedTimeDeny type="bool">false</UntrustedTimeDeny>
  <UntrustedTimeDestroy type="bool">false</UntrustedTimeDestroy>
  <UntrustedTimePrompts type="int">0</UntrustedTimePrompts>
</Rule>
<!---->
<Rule name="ConfigurationSetup">
  <Configurations />
</Rule>
<!---->
<Rule name="AlertSetup">
  <EmailAddress type="string"></EmailAddress>
  <IncludeDevice type="bool">true</IncludeDevice>
  <IncludeLicense type="bool">false</IncludeLicense>
  <IncludeLocation type="bool">true</IncludeLocation>
  <SMSAddress type="string"></SMSAddress>
  <UseEmail type="bool">false</UseEmail>
  <UseSMS type="bool">false</UseSMS>
</Rule>
<!---->
<Rule name="ApprovalSetup">
  <ExternalLength type="int">6</ExternalLength>
  <Approval name="6">
    <Enabled type="bool">false</Enabled>
    <Address type="string"></Address>
    <Description type="string"></Description>
    <Prompt type="string">6</Prompt>
    <Type type="string">SMS</Type>
  </Approval>
</Rule>
<!---->
<Rule name="EventSetup">
  <EmailAddress type="string">0</EmailAddress>
  <ExternalKey type="string">0</ExternalKey>
  <LogAccesses type="bool">true</LogAccesses>
  <LogFailures type="bool">true</LogFailures>
  <LogMessages type="bool">true</LogMessages>
  <LogRepeats type="bool">true</LogRepeats>
  <RecordEmail type="bool">false</RecordEmail>
  <RecordExternal type="bool">false</RecordExternal>
  <RecordLocal type="bool">true</RecordLocal>
  <RecordRemote type="bool">false</RecordRemote>
  <RecordSMS type="bool">false</RecordSMS>
  <SMSAddress type="string">0</SMSAddress>
```

```
      <URL type="string">0</URL>
    </Rule>
    <!---->
    <Rule name="ScheduleSetup">
      <Enabled type="bool">true</Enabled>
      <DaySunday type="bool">true</DaySunday>
      <DayMonday type="bool">true</DayMonday>
      <DayTuesday type="bool">true</DayTuesday>
      <DayWednesday type="bool">true</DayWednesday>
      <DayThursday type="bool">true</DayThursday>
      <DayFriday type="bool">true</DayFriday>
      <DaySaturday type="bool">true</DaySaturday>
      <StartDay type="int">-1</StartDay>
      <StartHour type="int">-1</StartHour>
      <StartMinute type="int">-1</StartMinute>
      <StartMonth type="int">-1</StartMonth>
      <StartYear type="int">-1</StartYear>
      <EndDay type="int">-1</EndDay>
      <EndHour type="int">-1</EndHour>
      <EndMinute type="int">-1</EndMinute>
      <EndMonth type="int">-1</EndMonth>
      <EndYear type="int">-1</EndYear>
    </Rule>
  </Rules>
  <!---->
  <Users>
    <User name="youremail@someplace.com" type="Personal">
      <Enabled type="bool">true</Enabled>
      <Email type="string">youremail@someplace.com</Email>
      <Expiration type="string">2019-01-01T05:59:00</Expiration>
      <FormalName type="string"></FormalName>
      <Privileges type="string">Read,Write,Delete,Print,Copy,Owner</Privileges>
      <!---->
      <Private>
        <Masking type="string">0</Masking>
        <!---->
        <Rules>
          <Rule name="UserAdvancedSetup">
            <MaximumTime type="int">30</MaximumTime>
            <MaximumTotalTime type="int">99</MaximumTotalTime>
            <MinimumTime type="int">0</MinimumTime>
            <MinimumTotalTime type="int">0</MinimumTotalTime>
          </Rule>
          <!---->
          <Rule name="UserBasicSetup">
            <IgnoreCase type="bool">true</IgnoreCase>
            <IgnoreChars type="string"></IgnoreChars>
            <MinimumPrompts type="int">3</MinimumPrompts>
          </Rule>
          <!---->
          <Rule name="UserRecoverySetup">
            <MaximumFailures type="int">1</MaximumFailures>
            <MinimumCorrect type="int">5</MinimumCorrect>
          </Rule>
        </Rules>
        <!---->
        <Challenges>
          <Challenge name="ch01">
            <Enabled type="bool">true</Enabled>
            <Hashed type="bool">false</Hashed>
            <Prompt type="string">test1</Prompt>
            <Required type="bool">false</Required>
            <Response type="string">test1</Response>
            <Softkb type="bool">false</Softkb>
          </Challenge>
          <!---->
          <Challenge name="ch02">
            <Enabled type="bool">true</Enabled>
            <Hashed type="bool">false</Hashed>
            <Prompt type="string">test2</Prompt>
            <Required type="bool">false</Required>
            <Response type="string">test2</Response>
            <Softkb type="bool">false</Softkb>
```

```
      </Challenge>
      <!---->
      <Challenge name="ch03">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test3</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test3</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
      <!---->
      <Challenge name="ch04">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test4</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test4</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
      <!---->
      <Challenge name="ch05">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test5</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test5</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
      <!---->
      <Challenge name="ch06">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test6</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test6</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
      <!---->
      <Challenge name="ch07">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test7</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test7</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
      <!---->
      <Challenge name="ch08">
        <Enabled type="bool">true</Enabled>
        <Hashed type="bool">false</Hashed>
        <Prompt type="string">test8</Prompt>
        <Required type="bool">false</Required>
        <Response type="string">test8</Response>
        <Softkb type="bool">false</Softkb>
      </Challenge>
    </Challenges>
  </Private>
  <!---->
  <Rules>
    <Rule name="ConfigurationSetup">
      <Configurations />
    </Rule>
    <!---->
    <Rule name="ApprovalSetup">
      <ExternalLength type="int">6</ExternalLength>
      <Approval name="6">
        <Enabled type="bool">false</Enabled>
        <Address type="string"></Address>
        <Description type="string"></Description>
        <Prompt type="string">6</Prompt>
        <Type type="string">SMS</Type>
      </Approval>
    </Rule>
```

```
        <!---->
        <Rule name="ScheduleSetup">
          <Enabled type="bool">true</Enabled>
          <DaySunday type="bool">true</DaySunday>
          <DayMonday type="bool">true</DayMonday>
          <DayTuesday type="bool">true</DayTuesday>
          <DayWednesday type="bool">true</DayWednesday>
          <DayThursday type="bool">true</DayThursday>
          <DayFriday type="bool">true</DayFriday>
          <DaySaturday type="bool">true</DaySaturday>
          <StartDay type="int">-1</StartDay>
          <StartHour type="int">-1</StartHour>
          <StartMinute type="int">-1</StartMinute>
          <StartMonth type="int">-1</StartMonth>
          <StartYear type="int">-1</StartYear>
          <EndDay type="int">-1</EndDay>
          <EndHour type="int">-1</EndHour>
          <EndMinute type="int">-1</EndMinute>
          <EndMonth type="int">-1</EndMonth>
          <EndYear type="int">-1</EndYear>
        </Rule>
      </Rules>
    </User>
  </Users>
</ID>
```

4.1.1.2 Example of User XML schema

The following represents a user XML schema:

```
<?xml version="1.0"?>
<!---->
<!-- ID Users: My Personal -->
<!-- Date:     2018-03-15T10:20:40 -->
<!---->
<ID>
  <Users>
    <User name="otheruser@someplace.com" type="Trusted">
      <Enabled type="bool">true</Enabled>
      <Email type="string">heruser@someplace.com</Email>
      <Expiration type="string">2117-03-14T20:00:00</Expiration>
      <FormalName type="string"></FormalName>
      <Privileges type="string">Read,Write,Delete,Print,Copy,Sign,Owner </Privileges>
      <!---->
      <Private>... Base64 ... encoded</Private>
      <!---->
      <Rules>
        <Rule name="ConfigurationSetup">
          <Configurations>
            <Configuration id="0" name="id">
              <Enabled type="bool">false</Enabled>
              <!---->
              <Device id="0" name="id">
                <Architecture type="string"></Architecture>
                <CpuModel type="string"></CpuModel>
                <CpuVendor type="string"></CpuVendor>
                <DeviceType type="string"></DeviceType>
                <Locale type="string"></Locale>
                <MachineModel type="string"></MachineModel>
                <MachineName type="string"></MachineName>
                <MachineSN type="string"></MachineSN>
                <MachineUUID type="string"></MachineUUID>
                <MachineVendor type="string"></MachineVendor>
                <OsName type="string"></OsName>
                <OsUserName type="string"></OsUserName>
                <OsVersion type="string"></OsVersion>
              </Device>
              <!---->
              <Location id="0" name="id">
```

```
                <Address type="string"></Address>
                <City type="string"></City>
                <Country type="string"></Country>
                <IpAddress type="string"></IpAddress>
                <Latitude type="string"></Latitude>
                <Longitude type="string"></Longitude>
                <ScoreAddress type="string"></ScoreAddress>
                <ScoreCity type="string"></ScoreCity>
                <ScoreCountry type="string"></ScoreCountry>
                <ScoreIP type="string"></ScoreIP>
                <ScoreState type="string"></ScoreState>
                <ScoreZipcode type="string"></ScoreZipcode>
                <State type="string"></State>
                <Zipcode type="string"></Zipcode>
              </Location>
            </Configuration>
          </Configurations>
        </Rule>
        <!---->
        <Rule name="ApprovalSetup">
          <ExternalLength type="int">6</ExternalLength>
          <Approval name="6">
            <Enabled type="bool">false</Enabled>
            <Address type="string"></Address>
            <Description type="string"></Description>
            <Prompt type="string">6</Prompt>
            <Type type="string">SMS</Type>
          </Approval>
        </Rule>
        <!---->
        <Rule name="ScheduleSetup">
          <Enabled type="bool">true</Enabled>
          <DaySunday type="bool">true</DaySunday>
          <DayMonday type="bool">true</DayMonday>
          <DayTuesday type="bool">true</DayTuesday>
          <DayWednesday type="bool">true</DayWednesday>
          <DayThursday type="bool">true</DayThursday>
          <DayFriday type="bool">true</DayFriday>
          <DaySaturday type="bool">true</DaySaturday>
          <StartDay type="int">-1</StartDay>
          <StartHour type="int">-1</StartHour>
          <StartMinute type="int">-1</StartMinute>
          <StartMonth type="int">-1</StartMonth>
          <StartYear type="int">-1</StartYear>
          <EndDay type="int">-1</EndDay>
          <EndHour type="int">-1</EndHour>
          <EndMinute type="int">-1</EndMinute>
          <EndMonth type="int">-1</EndMonth>
          <EndYear type="int">-1</EndYear>
        </Rule>
      </Rules>
    </User>
  </Users>
</ID>
```

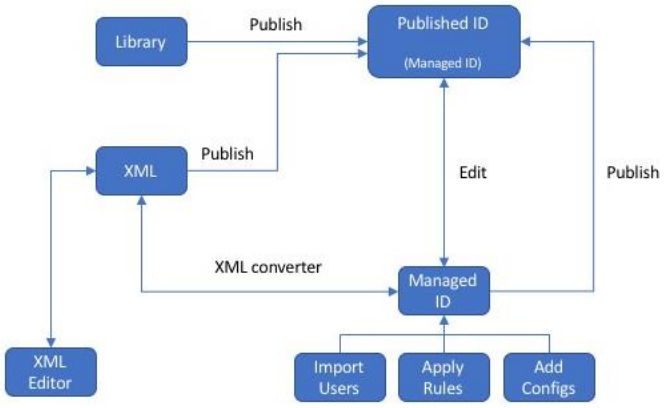
4.1.1.3 Example of a Machine Configuration XML Schema

The following represents a configuration XML schema:

```
<?xml version="1.0"?>
<!---->
<!-- ID Configuration -->
<!-- Date: 2018-03-15T10:12:34 -->
<!---->
<ID>
  <Configurations>
    <Configuration id="3196130144" name="HalethorpeMD-MacBookPro11,5">
      <Enabled type="bool">true</Enabled>
      <!---->
```

```
      <Device id="3906982300" name="MacBookPro11,5">
        <Architecture type="string">x86_64</Architecture>
        <CpuModel type="string">Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz</CpuModel>
        <CpuVendor type="string">GenuineIntel</CpuVendor>
        <DeviceType type="string">Mobile Device</DeviceType>
        <MachineModel type="string">MacBookPro11,5</MachineModel>
        <MachineSN type="string">1234567</MachineSN>
        <MachineUUID type="string">C43376B5-AAAA-5418-BAB2-FB77DSDF</MachineUUID>
        <MachineVendor type="string">Apple</MachineVendor>
        <OsName type="string">MacOSX</OsName>
        <OsUserName type="string">muser</OsUserName>
        <OsVersion type="string">10.12.6</OsVersion>
      </Device>
      <!---->
      <Location id="861180259" name="HalethorpeMD">
        <Address type="string"></Address>
        <City type="string">Halethorpe</City>
        <Country type="string">US</Country>
        <IpAddress type="string">107.1.244.110/10.10.1.2/172.20.1.31</IpAddress>
        <Latitude type="string">39.195</Latitude>
        <Longitude type="string">-76.6687</Longitude>
        <ScoreAddress type="string">0</ScoreAddress>
        <ScoreCity type="string">0</ScoreCity>
        <ScoreCountry type="string">0</ScoreCountry>
        <ScoreIP type="string">0</ScoreIP>
        <ScoreState type="string">0</ScoreState>
        <ScoreZipcode type="string">0</ScoreZipcode>
        <State type="string">MD</State>
        <TimeDiff type="string">58</TimeDiff>
        <Timestamp type="string">1521123095946</Timestamp>
        <Zipcode type="string">21227</Zipcode>
      </Location>
    </Configuration>
  </Configurations>
</ID>
```

Figure 1 - Managed ID Data Flow



4.1.1.4 Example of UXP ID Creation

- Load XML from file

```
bytearray b1, b2, b3;

b1 = file::readAll("MyDoc.xml");
```

- Add current machine fingerprint to document

```
b2 = id::getConfiguration();
b1 = id::addConfiguration(b1, b2, “*”);
```

- Apply rule preset to document

```
b1 = id::applyRulesByName(b1, "WorkflowMachine");
```

- Create ID from XML document

```
b3 = id::newIdFromDocument(b1);
```

- Save ID to file

```
file::writeAll(“id.iic”,b3,”replace”);
```

Rule Presets

A rule preset is a set of parameters that are used to construct a **UXP ID**. The underlying format of the preset is an XML document; however, the **Workflow Assistant** typically manages the presets using a structure editor.

When a **UXP ID** is being defined within the **Workflow Assistant**, a preset can be used to fill in the initial values for current rules. The user, if authorized, can then accept the initial values or change them to desired settings.

The **UXP Object** API can also create a **UXP ID** using the preset by name or by XML document. A named preset must be defined within the **Workflow Assistant**. The preset approach will allow a developer to programmatically construct custom **UXP IDs** without using the **Workflow Assistant**.

The following tables describe supported rules:

Table 7 – AlertSetup Rule Attributes

Attribute	Datatype	Description
EmailAddress	String	Specifies the email address to which alerts can be delivered.
IncludeDevice	Boolean	If true, an alert will contain the sender’s device fingerprint.
IncludeLicense	Boolean	If true, the alert will contain the sender’s Sertainty license information.
IncludeLocation	Boolean	If true, an alert will contain the sender’s network location.
SMSAddress	String	Specifies the SMS address to which alerts can be delivered. The address is actually the service providers email address used to deliver SMS messages.
UseEmail	Boolean	If true, alerts will be sent to the email as specified in the attribute **EmailAddress**.
UseSMS	Boolean	If true, alerts will be sent to the email as specified in the attribute **SMSAddress**.

Table 8 – ApprovalSetup Rule Attributes

Attribute	Datatype	Description
ApprovalEmail 'N'	String	Specifies the email address to which an approval request will be made. The recipient will be presented with a code that must be used when authenticating the user. **'N'** must be a number from 1 to 5. **Note**: there can be up to five email approvals.
ApprovalEmail 'N' Description	String	Description that will be included in the approval email. **'N'** must be a number from 1 to 5.
ApprovalEmail 'N' Prompt	String	Prompt that will be presented at authentication time.
ApprovalSMS 'N'	String	Specifies the service provider email address to which an SMS approval request will be made. The recipient will be presented with a code that must be used when authenticating the user. **'N'** must be a number from 1 to 5. **Note**: there can be up to five SMS approvals.
ApprovalSMS 'N' Description	String	Description that will be included in the approval SMS message. **'N'** must be a number from 1 to 5.
ApprovalSMS 'N' Prompt	String	Prompt that will be presented at authentication time. **'N'** must be a number from 1 to 5.
ExternalLength	Number	Specifies the length in characters of the approval codes. Approval codes are randomly generated numbers.

Table 9 – EventSetup Rule Attributes

Attribute	Datatype	Description
EmailAddress	String	Optional email address to which all events for the **UXP Object** will be sent.
ExternalKey	String	A key that must be provided to the API when recording events to an external callback routine. The external routine must use the key to decode the event data.
FileSpec	String	Optional file specification to which all events for the **UXP Object** will be written. If the output file is a **UXP Object**, the events will be recorded using the **UXP Object** journaling feature. To record events to standard output, use **console**: as the file specification.
FtpURL	String	Unsupported
LogAccesses	Boolean	If true, any access to **UXP Object** data will trigger an event.

Attribute	Datatype	Description
LogFailures	Boolean	If true, an authentication failure will trigger an event.
LogMessages	Boolean	If true, any external messages sent from the **UXP Object** will trigger and event.
LogRepeats	Boolean	If true, repeated unsuccessful authentication attempts will trigger an event.
RecordEmail	Boolean	If true, events will be sent to the specified email address.
RecordExternal	Boolean	If true, events will be sent to an external callback routine.
RecordFile	Boolean	If true, events will be sent to the file specified in **FileSpec**.
RecordFtp	Boolean	Unsupported
RecordLocal	Boolean	If true, events will be recorded within the **UXP Object**. The **UXP Object** must not be read-only.
RecordRemote	Boolean	If true, events will be sent to a reachable Data Services server.
RecordsSMS	Boolean	If true, events will be send to the specified SMS address.
RemoteURL	String	Contains the URL of a Data Services server to receive events.
SMSAddress	String	Optional SMS email address to which all events for the **UXP Object** will be sent. The address is actually the owner's mobile service provider address that will transform an email into an SMS message.

Table 10 – GeneralSetup Rule Attributes

Attribute	Datatype	Description
AdvancedDataLogging	Boolean	If true, debugging information will be written to the Sertainty log for the current application.
AlternateReality	String	Specifies an alternate workspace within the **UXP Object**. The default workspace is Public. Alternate realities require a special license.
Compliance	Number	Specifies the number of days until the **UXP Object** will automatically destroyed. The day value is relative to the date at which the **UXP Object** is created.
MaximumAccesses	Number	Specifies the maximum number of times the **UXP Object** can be authenticated. A zero indicates unlimited.
MaximumCycleFailures	Number	Specifies the maximum authentication attempts per session.
MaximumIdelTime	Number	Specifies the maximum number of seconds that a **UXP Object** can be idle. If idle time exceeds the time, the **UXP Object** will be closed.
MaximumTotalFailures	Number	Specifies the maximum number of failed authentication attempts without a successful authentication. If the number is reached, the **UXP Object** will self-destruct.

Attribute	Datatype	Description
UseLocalTime	Boolean	If true, the **UXP Object** will consider local machine date and time to be trusted. Otherwise, date and time will only be trusted if acquired from a trusted timer server.
Workflow	Boolean	If true, the **UXP Object** will not permit interactive authentication. All authentication challenges and responses must be provided ahead of the first authentication attempt.

Table 12 – RestrictionsSetup Rule Attributes

Attribute	Datatype	Description
ConfigurationAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current hardware and network location combination is unrecognized.
ConfigurationApproval	Boolean	Require an approval if the current hardware and network location combination is unrecognized.
ConfigurationDeny	Boolean	Deny access if the current hardware and network location combination is unrecognized.
ConfigurationDestroy	Boolean	Initiate a self-destruct if the current hardware and network location combination is unrecognized.
ConfigurationPrompts	Number	Add the specified number of challenges if the current hardware and network location combination is unrecognized.
EveryAuthenticationApproval	Boolean	Require an approval for every authentication attempt.
EveryAuthenticationPrompts	Number	Add the specified number of challenges for every authentication attempt.
HardwareAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current hardware is unrecognized.
HardwareApproval	Boolean	Require an approval if the current hardware is unrecognized.
HardwareDeny	Boolean	Deny access if the current hardware is unrecognized.
HardwareDestroy	Boolean	Initiate a self-destruct if the current hardware is unrecognized.
HardwarePrompts	Number	Add the specified number of challenges if the current hardware is unrecognized.
MovementAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current **UXP Object** file location is unrecognized.
MovementApproval	Boolean	Require an approval if the current **UXP Object** file location is unrecognized.
MovementDeny	Boolean	Deny access if the current network location is unrecognized.
MovementDestroy	Boolean	Initiate a self-destruct if the current **UXP Object** file location is unrecognized.

Attribute	Datatype	Description
MovementPrompts	Number	Add the specified number of challenges if the current network location is unrecognized.
NetworkAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current network location is unrecognized.
NetworkApproval	Boolean	Require an approval if the current network location is unrecognized.
NetworkDeny	Boolean	Deny access if the current network location is unrecognized.
NetworkDestroy	Boolean	Initiate a self-destruct if the current network location is unrecognized.
NetworkPrompts	Number	Add the specified number of challenges if the current network location is unrecognized.
Preset	String	Specifies the Sertainty rule preset that the current set of rules closely matches. This is purely informational and does not affect **UXP Object** construction.
ReadOnlyAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current **UXP Object** is read-only.
ReadOnlyApproval	Boolean	Require an approval if the current **UXP Object** is read-only.
ReadOnlyDeny	Boolean	Deny access if the current **UXP Object** is read-only.
ReadOnlyDestroy	Boolean	Initiate a self-destruct if the current **UXP Object** is read-only.
ReadOnlyPrompts	Number	Add the specified number of challenges if the current **UXP Object** is read-only.
ScheduleAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current **UXP Object** violates the schedule access window.
ScheduleApproval	Boolean	Require an approval if the current **UXP Object** violates the schedule access window.
ScheduleDeny	Boolean	Deny access if the current **UXP Object** violates the schedule access window.
ScheduleDestroy	Boolean	Initiate a self-destruct if the current **UXP Object** violates the schedule access window.
SchedulePrompts	Number	Add the specified number of challenges if the current **UXP Object** violates the schedule access window.
UntrustedSystemAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current operating system is an untrusted virtual machine.
UntrustedSystemApproval	Boolean	Require an approval if the current operating system is an untrusted virtual machine.
UntrustedSystemDeny	Boolean	Deny access if the current **UXP Object** operating system is an untrusted virtual machine.
UntrustedSystemDestroy	Boolean	Initiate a self-destruct if the current **UXP Object** operating system is an untrusted virtual machine.

Attribute	Datatype	Description
UntrustedSystemPrompts	Number	Add the specified number of challenges if the current **UXP Object** operating system is an untrusted virtual machine.
UntrustedTimeAltReality	Boolean	Force the **UXP Object** into an alternate workspace if the current date and time cannot be verified.
UntrustedTimeApproval	Boolean	Require an approval if the current date and time cannot be verified.
UntrustedTimeDeny	Boolean	Deny access if the current date and time cannot be verified.
UntrustedTimeDestroy	Boolean	Initiate a self-destruct if the current date and time cannot be verified.
UntrustedTimePrompts	Number	Add the specified number of challenges if the current date and time cannot be verified.

Table 13 – ScheduleSetup Rule Attributes

Attribute	Datatype	Description
DayFriday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DayMonday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DaySaturday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DaySunday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DayThursday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DayTuesday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
DayWednesday	Boolean	If false. **UXP Object** access will not be permitted on this day of the week.
Enabled	Boolean	If true, the scedule window is enforced.
EndDay	Number	Specifies the day of the month on which **UXP Object** access ends. A -1 indicates any day.
EndHour	Number	Specifies the hour of the day on which **UXP Object** access ends. A -1 indicates any hour.
EndMinute	Number	Specifies the minute of the hour on which **UXP Object** access ends. A -1 indicates any minute.
EndMonth	Number	Specifies the month of the year on which **UXP Object** access ends. A -1 indicates any month.
EndYear	Number	Specifies the year on which **UXP Object** access ends. A -1 indicates any year.

Attribute	Datatype	Description
StartDay	Number	Specifies the day of the month on which **UXP Object** access starts. A -1 indicates any day.
StartHour	Number	Specifies the hour of the day on which **UXP Object** access starts. A -1 indicates any hour.
StartMinute	Number	Specifies the minute of the hour on which **UXP Object** access starts. A -1 indicates any minute.
StartMonth	Number	Specifies the month of the year on which **UXP Object** access starts. A -1 indicates any month.
StartYear	Number	Specifies the year on which **UXP Object** access starts. A -1 indicates any year.

Table 14 – UserAdvancedSetup Rule Attributes

Attribute	Datatype	Description
MaximumTime	Number	Specifies the maximum number of seconds that will be tolerated when responding to each challenge. A zero indicates no limit.
MaximumTotalTime	Number	Specifies the maximum total number of seconds that will be tolerated when responding to all challenges. A zero indicates no limit.
MinimumTime	Number	Specifies the minimum number of seconds that will be tolerated when responding to each challenge. A zero indicates no minimum.
MinimumTotalTime	Number	Specifies the minimum total number of seconds that will be tolerated when responding to all challenges. A zero indicates no minimum.

Table 15 – UserBasicSetup Rule Attributes

Attribute	Datatype	Description
IgnoreCase	Boolean	If true, challenge response values will match regardless of letter case.
IgnoreChars	String	Specifies characters that will be ignored when provided as part of the challenge response value.

Table 16 – UserRecoverySetup Rule Attributes

Attribute	Datatype	Description
MaximumFailures	Number	Specifies the maximum number of failed challenges that will be tolerated during an authentication recovery

Attribute	Datatype	Description
		scenario. To permit authentication, this value must be met as well as the **MinimumCorrect** value.
MinimumCorrect	Number	Specifies the minimum number of correct challenge responses that must be met in order to authentication during a recovery scenario.

The following table describes where a rule and its attributes can be applied:

Table 17 – Rule application

Rule	ID	User
AlertSetup	Yes	No
ApprovalSetup	Yes	Yes
ConfigurationSetup	Yes	Yes
EventSetup	Yes	No
GeneralSetup	Yes	No
ScheduleSetup	Yes	Yes
RestrictionsSetup	Yes	No
UserBasicSetup	No	Yes
UserAdvancedSetup	No	Yes
UserRecoverySetup	No	Yes

The format of the preset XML document is:

```
<?xml version="1.0"?>
<!--Rule Presets-->
<RulePresets>
  <LastEditBy>MacOSX/gsmith</LastEditBy>
  <LastEditDate>03/15/2018 10:35</LastEditDate>

  <!---->
  <Preset name="Company Confidential" position="3">
    <!---->
    <Rule name="AlertSetup">
      <EmailAddress type="string"></EmailAddress>
      <IncludeDevice type="bool">true</IncludeDevice>
      <IncludeLicense type="bool">false</IncludeLicense>
      <IncludeLocation type="bool">true</IncludeLocation>
      <SMSAddress type="string"></SMSAddress>
      <UseEmail type="bool">false</UseEmail>
      <UseSMS type="bool">false</UseSMS>
    </Rule>
    <!---->
    <Rule name="ApprovalSetup">
      <ApprovalEmail 1 type="string"></ApprovalEmail 1>
      <ApprovalEmail 1 Description type="string"></ApprovalEmail 1 Description>
      <ApprovalEmail 1 Prompt type="string"></ApprovalEmail 1 Prompt>
      <ApprovalEmail 2 type="string"></ApprovalEmail 2>
      <ApprovalEmail 2 Description type="string"></ApprovalEmail 2 Description>
      <ApprovalEmail 2 Prompt type="string"></ApprovalEmail 2 Prompt>
      <ApprovalEmail 3 type="string"></ApprovalEmail 3>
      <ApprovalEmail 3 Description type="string"></ApprovalEmail 3 Description>
```

```
    <ApprovalEmail 3 Prompt type="string"></ApprovalEmail 3 Prompt>
    <ApprovalEmail 4 type="string"></ApprovalEmail 4>
    <ApprovalEmail 4 Description type="string"></ApprovalEmail 4 Description>
    <ApprovalEmail 4 Prompt type="string"></ApprovalEmail 4 Prompt>
    <ApprovalEmail 5 type="string"></ApprovalEmail 5>
    <ApprovalEmail 5 Description type="string"></ApprovalEmail 5 Description>
    <ApprovalEmail 5 Prompt type="string"></ApprovalEmail 5 Prompt>
    <ApprovalSMS 1 type="string"></ApprovalSMS 1>
    <ApprovalSMS 1 Description type="string"></ApprovalSMS 1 Description>
    <ApprovalSMS 1 Prompt type="string"></ApprovalSMS 1 Prompt>
    <ApprovalSMS 2 type="string"></ApprovalSMS 2>
    <ApprovalSMS 2 Description type="string"></ApprovalSMS 2 Description>
    <ApprovalSMS 2 Prompt type="string"></ApprovalSMS 2 Prompt>
    <ApprovalSMS 3 type="string"></ApprovalSMS 3>
    <ApprovalSMS 3 Description type="string"></ApprovalSMS 3 Description>
    <ApprovalSMS 3 Prompt type="string"></ApprovalSMS 3 Prompt>
    <ApprovalSMS 4 type="string"></ApprovalSMS 4>
    <ApprovalSMS 4 Description type="string"></ApprovalSMS 4 Description>
    <ApprovalSMS 4 Prompt type="string"></ApprovalSMS 4 Prompt>
    <ApprovalSMS 5 type="string"></ApprovalSMS 5>
    <ApprovalSMS 5 Description type="string"></ApprovalSMS 5 Description>
    <ApprovalSMS 5 Prompt type="string"></ApprovalSMS 5 Prompt>
    <ExternalLength type="int">6</ExternalLength>
  </Rule>
  <!---->
  <Rule name="ConfigurationSetup">
    <ConfigurationFile 1 type="string"></ConfigurationFile 1>
    <ConfigurationFile 2 type="string"></ConfigurationFile 2>
    <ConfigurationFile 3 type="string"></ConfigurationFile 3>
    <ConfigurationFile 4 type="string"></ConfigurationFile 4>
    <ConfigurationFile 5 type="string"></ConfigurationFile 5>
    <ReplaceConfigurations type="bool">true</ReplaceConfigurations>
  </Rule>
  <!---->
  <Rule name="EventSetup">
    <EmailAddress type="string"></EmailAddress>
    <ExternalKey type="string"></ExternalKey>
    <LogAccesses type="bool">true</LogAccesses>
    <LogFailures type="bool">true</LogFailures>
    <LogMessages type="bool">true</LogMessages>
    <LogRepeats type="bool">true</LogRepeats>
    <RecordEmail type="bool">false</RecordEmail>
    <RecordExternal type="bool">false</RecordExternal>
    <RecordLocal type="bool">true</RecordLocal>
    <RecordRemote type="bool">false</RecordRemote>
    <RecordSMS type="bool">false</RecordSMS>
    <SMSAddress type="string"></SMSAddress>
    <URL type="string"></URL>
  </Rule>
  <!---->
  <Rule name="GeneralSetup">
    <AdvancedDataLogging type="bool">true</AdvancedDataLogging>
    <AlternateReality type="string"></AlternateReality>
    <Compliance type="date">03/14/2118 20:00</Compliance>
    <MaximumAccesses type="int">0</MaximumAccesses>
    <MaximumCycleFailures type="int">4</MaximumCycleFailures>
    <MaximumIdleTime type="int">99</MaximumIdleTime>
    <MaximumTotalFailures type="int">0</MaximumTotalFailures>
    <UseLocalTime type="bool">false</UseLocalTime>
    <Workflow type="bool">false</Workflow>
  </Rule>
  <!---->
  <Rule name="MemberPrivileges">
    <Copy type="bool">true</Copy>
    <Delete type="bool">true</Delete>
    <Owner type="bool">true</Owner>
    <Print type="bool">true</Print>
    <Read type="bool">true</Read>
    <Read Events type="bool">true</Read Events>
    <Read Signature type="bool">true</Read Signature>
    <Sign type="bool">true</Sign>
    <Write type="bool">true</Write>
  </Rule>
```

```
<!---->
<Rule name="RestrictionsSetup">
  <ConfigurationAltReality type="bool">false</ConfigurationAltReality>
  <ConfigurationApproval type="bool">false</ConfigurationApproval>
  <ConfigurationDeny type="bool">false</ConfigurationDeny>
  <ConfigurationDestroy type="bool">false</ConfigurationDestroy>
  <ConfigurationPrompts type="int">0</ConfigurationPrompts>
  <EveryAuthenticationApproval type="bool">false</EveryAuthenticationApproval>
  <EveryAuthenticationPrompts type="int">3</EveryAuthenticationPrompts>
  <HardwareAltReality type="bool">false</HardwareAltReality>
  <HardwareApproval type="bool">false</HardwareApproval>
  <HardwareDeny type="bool">false</HardwareDeny>
  <HardwareDestroy type="bool">false</HardwareDestroy>
  <HardwarePrompts type="int">0</HardwarePrompts>
  <MovementAltReality type="bool">false</MovementAltReality>
  <MovementApproval type="bool">false</MovementApproval>
  <MovementDeny type="bool">false</MovementDeny>
  <MovementDestroy type="bool">false</MovementDestroy>
  <MovementPrompts type="int">0</MovementPrompts>
  <NetworkAltReality type="bool">false</NetworkAltReality>
  <NetworkApproval type="bool">false</NetworkApproval>
  <NetworkDeny type="bool">false</NetworkDeny>
  <NetworkDestroy type="bool">false</NetworkDestroy>
  <NetworkPrompts type="int">0</NetworkPrompts>
  <Preset type="string"></Preset>
  <ReadOnlyAltReality type="bool">false</ReadOnlyAltReality>
  <ReadOnlyApproval type="bool">false</ReadOnlyApproval>
  <ReadOnlyDeny type="bool">false</ReadOnlyDeny>
  <ReadOnlyPrompts type="int">0</ReadOnlyPrompts>
  <ScheduleAltReality type="bool">false</ScheduleAltReality>
  <ScheduleApproval type="bool">false</ScheduleApproval>
  <ScheduleDeny type="bool">false</ScheduleDeny>
  <ScheduleDestroy type="bool">false</ScheduleDestroy>
  <SchedulePrompts type="int">0</SchedulePrompts>
  <UntrustedSystemAltReality type="bool">false</UntrustedSystemAltReality>
  <UntrustedSystemApproval type="bool">false</UntrustedSystemApproval>
  <UntrustedSystemDeny type="bool">false</UntrustedSystemDeny>
  <UntrustedSystemDestroy type="bool">false</UntrustedSystemDestroy>
  <UntrustedSystemPrompts type="int">0</UntrustedSystemPrompts>
  <UntrustedTimeAltReality type="bool">false</UntrustedTimeAltReality>
  <UntrustedTimeApproval type="bool">false</UntrustedTimeApproval>
  <UntrustedTimeDeny type="bool">false</UntrustedTimeDeny>
  <UntrustedTimeDestroy type="bool">false</UntrustedTimeDestroy>
  <UntrustedTimePrompts type="int">0</UntrustedTimePrompts>
</Rule>
<!---->
<Rule name="ScheduleSetup">
  <DayFriday type="bool">true</DayFriday>
  <DayMonday type="bool">true</DayMonday>
  <DaySaturday type="bool">true</DaySaturday>
  <DaySunday type="bool">true</DaySunday>
  <DayThursday type="bool">true</DayThursday>
  <DayTuesday type="bool">true</DayTuesday>
  <DayWednesday type="bool">true</DayWednesday>
  <Enabled type="bool">false</Enabled>
  <EndDay type="int">-1</EndDay>
  <EndHour type="int">-1</EndHour>
  <EndMinute type="int">-1</EndMinute>
  <EndMonth type="int">-1</EndMonth>
  <EndYear type="int">-1</EndYear>
  <StartDay type="int">-1</StartDay>
  <StartHour type="int">-1</StartHour>
  <StartMinute type="int">-1</StartMinute>
  <StartMonth type="int">-1</StartMonth>
  <StartYear type="int">-1</StartYear>
</Rule>
<!---->
<Rule name="UserAdvancedSetup">
  <MaximumTime type="int">30</MaximumTime>
  <MaximumTotalTime type="int">99</MaximumTotalTime>
  <MinimumTime type="int">0</MinimumTime>
  <MinimumTotalTime type="int">0</MinimumTotalTime>
</Rule>
```

```
      <!---->
      <Rule name="UserBasicSetup">
        <IgnoreCase type="bool">true</IgnoreCase>
        <IgnoreChars type="string"></IgnoreChars>
        <MinimumPrompts type="int">1</MinimumPrompts>
      </Rule>
      <!---->
      <Rule name="UserRecoverySetup">
        <MaximumFailures type="int">1</MaximumFailures>
        <MinimumCorrect type="int">5</MinimumCorrect>
      </Rule>
    </Preset>
  </RulePresets>
```

4.2 Single-Sign-On

A **UXP ID** can be used to establish a **single-sign-on** within the user's session. A **single-sign-on** is a special authentication that will follow the same rules and policies as defined by the **UXP iD**. Where it differs from typical **UXP Object** authentication is that it can become a proxy and represent the current user in subsequent **UXP Object** sessions.

> **Example:** If one successfully signs on as me@myplace.com and a **UXP Object** is opened, the user can allow the current **UXP ID** to attempt to authenticate access on the user's behalf. If the **UXP ID** cannot establish a valid connection to the **UXP Object**, the conventional authentication process will be invoked.

A **single sign-on** session can be set up as a local session, valid only for the current application, and global, which will allow other processes to utilize the session.

Single-sign-on works for conventional **UXP Object** authentication as well as the **Sertainty Drive Technology**.

5 Sertainty Drive

Sertainty Drive is a technology extension that facilitates direct access to **UXP Object** data via the conventional operating system file system. Using this path, an application does not have to understand or even be aware of the **UXP Technology**. It would see accessible protected data just as it would see unprotected data that may reside in a folder.

On macOSX and Linux, the Drive is based on FUSE. For Windows, a proprietary driver is installed, requiring a reboot prior to invoking the open drive operation.

Sertainty mobile libraries do not support opening a **UXP Object** as a drive.

Benefits of using a **Sertainty Drive**:

- Protected folders and files would be accessible via standard utilities.

  Example: On Windows, the Explorer would see a drive as a standard random-access device with an assigned drive letter. On MacOS X, the Finder would see a Drive as a mounted external drive. The mounted **UXP Object** would be similar to the effect of inserting a USB thumb drive and accessing the contents via utilities.

- A drive can be closed or ejected just like any other external drive. The **Workflow Assistant** can also close drives from within the application.
- Applications do not require any code changes to access protected data. Normal operations, such as creating new files, reading or updating existing files, are supported.
- Data files remain protected even though they appear as normal files in your system.
- **UXP Objects** can be mounted in read-only mode to prevent any changes via conventional applications.
- When a **UXP Object** is mounted, authentication occurs exactly like it would if opening it via a **UXP-aware application**.

Limitations of the **Drive**:

- Write-once architecture **UXP Objects** can only be mounted read-only. Read-write architecture **UXP Objects** allow full read-write activities.
- When a **UXP Object** is mounted as a **Drive** device, the access is restricted to the current operating system user. If another user is logged into the system, the drive would not be visible to that user.
- A drive requires exclusive access to the **UXP Object**. This means that a UXP**-aware application**, such as the **Workflow Assistant** cannot open the same **UXP Object** that is mounted as a **Drive** device.
- The only way to mount a **UXP Object** as a drive is using the **Sertainty Workflow Assistant** or the **Drive Utility**. Future releases of the drive technology may introduce alternate methods of mounting a **UXP Object**.
- Advanced features of the **UXP Object**, such as file access restrictions cannot be changed via the **Drive**. All advanced operations must be performed using the **Workflow Assistant** or through the **UXP Object** API.
- A **Drive** will not timeout. Even though a **UXP Object** may have an idle-time restriction, the **Drive** will not permit the **UXP Object** to automatically close. This restriction may be removed in a future release.