Sertainty

KCL Developer Guide

Version: V3.6.0

Table of Contents

1 KCL Code Program

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, **UXP Engine** embeds active technology within a protected Object to prevent unauthorized access no matter where the document resides.

1.1 KCL Code Module

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

- **Custom KCL**
  KCL is a proprietary C-like language that allows for a flexible way of constructing the **KCL Code** program. 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.

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

1.1.1 UXP Engine Module Control using KCL

Knowledge-Control-Language (**KCL**) are executable procedures that follow the **UXP Object**, no matter where the **UXP Object** resides. The advantage of this approach is to apply behavior rules within the **UXP Object** without having to build the knowledge into the **UXP Engine**.

Potential benefits of rules:

- **KCL Code** can access cloaked **UXP ID** artifacts in order to validate the client without exposing the unprotected data to world.
- **KCL Code** follow the **UXP Object**. **KCL Code** can be written in such a way as to allow the **UXP Object** to be offline and still be fully protected against unauthorized access.
- **KCL Code** add layered protection. For example, if the **UXP Object** represents a protected document that is meant to read by a single client within one hour of creation, a rule can implement the timer and issue a self-destruct mechanism.
- **KCL Code** can be compiled independently and can then be used to protect many **UXP Objects**, or **KCL Code** can be unique to each **UXP Object**.

A **KCL procedure** has the following format:

```
datatype funcName(optional-argument-list)
{
  procedure-specific-code;

  return result;
}
```

A **KCL rule** has the following format:

```
datatype ruleName()
{
  rule-specific-code;

  return result;
}
```

Technically, a rule and a procedure are very similar. A procedure can accept function arguments from the caller, where as a rule cannot. Rules and procedures support the same **KCL Code** features, otherwise.

KCL Code can be manually created, if it contains the required components. To effectively code a **KCL Identity** and **Governance**, the user must fully understand basic program development concepts.

1.2 Required and Recommended User Rules

When constructing custom **KCL Code** for a **UXP Object**, the following procedures should be noted:

- **Authentication::fileAccess()**

  This rule is called when the authorized user attempts to access a virtual file. The rule can determine if the user can access the file at this point.

  If the procedure is not defined, fine grained file access will be disabled.

- **Authentication::main()**

  This routine is call for any authentication operation. It is responsible for interacting with the **UXP Engine** to identify a valid user. It is also responsible for setting the appropriate access status and access privileges for a valid user.

  This procedure is required.

- **Authentication::userSetup()**

  This rule is called prior to any authentication. The rule can typically define global settings for the **UXP Object**.

  This procedure is optional.

- **Compliance::main()**

  This rule is only called when a **UXP Object** accessed via the compliance interface.

  This procedure is optional.

- **NewAppliance::setup()**

  When a **UXP Object** is created, the owner must supply credential via this **KCL** procedure.

  This procedure is required.

- **Touch::main()**

  This rule is only called when a **UXP Object** accessed via the touch interface. Touching a **UXP Object** is a simple access point that executes rules that are common to all users. For example, a touch routine would validate global device and location configurations, compliance and global schedule violations.

  This procedure is optional.

1.3 KCL Code Workflow

When a **UXP Object** is created or access, the user's **KCL Code** routines may interact with the **UXP Engine**. The following execution points interact with the **KCL Code** routines:

1.3.1 UXP Object Creation

To create a new **UXP Object**, the user must provide **KCL Code** before adding any documents to the **UXP Object**. The **KCL Code** must contain required routines to be valid.

Once loaded, the **UXP Engine** calls the **KCL Code** routine **NewAppliance::setup** to retrieve authentication data. The routine must define credential and challenge data for any user that will access the **UXP Object** after creation time. Upon return, the routine is never called again.

1.3.2 Opening an Existing UXP Object

Prior to any activity, the **UXP Object** will invoke the KCL 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 **UXP 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 your **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.

1.3.3 Sample Native Construction Flow using KCL

Included in the SDK Examples folder are two source files that demonstrate how to create a **UXP Object** from both 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++ interfaces are also available.

a. **Validate the library license**

When a **UXP** entity-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** entity function successfully.

The call is:

```
if (!uxpsys_initializeLibrary(error-buffer,
                              argc,
                              argv,
                              “*”,
                              “Sertainty”))
 {
   handle error
 }
```

b. **Create a KCL Code set and compile the file**

KCL Code can be unique to a **UXP Object** or it can be designed as a shareable set of procedures for many objects. In either case, the format of the **KCL Code** is the same.

```
Uxpsys_compileKcl(errors, “mycode.kcb”, “mycode.kcl”, defines, macros, 0)
```

We now have a compiled **KCL Code** file called mycode.kcb.

Note: The **defines** argument is an optional list of comma-separated strings containing keywords that trigger conditional compilation using #ifdef / #endif techniques.

The macros argument is a list of uxlVariableHandle items, where each handle is a macro substitution candidate for the source **KCL Code** file.

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”, “mycode.kcb”, KclFile,
                    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 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** routines to determine who is trying to access the **UXP Object**.

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

If the **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 met, the user is granted access to the **UXP Object** with the privileges defined for the matching credential.

If the user incorrectly responded, the **KCL Code** can decide to challenge the user with more questions 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 KCL Language Extensions

KCL is a special extension to the **UXL language** as defined in the **Workflow Guide**. **KCL** artifacts can only be used by **KCL Code** within the **UXP Object**; and **UXL** extensions from the Workflow Guide can only be used in **UXL** scripts outside of the **UXP Object**. Other than function access restrictions, the syntax and language concepts are identical.

2.1 Built-In Structure Types

The following data types are automatically defined by the system:

Table 11 – Built-In Data Types

Type Name	Member Name	Data Type	Description
ChallengeType		Structure	Defines the credential challenge data.
	Name	String	Name of the challenge
	Prompt	String	User prompt
	Value	String	Required response for the challenge.
	UseDataMask	Integer	0 – Not used, 1 – Used
	DataType	Integer	Data type of the challenge
	SubType	Integer	Sub-type of the challenge
	FormatType	Integer	Formatting type for the challenge presentation.
	TimeLow	Integer	Low value for response window. A zero disables the low boundary.
	TimeHigh	Integer	High value for the response window. A zero disables the high boundary.
	Required	Integer	0 – The challenge is not required, 1 – The challenge is required every time a user opens the UXP entity.
	Address	String	E-mail address or SMS phone number for external challenges and approvals. Ignored by other challenge sub-types.

			Multiple addresses or phone numbers may be entered by separating the items with a semi-colon character. A challenge with multiple addresses or phone numbers would send the same challenge or approval information to each target listed. The authentication process would require the code from at least one of the target addresses or phone numbers.
	Key	String	Unique key for external challenges and approvals. The key differentiates multiple external challenges. Ignored by other challenge sub-types.

Type Name	Member Name	Data Type	Description
ConfigType		Structure	Defines a simple address configuration.
	DeviceID	Integer	Contains a device identifier.
	FilePaths	String	Contains zero or more valid folder paths at which the UXP entity may reside. Multiple paths are separated by '|' characters.
	LocationID	Integer	Contains a network location identifier.
ScheduleType		Structure	Defines the credential schedule data.
	Sunday	Integer	0 – disable access, 1 – enable access.
	Monday	Integer	0 – disable access, 1 – enable access.
	Tuesday	Integer	0 – disable access, 1 – enable access.
	Wednesday	Integer	0 – disable access, 1 – enable access.
	Thursday	Integer	0 – disable access, 1 – enable access.
	Friday	Integer	0 – disable access, 1 – enable access.
	Saturday	Integer	0 – disable access, 1 – enable access.
	StartMinute	Integer	Indicates starting minute for access. A -1 indicates no restrictions. Valid range is 0 to 59.
	StartHour	Integer	Indicates starting hour for access. A -1 indicates no restrictions. Valid range is 0 to 23.
	StartDay	Integer	Indicates starting day for access. A -1 indicates no restrictions. Valid range is 1 to number of days in month.

	StartMonth	Integer	Indicates starting month for access. A -1 indicates no restrictions. Valid range is 1 to 12.
	StartYear	Integer	Indicates starting year for access. A -1 indicates no restrictions. Value must be greater than or equal to current year.
	EndMinute	Integer	Indicates ending minute for access. A -1 indicates no restrictions. Valid range is 0 to 59.
	EndHour	Integer	Indicates ending hour for access. A -1 indicates no restrictions. Valid range is 0 to 23.
	EndDay	Integer	Indicates ending day for access. A -1 indicates no restrictions. Valid range is 1 to number of days in month.
	EndMonth	Integer	Indicates ending month for access. A -1 indicates no restrictions. Valid range is 1 to 12.
	EndYear	Integer	Indicates ending year for access. A -1 indicates no restrictions. Value must be greater than or equal to current year.
	Enabled	Integer	Indicates whether the schedule is enabled.

Type Name	Member Name	Data Type	Description
CredentialType		Structure	Defines a user credential.
	Id	Integer	UXP-assigned identifier for the user.
	Name	String	Full user name.
	E-mail	String	E-mail address for the user.
	ConfigCount	Integer	The number of approved configurations for the credential.
	DataMask	String	Used to make challenges dynamic. The mask is constructed as a comma-separated list of masking codes.
	Schedule	Schedule Type	Access schedule for this user.
	TimeLow	Integer	Low value for response window. A zero disables the low boundary.
	TimeHigh	Integer	High value for the response window. A zero disables the high boundary.
	ValidationType	Integer	
	Privileges	Integer	Granted privileges upon authorized access.
	Challenges	List	List of ChallengeType entries.

2.2 Built-In Variables

The following variables are automatically defined by the system:

Table 12 – Built-In Variables

Variable Name	Data Type	Description
AccessCopy	Integer	A constant containing the internal access privilege value. The privilege grants the ability to export and copy virtual files.
AccessCount	Integer	Maintains the total number of access since object creation.
AccessDelete	Integer	A constant containing the internal access privilege value. The privilege grants the ability to delete virtual files.
AccessNone	Integer	A constant containing the internal access privilege value.
AccessOwner	Integer	A constant containing the internal access privilege value. The privilege grants the ability to control the entire **UXP Object** virtual files.
AccessPrint	Integer	A constant containing the internal access privilege value. The privilege grants the ability to print virtual file contents. Presently, not supported.
AccessRead	Integer	A constant containing the internal access privilege value. The privilege grants the ability to read virtual files. **Note**: The **UXP Engine** requires every user to have **AccessRead** privilege, regardless of the privilege settings.
AccessReadEvent	Integer	A constant containing the internal access privilege value. The privilege grants the ability to read event data.
AccessReadSignature	Integer	A constant containing the internal access privilege value. The privilege grants the ability to read virtual signatures.
AccessUnlimited	Integer	A constant containing the internal access privilege value. The privilege grants the ability control the entire **UXP Object.**

Variable Name	Data Type	Description
AccessWrite	Integer	A constant containing the internal access privilege value. The privilege grants the ability to create and update virtual files.
AdvancedThreatDetection	Integer	Flag to enable an algorithm to detect break-in attempts using your username. The detection is **UXP Object**-independent, meaning that the analytics occur over time and against any **UXP Object.**that has a particular username. Valid values are: • Disable threat detection • Enable threat detection
AlertDevice	Integer	A constant indicating that device fingerprint information will be included with all alerts.
AlertLocation	Integer	A constant indicating that location fingerprint information will be included with all alerts.
AlertOptions	Integer	Contains bit settings for alerts. The possible bit values are: • **AlertDevice** • **AlertLocation**
AllowedMisses	Integer	Within a single authentication pass, this value specifies the number of invalid challenge responses that can be tolerated and still permit access. To be considered, the user must provide a number of valid challenge responses as specified with the **AllowedMissesThreshold**.
AllowedMissesThreshold	Integer	Within a single authentication pass, this value specifies the number of valid challenge responses that must be provided before considering the **AllowedMisses** value. Only local challenges are counted towards the threshold. External challenges and approvals must always be valid.
AuthenticationStatus	Integer	Contains the current authentication status. Authentication status values are bit masks that permit multiple concurrent status scenarios. For

Variable Name	Data Type	Description
		example, a user may be at an invalid location and violate the schedule window at the same time. This variable will contain the bit mask for both status values. The value is set when the intrinsic function **validateUser** is called.
AuthorizedUserId	Integer	Contains the credential identifier of the current authorized user.
CloneConfigurations	Integer	A constant containing the internal bit value for including configuration, device and location data into a cloned **UXP Object**.
CloneEvent	Integer	A constant containing the internal bit value for including event data into a cloned **UXP Object**.
CloneSignatures	Integer	A constant containing the internal bit value for including signature data into a cloned **UXP Object**.
CloneStatistics	Integer	A constant containing the internal bit value for including access statistics data into a cloned **UXP Object**.
CloneUserData	Integer	A constant containing the internal bit value for including user data into a cloned **UXP Object**.
ConfigId	Integer	Contains the configuration signature for the current access.
CurrentFileAction	String	Contains the action associated with a virtual file access attempt. Possible values are: • Copy • Delete • Directory • Read • Rename • Write
CurrentFileId	Integer	Contains the current **UXP Object** file identifier. If the original **UXP Object** was moved or copied, this value should be different from the value stored in UxpFileId.
CurrentUsername	String	Contains the current user's USERNAME challenge value.
CurrentVirtualFile	String	Contains the current virtual file name.

Variable Name	Data Type	Description
CurrentVirtualFileAccess	Integer	Contains the access setting for the current user and virtual file. Possible values are: • **AccessNone** • **AccessUnlimited**
DataTypeDate	Integer	A constant containing the internal date data type value.
DataTypeFloat	Integer	A constant containing the internal float data type value.
DataTypeInteger	Integer	A constant containing the internal integer data type value.
DataTypeMultiChoice	Integer	A constant containing the internal multiple-choice data type value.
DataTypeString	Integer	A constant containing the internal string data type value.
DeviceCount	Integer	Specifies the number of approved devices for the **UXP Object**.
DeviceId	String	Contains the device signature for the current access.
DeviceType	String	Contains the current device type for the current access.
EmailAuthentication	Integer	Specifies SMTP authentication. Valid values are: • **Authentication disabled** • **Authentication enabled**
EmailPort	Integer	Specifies the SMTP port number used to send E-mail.
EmailPwd	String	Specifies the require password for the SMTP server.
EmailReplyTo	String	Specifies the return address for sending E-mail.
EmailSecurity	String	Specifies the type of security for the current SMTP server. Valid values are: • **NONE** • **SSL** • **TLS**
EmailServer	String	Specifies the SMTP server used to send E-mail messages from within the **UXP Object**.
EmailUser	String	Specifies the required username for the SMTP server.

Variable Name	Data Type	Description
EnableDebugging	Integer	Enables debug logging for **KCL** authentication. Should always be disabled unless advanced debugging is desired. Valid values are: • **Debug disabled** • **Debug enabled**
error	Integer	Contains the value indicating error.
errorCode	Integer	After a procedure call, this will contain the error code. A zero indicates no error has occurred.
errorString	String	After a procedure call, this will contain the error message. An empty string indicates no error has occurred.
EventAccess	Integer	A constant containing the internal bit value for recording an event entry for every **UXP Object** access.
EventEmail	Integer	A constant containing the internal bit value for recording an event entry by sending the data to the email address specified in **EventEmailAddress**. . Event data is sent in clear form.
EventEmailAddress	String	Specifies the email address to use when **EventEmail** option is selected.
EventExternal	Integer	A constant containing the internal bit value for recording an event entry using the external user callback. 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	Integer	A constant containing the internal bit value for recording an event entry for every **UXP Object** access failure.
EventFile	Integer	A constant containing the internal bit value Indicates an event entry will be recorded in a local file.

Variable Name	Data Type	Description
		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	String	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. **Example**: $(HOME)/$(UXP_FILENAME).log would become /home/myhome/foo.log
EventFtp	Integer	Indicates an event entry will be recorded at a remote FTP site (Not yet implemented). **EventFtpURL** must contain the server URL.
EventFtpURL	Integer	Indicates the remote ftp server URL.
EventLicense	Integer	A constant containing the internal bit value for including license information in every event entry.
EventLocal	Integer	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	Integer	A constant containing the internal bit value for recording and event entry for

Variable Name	Data Type	Description
		any external E-mail and SMS messages that are sent.
EventOptions	Integer	Contains bit settings for recording event data. The possible bit values are: • **EventAccess** • **EventEmail** • **EventExternal** • **EventFailure** • **EventFile** • **EventLicense** • **EventLocal** • **EventMessages** • **EventRemote** • **EventSMS**
EventRemote	Integer	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	Integer	Indicates an event entry will be recorded for repeated failed authentication attempts.
EventSecureKey	String	Key used to protect event data when sent to remote or external targets. It is recommended that the key be a set of random characters having a length of at least 50 characters. This option is used in conjunction with the **EventExternal** option. **Note:** This key must be used when attempting to read an external event record. The API for event logging contains a **SetKey** routine.
EventSMS	Integer	A constant containing the internal bit value for recording an event entry by sending the data to the email address specified in **EventSmsAddress**. . Event data is sent in clear form.

Variable Name	Data Type	Description
EventSmsAddress	String	A valid SMS phone number or address to be used when sending event data via SMS
EventURL	String	When the event option **EventRemote** is set, this variable specifies either a local file specification or a server URL that will receive event data.
ExternalChallengeLength	Integer	Sets the length of a randomly generated challenge string sent to a user's E-mail or phone.
False	Integer	Contains the value 0.
FormatTypeNatural	Integer	A constant containing the internal format type value for simple challenge phrases.
IgnoreCase	Integer	Challenge response sensitivity flag. Valid values are: • **Challenge responses are case sensitive** • **Challenge responses are case insensitive**
IgnoreCharacters	String	Sets a set of characters that will be ignored when comparing a user's answers to a challenge response. Can be any printable character.
IsWorkflow	Integer	Indicates whether the **UXP Object** is limited to workflow applications. Not a workflow **UXP Object** A workflow **UXP Object** When set, authentication will not prompt an external user for any challenge responses. All required responses must be provided prior to calling the authentication routine.
Locale	String	Contains the current locale for the current access.
LocationCount	Integer	Specifies the number of approved locations for the **UXP Object**.
LocationId	String	Contains the location signature for the current access.
MaskAmPm	Integer	A constant containing the internal mask value. The mask value entered by the user must be either AM or PM.

Variable Name	Data Type	Description
MaskDay	Integer	A constant containing the internal mask value. The user must enter the day of the month.
MaskHour12	Integer	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	Integer	A constant containing the internal mask value. The user must enter the current hour in 24-hour format where midnight to 1AM is 0 and 11PM to midnight is 23.
MaskLastMonth	Integer	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	Integer	A constant containing the internal mask value. The user must enter the four-digit year for last year.
MaskMonth	Integer	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	Integer	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	Integer	A constant containing the internal mask value. The user must enter the four-digit year for next year.
MaskToday	Integer	A constant containing the internal mask value. The user must enter the day of the month.
MaskTomorrow	Integer	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.

Variable Name	Data Type	Description
MaskUserData	Integer	A constant containing the internal mask value. The user must enter the actual challenge value.
MaskYear	Integer	A constant containing the internal mask value. The user must enter the current four-digit year.
MaskYesterday	Integer	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.
MasterKey	String	A string value used to validate the **KCL** code with the user’s protected data. It can be any string of printable characters up to 40 characters in length. The value is required.
MaximumIdleTime	Integer	Specifies the maximum number of seconds that an open **UXP Object** can remain idle. If non-zero, an expiration will initiate a close operation.
Missing	Integer	Contains the value indicating missing value.
Missing_str	String	Contains the value indicating missing string value.
No_action	Integer	Contains the value indicating no action.
Not_found	Integer	Contains the value indicating data not found.
PanicEmailAddress	String	Contains the email address to alert when panic mode is enabled.
PanicSMSAddress	String	Contains the address or phone to alert when panic mode is enabled.
PanicWord	String	Contains a keyword when used as a response to a challenge. The panic word can be used by any user and any challenge.
Procedure_name	String	Contains the name of the currently executing procedure.
ReadOnly	Integer	Contains either a zero or a one to indicate the status of the current **UXP Object**. A zero indicates a regular read-write object. A one value indicates the **UXP Object** is read-only.

Variable Name	Data Type	Description
SessionFailureCount	Integer	Number of failures for this access attempt.
SMSPwd	String	Specifies the account password used to connect to the SMS service.
SMSService	String	Specifies whether SMS service provider. Supported services are: Message Media Service
SMSUser	String	Specifies the account used to connect to the SMS service.
StatusAuthorized	Integer	A constant that indicates the user was granted access to the **UXP Object**.
StatusBitMax	Integer	A constant that indicates the maximum number of status bits used by functions returning the authentication status.
StatusChallenged	Integer	A constant that indicates the user was found but requires more challenges before access is granted.
StatusConfigFound	Integer	A constant containing configuration status. This value indicates the device and location combination were recognized for the current user.
StatusConfigNotFound	Integer	A constant containing configuration status. This value indicates neither the device nor the location was recognized for the current user.
StatusDeviceFound	Integer	A constant containing configuration status This value indicates the device was recognized for the current user but not the location.
StatusDeviceLocationFound	Integer	A constant containing configuration status. This value indicates the device and the location were recognized for the current user but not as a combination.
StatusInvalidUsername	Integer	A constant that indicates the user name was not recognized.
StatusLocationFound	Integer	A constant containing configuration status. This value indicates the location was recognized for the current user but not the device.
StatusNotAuthorized	Integer	A constant that indicates the user was denied access to the **UXP Object**.
StatusPanic	Integer	A constant that indicates the user is in panic mode.

Variable Name	Data Type	Description
StatusScheduleViolation	Integer	A constant that indicates the user was recognized, but the current date and time violates the access schedule for the user.
SubTypeExternalPhrase	Integer	A constant containing the internal subtype value for external challenge phrases.
SubTypeUserPhrase	Integer	A constant containing the internal subtype value for user challenge phrases.
Success	Integer	Contains the value indicating success.
TotalFailureCount	Integer	Number of consecutive failures since the last valid access. When a valid user successfully authenticates, this value is reset to zero.
True	Integer	Contains the value 1.
UntrustedDevice	Integer	Deprecated
UntrustedLocation	Integer	Deprecated
UntrustedSystem	Integer	Contains a 1 if the current system data is considered unreliable. Typically a system is considered not trustworthy when the current device is running a remote desktop application that would mislead location validation.
UntrustedTime	Integer	Contains a 1 if the current timestamp data is considered unreliable. Typically a timestamp is considered not trustworthy when the current time cannot be validated by a trusted remote time-server.
UseLocalTime	Integer	Contains a 1 if the current environment should use local time for the trusted time. When set to 1, the system assumes time is trusted even though it may not use the trusted remote time-server.
UxpCredential	UxpCredentialType	A structure containing the current authenticated user credential.
UxpCredentialList	List	Contains all defined credentials for the **UXP Object**. For new **UXP Object** setup, this list must contain all the credential values for the object. For

Variable Name	Data Type	Description
		authentication, this list is optional, but can be populated by the call **getCredentialList**.
UxpFileId	Integer	Contains the **UXP Object** file identifier when the **UXP Object** was created.
UxpUXP entity	Structure	A structure containing the primary attributes of the current **UXP Object**.
ValidationLocal	Integer	A constant containing the internal validation type value for local credential validation.
ValidationRemote	Integer	A constant containing the internal validation type value for remote credential validation via an identity delegate.

All event 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”;
```

2.3 KCL Code Functions

In addition to intrinsic functions described in the **Workflow Guide**, the following procedures are specific to **UXP Object** operations and can only be used in **KCL Code** authentication and access modules.

Table – 13 UXP Engine Functions

Procedure	Description
addCommonConfiguration	Adds the specified configuration to the list of acceptable configurations for all users
addConfiguration	Adds the current configuration to the list of acceptable configurations for the current user.
addChallenge	Adds the specified challenge name as a challenge for the current user. The user is then prompted for the correct response to the challenge.
addRandomChallenge	Adds a random challenge for the current user. The user is then prompted for the correct response to the challenge.
defineChallenge	Defines the specified challenge for a credential. The challenge structure must contain the base challenge data.

Procedure	Description
defineConfiguration	Defines the specified default configuration for a credential. The configuration structure must contain the base configuration data.
defineCredential	Defines the specified credential for an object. The credential structure must contain the base credential data and the schedule data.
destroy	Destroys the internal data for the **UXP Object**.
downloadUpdates	Downloads authorized updates the **UXP Object** from a known Sertainty server.
getChallengeCount	Gets the number of challenges that have been selected for presentation to the user.
getComplianceDate	Gets the compliance date/time for the UXP.
getCreationDate	Gets the UXP creation date/time.
getCredentialList	Gets the current list of credentials and places data in the built-in variable **UxpCredentialList**.
getFileFlags	Retrieves the ACL flags for a virtual file.
hasConfiguration	Determines if the current authentication process has any defined configurations.
isCommonConfiguration	Determines if the current configuration is a certified common configuration.
isOwner	Determines if current user is the owner of the **UXP Object**.
isScheduleViolation	Determines if the current object access is valid with respect to the specified schedule.
readVariable	Reads the variable from within the **UXP Object**.
recordEvent	Records a custom event with the **UXP Object**. The event will be delivered based on the event settings within the **UXP Object.**
sendEmail	Sends an E-mail.
sendFtp	Sends a copy of the current UXP entity to the specified FTP server.
sendSMS	Sends an SMS text message.
setAuthorization	Sets the authorization for access to the protected data for the current user.
setReality	Sets the current reality.
signUxp	Signs the current **UXP Object**.
uploadEvent	Uploads the event history for the **UXP Object** to a known Sertainty server.
uxpVariable	Gets the specified external **UXP Object** variable value
uxpVariableExists	Determines if the specified external **UXP Object** variable exists.

Procedure	Description
validateUser	Evaluates the current user based on user-supplied challenge responses.
virtualFileExists	Determines if the virtual file exists in the current **UXP Object**.
writeVariable	Saves the value as a persistent variable within the **UXP Object**.

addCommonConfiguration (config)

Adds the specified configuration to the list of acceptable configurations for all users. This simplifies configuration management by setting a set of fixed configurations for all users.

Parameters:

config	A **UxpConfig** structure containing the desired configuration identifier, device identifier and location identifier. If omitted, the current configuration is added.

addConfiguration ( )

Adds the current configuration to the list of acceptable configurations for the current user.

Parameters:

None

addChallenge (name)

Adds the specified challenge name as a challenge for the current user. The user is then prompted for the correct response to the challenge.

Parameters:

name	The name of the challenge as defined for this user.

addRandomChallenge (count)

Adds a random challenge for the current user. The user is then prompted for the correct response to the challenge.

Parameters:

count	Specifies the number of challenges to add.

defineChallenge (credential, challenge)

Defines the specified challenge for a credential. The challenge structure must contain the base challenge data.

Parameters:

credential	The name of the UxpCredentialType structure containing credential data.
Challenge	The name of the UxpChallengeType structure containing challenge data.

defineConfiguration (credential, config)

Defines the specified default configuration for a credential. The configuration structure must contain the base configuration data.

A default configuration permits the owner of the object to defined known device and locations for a user. Typically, when a user attempts to open a **UXP Object** on a new device or at a new location, the system will challenge the user to verify identity. By pre-defining configurations, a user already appears to be "known" to the **UXP Object**.

Parameters:

credential	The name of the **UxpCredentialType** structure containing credential data.
Config	The name of the **UxpConfigType** structure containing configuration data.

defineCredential (credential)

Defines the specified credential for a **UXP Object**. The credential structure must contain the base credential data and the schedule data. Challenges and default configurations can also be included or added at a later time.

Parameters:

credential	The name of the **UxpCredentialType** structure containing credential data.

Destroy ( )

Destroys the internal data for the **UXP Object**. This is **irreversible**.

downloadUpdates ( )

Downloads authorized updates the **UXP Object** from a known Sertainty server. (Not presently implemented)

int getChallengeCount ( )

Gets the number of challenges that have been selected for presentation to the user.

Returns:

Number of challenges currently selected for the user.

date getComplianceDate (def-date)

Gets the compliance date and time for the **UXP Object**. A date can come from two places: the base Identity when the **UXP Object** was created. This is the legacy value. The second comes from **UXP Object**-based compliance data. If the compliance data is not present or is not set, then the default date is returned. If the compliance settings contain an expiration date/time, then that value is returned.

Parameters:

def-date	The default compliance date/time as defined in the original UXP identity.

Returns:

Compliance date and time.

date getCreationDate ( )

Gets the date and time when the **UXP Object** was created.

Parameters:

None

Returns:

Creation date and time.

getCredentialList ( )

Gets the current list of credentials and places data in the built-in variable **UxpCredentialList**.

Int getFileFlags (virtual-file)

Retrieves the ACL flags for a virtual file. File flags are a bitmask with the following values:

2	File is Read-only
4	File is Read-write
32	Send SMS message on access
64	Send email message on access
256	No access
8192	No movement
32768	Requires **UXP Object** signature

Parameters:

virtual-file	Virtual file path to test.

Returns:

ACL file flags.

Int hasConfiguration (is-global , option)

Determines if the current authentication process has any defined configurations.

Parameters:

is-global	True if testing global configuration list. False if testing the current user's configuration list.
Option	Determine what portion of the configuration is tested. Possible values are: **1 - Entire configuration** **2 - Device only** 3 - **Location only**

Returns:

1 if there are existing configurations. 0 if there are no existing configurations.

Int isCommonConfiguration ( )

Determines if the current configuration is a certified common configuration.

Parameters:

None

Returns:

1 if current configuration is certified. 0 if the current configuration has not be certified by the **UXP Object**.

Int isOwner ()

Determines if current user is the owner of the **UXP Object**.

Returns:

1 if the user is the owner. 0 if the user is not the owner.

Int isScheduleViolation (sched)

Determines if the current object access is valid with respect to the specified schedule.

Parameters:

sched	**ScheduleType** structure containing a schedule definition.

Returns:

1 if schedule has been violated. 0 if access is permitted.

readVariable (name , variable)

Reads the variable from within the **UXP Object**. If the variable is not found, an empty string is returned. A persistent variable can be used to dynamically save data from within the **KCL Code** routine.

Parameters:

name	Name of the persistent variable.
Variable	The variable to receive the persistent data.

recordEvent (action, status, msg [, variables])

Records a custom event with the **UXP Object**. The event will be delivered based on the event settings within the **UXP Object**.

Parameters:

action	Action value for the event. Possible values range from 22 to 50. Values 0 to 21 are reserved for use by the UXP engine.
status	Any status value as defined by the user. Typically, the status values are UXP entity error codes.
msg	A string to be recorded within the event.
variables	A list of optional variables that will be added to existing event variables. A variable has a name and a value. The list is a value-pairs string, separated by a '|' character. **Example**: MobileDevice=iPhone|IPhone Version=V4.5.6

sendEmail (to , subject , body [, secure , incUxp , options , attName])

Sends an E-mail.

Parameters:

to	The address of the E-mail recipient.
Subject	The subject of the E-mail.
Body	The body of the E-mail.

Secure	A Boolean flag indicating the E-mail should be sent as a **UXP Object** attachment. The attached UXP Object will contain the Identity as the current UXP Object. The body of the E-mail will be a document called **E-mail** within the UXP Object. If the **incUxp** flag is set, the attachment will also contain the UXP Object copy. The flag is optional. The default value is False.
incUxp	A Boolean flag indicating the E-mail should include a UXP Object copy of the current UXP Object. The **options** parameter controls what elements of the current UXP Object will be included. The flag is optional. The default value is False.
Options	The options that control what data is included in the UXP Object copy. Using the following built-in variables, the user can combine the values into a single integer: **CloneEvent** Copies all event entries to the new object. **CloneConfigurations** Copies all configuration, device and location data to the new object. **CloneSignatures** Copies all signature data to the new object. **CloneStatistics** Copies the access statistics to the new object. **CloneUserData** Copies all user data to the new object. Note, copying user data may increase the size of the new object and subsequently prevent successful delivery. The parameter is optional. The default value is 0.
attName	The name to be given the attachment, if either the **secure** or **incUxp** flags are set.

sendFtp (output-name , server , user , password , port , options)

Sends a copy of the current **UXP Object** to the specified FTP server.

Note: this is currently disabled due to technical restrictions.

Parameters:

output-name	The output file name to which the object will be saved.
Server	The FTP server IP address.
User	The username as required for authentication at the FTP server.
Password	The password as required for authentication at the FTP server.
Port	The port to use for connecting to the specified FTP server.
Options	The options that control what data will be included in the **UXP Object** copy. Using the following built-in variables, the user can combine the values into a single integer: **CloneEvent** Copies all event entries to the new object. **CloneConfigurations** Copies all configuration, device and location data to the new object. **CloneSignatures** Copies all signature data to the new object. **CloneStatistics** Copies the access statistics to the new object. **CloneUserData** Copies all user data to the **UXP Object**. **Note:** copying user data may increase the size of the new object and subsequently prevent successful delivery.

sendSMS (to , message [, incConfig])

Sends an SMS text message. The message will include a minimal amount of UXP device and location data.

Parameters:

to	The phone number of the recipient.
Message	Message to send
incConfig	Optional Boolean to indicate the desire to include minimal configuration data in the message. The default is **False**.

setAuthorization (status , priv)

Sets the authorization for access to the protected data for the current user.

Parameters:

status	The status of the access. Possible values are: **0 - NotAuthorized** **1 - Authorized** 2 - **Challenged**
priv	The privileges granted to the user, if authorized. Possible bit values are: • **AccessAltRealities** • **AccessCopy** • **AccessDelete** • **AccessNone** • **AccessOwner** • **AccessPrint** • **AccessRead** • **AccessReadEvent** • **AccessReadSignature** • **AccessSign** • **AccessUnlimited** • **AccessWrite**

setReality (name)

Sets the current reality within the **UXP Object**.

Parameters:

name	Name of the reality.

Note: only available with **AltReality** privileges.

signUxp (key , data)

Signs the current **UXP Object** with a permanent timestamp, configuration and user-specified data. A **UXP Object** can be signed any number of times by the current authorized user.

Parameters:

key	Optional key information that can tie the signature to a particular event.
Data	Optional data that can be stored in the signature. The data can be anything the owner requires.

uploadEvents ( )

Uploads the event history for the **UXP Object** to a known Sertainty server. (Not presently implemented)

string uxpVariable (expr)

Gets the specified external UXP variable value.

Parameters:

name	Variable to find. The variable name must be quoted.

Returns:

The variable value as a string.

Int uxpVariableExists (expr)

Determines if the specified external **UXP Object** variable exists.

Parameters:

name	Variable to find. The variable name must be quoted.

Returns:

1 if variable exists. 0 if variable does not exist.

Int validateUser( )

Evaluates the current user based on user-supplied challenge responses. The returned status is actually a bit mask that can indicate multiple status values concurrently. For example, a user may be violating a schedule and have an unrecognized configuration. In this example, the return status value would be both **StatusConfigNotFound** and **StatusScheduleViolation**.

In some cases, a status may only contain a single status bit set.

Parameters:

None

Returns:

The following status values can be returned by this call:

StatusConfigNotFound

Indicates neither the device nor the location was recognized for this user.

StatusDeviceFound

Indicates the device was recognized for this user, but the location was not.

StatusDeviceLocationFound

Indicates the device and the location were recognized for this user, however, not as combination.

StatusInvalidUsername

Indicates the user name as not valid.

StatusLdapViolation

Indicates a remove LDAP directory services lookup failed.

StatusLocationFound

Indicates the location was recognized for this user, but the device was not.

StatusPanic

Indicates the user is in panic mode. The **KCL** can then decide what to do. Typically, the code should send an alert to an alternate contact, who can then call an appropriate authority.

StatusScheduleViolation

Indicates the current time does not occur within the specified schedule for this user.

Int virtualFileExists (name)

Determines if the virtual file exists in the current **UXP Object**.

Parameters:

name	Virtual file name to find.

Returns:

1 if virtual file exists. 0 if virtual file does not exist.

writeVariable (name , value)

Saves the value as a persistent variable within the **UXP Object**. The variable can be read at the next authentication point. A persistent variable can be used to dynamically save data from within the **KCL Code** routine.

Parameters:

name	Name of the persistent variable.
Value	The data to save in the **UXP Object**. The maximum size is 500 bytes.

2.4 Example UXP Engine KCL Code Module

The following is a simple example of a complete **UXP Engine** module:

```
/*! \copyright Sertainty Corporation, 2016.  All Rights Reserved.

    \file     sample.kcl

    \details  Sample kcl code source

    \author   gsmith
    \date     1/01/2012
 */

/**
 * Called by UXP Object when the user wishes to initialize Object structures
 * from fixed settings within this file.
 */
rule NewAppliance::setup()
{
  depends on rule commonSetup(success);

  CredentialType cred;
  ChallengeType ch;
  ConfigType config;

#ifdef TEST
  config.ConfigId = ConfigId;
  config.DeviceId = DeviceId;
  config.LocationId = LocationId;

  addCommonConfiguration(config);
#endif

  /* Define the base credential */

  cred.Name = "Jon Smith";
  cred.TimeLow = 0;
  cred.TimeHigh = 300;
  cred.ValidationType = ValidationLocal;
  cred.Privileges = AccessOwner;
  cred.Email = "your@emailsys.com";

  cred.Schedule.Sunday = cred.Schedule.Saturday = true;
  cred.Schedule.Monday = cred.Schedule.Tuesday = cred.Schedule.Wednesday = true;
  cred.Schedule.Thursday = cred.Schedule.Friday = true;
  cred.Schedule.StartHour = 8;
  cred.Schedule.EndHour = 23;
  cred.Schedule.StartMinute = cred.Schedule.EndMinute = -1;
  cred.Schedule.StartDay = cred.Schedule.EndDay = -1;
  cred.Schedule.StartMonth = cred.Schedule.EndMonth = -1;
  cred.Schedule.StartYear = cred.Schedule.EndYear = -1;

  defineCredential(cred);

  /* Add challenges */

  ch.Name = "USERNAME";
  ch.Prompt = "User";
  ch.Value = "Test";
  ch.TimeLow = 0;
  ch.TimeHigh = 10;
  ch.DataType = DataTypeString;
  ch.SubType = SubTypeUserPhrase;
  ch.FormatType = FormatTypeNatural;
  ch.Required = true;

  defineChallenge(cred, ch);

  ch.Name = "TEST1";
  ch.Prompt = "Test 1";
  ch.Value = "test1";

  defineChallenge(cred, ch);

  ch.Name = "TEST2";
  ch.Prompt = "Test 2";
  ch.Value = "test2";

  defineChallenge(cred, ch);

  ch.Name = "TEST3";
  ch.Prompt = "Test 3";
  ch.Value = "test3";
  ch.Required = false;

  defineChallenge(cred, ch);

  ch.Name = "EMAILCHALLENGE";
  ch.Prompt = "Enter the code sent to email address (Test)";
  ch.Value = "";
  ch.Address = "your@emailsys.com";
  ch.Key = "TEST";
  ch.TimeLow = 0;
  ch.TimeHigh = 0;
  ch.SubType = SubTypeExternalEmail;

  defineChallenge(cred, ch);

  ch.Name = "EMAILAPPROVAL";
  ch.Prompt = "Enter the phrase sent to approver TEST";
  ch.Value = "";
  ch.Address = "your@emailsys.com";
  ch.Key = "TEST";
  ch.TimeLow = 0;
  ch.TimeHigh = 0;
  ch.SubType = SubTypeExternalEmailApproval;

  defineChallenge(cred, ch);

  ch.Name = "SMSCHALLENGE";
  ch.Prompt = "Enter the code sent to your phone (TEST)";
  ch.Value = "";
  ch.Address = "000000000";
  ch.Key = "TEST";
  ch.SubType = SubTypeExternalSMS;

  defineChallenge(cred, ch);

  ch.Name = "SMSAPPROVAL";
  ch.Prompt = "Enter the phrase sent to approver TEST";
  ch.Value = "";
  ch.Address = "000000000";
  ch.Key = "TEST";
  ch.SubType = SubTypeExternalSMSApproval;

  defineChallenge(cred, ch);
```

```
}

/**
 * Called prior to actual user authentication.  This gives the
 * owner a chance to setup global elements.
 */
rule Authentication::setup()
{
  depends on rule commonSetup(success);
}

/**
 * This routine is called whenever a user is tested for authenticity.  This
 * routine will continue to be called until the following occurs:
 *
 *    o StatusAuthorized is the authorization status.  The current user is
 *      now authorized to access the UXP and its data base on the
 *      designated privileges
 *
 *    o StatusNotAuthorized is the authorization status.  The current user
 *      is denied access and the UXP engine exits.
 *
 * All other authorization status values are interpreted as challenges to
 * the current user to prove identity.
 *
 * Prior to returning, the routine should call the setAuthorization
 * routine to indicate authentication status and assigned privileges.
 */
rule Authentication::main()
{
  /* These rules execute one time */

  depends on rule Authentication::checkReadOnly(Success);
  depends on rule Authentication::complianceCheck(Success);
  depends on rule Authentication::scheduleCheck(Success);
  depends on rule Authentication::locationCheck(Success);

  /* These rules execute upon entry and return */

  on entry Authentication::checkFailures();
  on return Authentication::checkFailures();

  /* Find the user credential.  If found, UxpCredential is set.
   *
   * The return status will contain one or more status bits set.  For
   * that reason, we must loop through the bits to determine which
   * status conditions must be addressed for the next authentication pass.
   */

  int i, status = validateUser();

  for (i = 1; i <= StatusBitMax; i++)
  {
    switch (bitTest(status, i))
    {
      case 0:
        break;

      case StatusAuthorized:
        addConfiguration();
        setAuthorization(StatusAuthorized, UxpCredential.Privileges);
        return;

      /* Invalid username entered */
      case StatusInvalidUsername:
        return;

      /* Schedule violation */

      case StatusScheduleViolation:
        logFailure("UXP entity access attempted outside of valid time window for this user");
        return;

      /* Neither the device nor the location was recognized */

      case StatusConfigNotFound:
        addRandomChallenge(3);
        break;

      /* Device was recognized, but the location was not */

      case StatusDeviceFound:
        addRandomChallenge(2);
        break;

      /* Location recognized, device was not */

      case StatusLocationFound:
        addRandomChallenge(3);
        break;

      /* Device and location recognized, but not at the same time. */

      case StatusDeviceLocationFound:
        addRandomChallenge(1);
        break;
    }
  }

  /* Check for trusted elements */

#ifdef UNTRUSTED
  if (!TrustedDevice || !TrustedLocation || !TrustedTime || !TrustedSystem)
    addChallenge("SMSCHALLENGE");
#endif

  /* Add challenges equal to the current session failure count */

  addRandomChallenge(SessionFailureCount);

  /* Example: If object has been moved, add challenges */

#ifdef ENFORCE_CIPHER_LOCATION
  if (UxpFileId != CurrentFileId)
  {
    addRandomChallenge(3);
  }
#endif

  /* Requires approval by 2nd person */

#ifdef ADD_EXTERNAL_CHALLENGES
  addChallenge("EMAILAPPROVAL");
#endif

  /* Assume more challenges */

  setAuthorization(StatusChallenged, AccessNone);
  return;
}

/**
 * Checks session and total failure counts
 */
rule Authentication::checkFailures()
{
  if (TotalFailureCount >= 10)
  {
    logFailure("UXP entity shredded and destroyed due to attack");
    destroy();

    return (error);
  }

  if (SessionFailureCount >= 2)
  {
    logFailure("UXP entity access rejected due to excessive failures");

    return (error);
  }

  return (noaction);
}

/**
 * Rule to check compliance independent of the user
 */
rule Authentication::complianceCheck()
{
  depends on rule Compliance::main(success);

  if (today() > ComplianceExpirationDate)
  {
    logFailure(concat("UXP entity has expired as of ",
            ComplianceExpirationDate," and will be shredded."));
    destroy();

    return (error);
  }

  return (success);
}

/**
 * Rule to check if uxp is read-only
 */
rule Authentication::checkReadOnly()
{
  if (ReadOnly)
  {
    logFailure("UXP entity is read-only and cannot be opened.");

    return (error);
  }

  return (success);
}

/**
 * Rule to check a schedule independent of the user
 */
rule Authentication::scheduleCheck()
{
  ScheduleType sched;

  sched.Sunday = sched.Saturday = true;
  sched.Monday = sched.Tuesday = sched.Wednesday = true;
  sched.Thursday = sched.Friday = true;
  sched.StartHour = sched.EndHour = -1;
  sched.StartMinute = sched.EndMinute = -1;
  sched.StartDay = sched.EndDay = -1;
  sched.StartMonth = sched.EndMonth = -1;
  sched.StartYear = sched.EndYear = -1;

  if (isScheduleViolation(sched))
  {
    logFailure("UXP entity access attempted outside of valid time window");

    return (error);
  }

  return (success);
}

/**
 * Rule to check device/location independent of the user
 */
rule Authentication::locationCheck()
{
#ifdef ENFORCE_CIPHER_LOCATION
  if (UxpFileId != CurrentFileId)
  {
    logFailure("UXP entity has been illegally moved");

    return (error);
  }
#endif

#ifdef TEST
  if (!isCommonConfiguration())
  {
    logFailure("Invalid common configuration");

    return (error);
  }
#endif

  return (success);
}

/**
 * This rule is used to check compliance for the object
 *
 */
rule Compliance::main()
{
  ComplianceExpirationDate = toDate("01/10/2099 00:00");

  return (success);
}

/**
 * This routine is called whenever a virtual file is access.
 */
rule Authentication::fileAccess()
{

  if (CurrentFileAction != "Directory")
  {
    string buf = concat("Virtual file '", CurrentVirtualFile, "' accessed for ",
            CurrentFileAction, ": ", toString(today()), " (UTC)");

    sendEmail("your@emailsys.com", "Alert: UXP entity virtual file accessed", buf);
  }

  CurrentVirtualFileAccess = AccessUnlimited;

  return;
}

/**
 * A common routine to setup global elements.
 */
rule commonSetup()
{
  MasterKey = "a set of random characters";
  MaximumIdleTime = 300;
  ExternalChallengeLength = 7;
  IgnoreCase = true;
  EventOptions = EventFailure | EventLocal | EventAccess | EventExternal | EventMessages;
  AlertOptions = AlertLocation | AlertDevice;

  EmailReplyTo = "Your reply-to";
  EmailServer = "your smtp server address";
  EmailPort = 587;
  EmailSecurity = "TLS";
  EmailAuthentication = true;

  EmailSenderName = "Your Service";
  EmailUser = "your email account";
  EmailPwd = "your password";

  EventSecureKey = "a set of random characters";

  return (success);
}

/**
 * Logs a failure and sends email
 */
procedure logFailure(message)
{
  string buf = concat(message, ": ", toString(today()), " (UTC)");

  sendEmail("your@emailsys.com", "Alert: UXP entity access failure", buf);

  setAuthorization(StatusNotAuthorized, AccessNone, message);
}
```