Sertainty

UXP Scripting Guide

Version: V2.2.0

Table of Contents

1 Sertainty Script

UXP Engine scripting is used to construct the authentication and governance engine. The resulting script is **KCL Code**. When a **UXP Object** is created, a **UXP Engine** script, **KCL Code,** is required and must contain a specific set of defined functions. A special set of **UXL** functions are accessible only within this mode.

The **UXL Scripting Engine** permits a user to build batch-style scripts that can perform basic utility operations. Like the **UXP Engine**, the **UXL Script Engine** has a special set of **UXL** functions that are accessible with this mode.

1.1 Sertainty Scripting Utility, UXL

The **Scripting Utility** is command line tool that supports the **UXL Script Engine**.

A sample of supported operations:

- Create a new **UXP Object**
- Add files to a **UXP Object**
- Export protected data from an existing **UXP Object**
- View the status of an existing **UXP Object**
- View the event history of an existing **UXP Object**
- Compile **UXL** source into a protected binary form

To execute the utility:

UXP entity-SDK-location/UXP entity [options] [command]

Options:

-h, --help	Displays this help.
-b, --batch	Execute scripting command and exit utility.
-d, --drives	Shows current mounted drives.
-f, --file <file>	File to include using protect.
-g, --gui	Use dialogs for authentication.
-I, --id <id>	ID to use when protecting files.
-l, --log <logfile>	Records all output into <logfile>.
-n, --noscript	Do not invoke script engine.
-m, --mount <UXP entity>	Mounts a UXP entity.
-p, --protect <UXP entity>	Creates a UXP entity.
-r, --replace	Replace output file when using protect.
-s, --set <pref=value>	Sets a Sertainty preference.
-v, --version	Displays version information.
-x, --readonly	Mount UXP entity in read-only mode.
--p1 <p1>	Parameter passed into UXL engine.
--p2 <p2>	Parameter passed into UXL engine.
--p3 <p3>	Parameter passed into UXL engine.
--p4 <p4>	Parameter passed into UXL engine.
--p5 <p5>	Parameter passed into UXL engine.
--p6 <p6>	Parameter passed into UXL engine.
--p7 <p7>	Parameter passed into UXL engine.
--p8 <p8>	Parameter passed into UXL engine.
--p9 <p8>	Parameter passed into UXL engine.
--p10 <p8>	Parameter passed into UXL engine.

Arguments:

command	Optional UXL command to execute.

See **Scripting Functions** for descriptions of supported routines.

Example: Creating a new **UXP Object** using the interactive feature.

```
<Installation-location>/bin/SertaintyScript
uxl> int id;
uxl> id = sf::newUxp(“mymusic.uxp”, “myid.iic”, “IdFile”);
uxl> sf::setAttribute(id, “NAME”, “Music Backup”);
uxl> sf::setAttribute(id, “DESCRIPTION”, “Backup of my purchased songs”);
uxl> sf::setAttribute(id, “OWNER”, “Greg”);
uxl> sf::addFile(id, “song1.m4a”, “song1.m4a”, 10000, 2);
uxl> sf::addFile(id, “song2.m4a”, “song2.m4a”, 10000, 2);
uxl> sf::closeUxp(id);
uxl> exit
```

Example: Creating a new **UXP Object** using the single command line feature where the above commands are stored in a script file

```
<Installation-location>/bin /SertaintyScript -b @backup.uxl
```

In the above example, one creates a new **UXP Object** from one or more music files.

1.1.1 Initialization File

When the **Script Utility** starts, it looks for the file **scriptinit.uxl** in the **Sertainty home folder**. If found, it will attempt to execute the file. The execution is performed prior to any other script operations.

1.2 UXL Language

The **UXP Scripting Language** (**UXL**) starts as a sub-set of the C programming language and adds some proprietary constructs that enable security, portability and ease-of-use. The following sections describe the differences between C and **UXL**.

1.2.1 Comments

Like C, the /* and */ comment-block is supported. **UXL** also supports the C++ style comment of //.

1.2.2 Identifiers and Keywords

Variable and procedure names are case sensitive. All keywords and built-in procedure names are case-insensitive.

Due to optimization, all identifiers are converted to internal codes. As a result, a **UXL** error may return an error message containing an incoherent variable name if an exception occurs. To produce the same error without optimization, disable optimization by setting the **ModifierNoOptimize** at compilation or **UXP Object** creation time.

1.2.3 Data Types

The following data types are supported:

Table 1 – UXL Data Types

Data Type	Description
bytearray	Variable length array of bytes. Can handle text or binary data.
int	Integers are always 64bit and signed.
float	Floating point numbers are always double precision.
string	Character strings are always variable length and have no maximum size.
date	Standard date and time
list	A list is a variable length array containing any **UXL** type or user-defined type.

Syntax:

```
datatype variable-name, variable-name;
```

Like C, variables can be declared as arrays.

All other C data types are unsupported.

All data types will attempt to automatically convert data to the correct format.

> **Example:** If an integer variable is assigned a value from a string variable, **UXL** will attempt to convert the string to an integer. If the data cannot be converted, then an error will stop **UXL** execution.

Unlike conventional language compilers, **UXL** only supports variable scoping at two levels: **global** and **procedure**.

> **Example:** A C-language procedure allows one to create a variable within a logical block. Once the block ends, the variable will be deallocated.

For **UXL**, variables can be declared at any point within a procedure; however, they will be visible for the life of the procedure.

1.2.4 User-Defined Procedures and Rules

To declare a function in **UXL**, the following syntax must be used.

For procedures and functions:

```
[ REPLACE ] PROCEDURE [domain-name::]proc-name ( optional-arg-list )
{
  statement-list
}
```

A procedure behaves like a C-language procedure in that it can have formal arguments. Unlike C, **UXL** supports untyped arguments. An untyped argument means that a procedure can be passed any type variable without compiler restrictions. For example, a procedure expects one parameter. In typical C-language cases, the argument is a hard-wired type such as a string or a number, but in **UXL**, the data type can be evaluated at runtime. In **UXL**, the procedure can make decisions based on the data type of the argument.

As with argument passing, the **UXL** procedure can return any data type as a return argument.

For rules, the following syntax applies:

```
[ REPLACE ] RULE [domain-name::]rule-name ( )
{
  inference-statement-list
  statement-list
}
```

A rule is a special type procedure that accepts no formal arguments. It also can declare inference dependencies on other rules as well as data.

Inference is defined by the following statement syntax:

```
DEPENDS ON [RULE | PROCEDURE]  ( SUCCESS | ERROR | NOACTION | ENDOFDATA)
  {
    ON MATCH { statement-list }
    ON NOMATCH { statement-list }
    BEFORE INFERENCE { statement-list }
    AFTER INFERENCE { statement-list }
  }

DEPENDS ON DATA expression
  {
    ON MATCH { statement-list }
    ON NOMATCH { statement-list }
    BEFORE INFERENCE { statement-list }
    AFTER INFERENCE { statement-list }
  }
```

When a rule contains a dependency clause, the actual body of the rule will not execute until the dependencies are met. Conversely, if a rule or procedure is declared as a dependent, then execution may implicitly execute if the parent rule requires.

Note: Dependent rules will only execute if their current execution status is **NoAction**.

1.2.5 Macros and Conditional Compilation

The **UXL** language compiler supports several compile-time options that provide template features.

#ifdef

Syntax: `#ifdef` UXL-variable-name

UXL-code

#endif

Description: If **UXL-variable-name** exists, then compile all code that occurs until the next **#endif** occurs. The **#ifdef** and **#endif** must be the first item on the line to be valid.

#ifndef

Syntax: `#ifndef` UXL-variable-name

UXL-code

#endif

Description: If **UXL-variable-name** does not exist, then compile all code that occurs until the next **#endif** occurs. The **#ifndef** and **#endif** must be the first item on the line to be valid.

#define

Syntax: `#define` UXL-variable-name

Description: Defines the specified **UXL-variable-name** for the purpose of conditional compilation. The variable will not be compiled into the executable **UXL** code. The **#define** must be the first item on the line to be valid.

#undefine

Syntax: `#undefine` UXL-variable-name

Description: Undefines the specified **UXL-variable-name** for the purpose of the conditional compilation. The **#undefine** must be the first item on the line to be valid.

When compiling **UXL** source code, dynamic elements can be substituted for fixed placeholders. Using the **$(var-name)** token, the **UXL** compiler will substitute the contents of **var-name** for the entire placeholder. The substituted element can be anything that forms a valid **UXL** construct after substitution.

Example:

```
procedure test()
{
  String a = $(external-def-value);
}
```

To work properly, the user must pre-define the **UXL** variable **external-def-value** and populate it with a value that forms a value string assignment. In this example, one could assign the variable the value:

```
"My value"
```

After compilation, the procedure would look like:

```
procedure test()
{
  String a = "My value";
}
```

Macro variables can be defined as part of the compilation process using the C++ API routine **uxp::sys::compileUXL.**

1.2.6 Built-In Variables

The following variables are automatically defined by the system:

Table 7 – Built-In Variables

Variable Name	Data Type	Description
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.
false	Integer	Contains the value 0.
locale	String	Contains the current locale for the current access.
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.
p1	String	Contains a parameter that was passed into the engine from an application or the command line.

Variable Name	Data Type	Description
p2	String	Contains a parameter that was passed into the engine from an application or the command line.
p3	String	Contains a parameter that was passed into the engine from an application or the command line.
p4	String	Contains a parameter that was passed into the engine from an application or the command line.
p5	String	Contains a parameter that was passed into the engine from an application or the command line.
p6	String	Contains a parameter that was passed into the engine from an application or the command line.
p7	String	Contains a parameter that was passed into the engine from an application or the command line.
p8	String	Contains a parameter that was passed into the engine from an application or the command line.
procedure_name	String	Contains the name of the currently executing procedure.
success	Integer	Contains the value indicating success.
true	Integer	Contains the value 1.

1.2.7 Intrinsic Functions

The following procedures are native to the **UXL** language and are available for general scripting.

1.2.7.1 Intrinsic Function Summary

Table 8 – Function Summary

Function	Description
abort	Logs a fatal error and then exits the process.
abs	Determines the absolute value of an expression.
addMinutes	Adds the specified number of minutes to the date.
appendList	Appends the evaluated expression to the list.
bitTest	Determines if the bit at the specified offset is set in the number.
chr	Converts the specified expression to a string.
clear	Clears a target variable.
clearList	Removes all elements from the list.
concat	Concatenates all arguments into a single string.
copyright	Returns the product copyright declaration.
countList	Gets the length of the specified list.
execute	Dynamically executes a **UXL** expression.
exp	Determines the exponential value of an expression.

Function	Description
formatDate	Formats a date and time value using formatting rules.
getenv	Gets an environment variable.
getList	Gets an item from the specified list.
hash	Computes a hash of the specified expression.
isByteArray	Tests the data type of the expression as a bytearray type.
isDate	Tests the data type of the expression as a date type.
isFloat	Tests the data type of the expression as a float type.
isInt	Tests the data type of the expression as an integer type.
isList	Tests the data type of the expression as a list type.
isString	Tests the data type of the expression as a string type.
isStruct	Tests the data type of the expression as a structure type.
ln	Determines the natural log value of an expression.
locate	Searches for the first occurrence of string in a source string.
log	Determines the $\log^{10}$ value of an expression.
max	Extracts the maximum value from a list of values.
memcpy	Copies source data to a target variable.
memset	Fills the specified variable with an expression.
Min	Extracts the minimum value from a list of values.
osPlatform	Returns the current operating system name.
pow	Calculates a number raised to the specified power.
print	Evaluates and prints the specified expression to standard output. This is more of a utility function to support **UXL** development.
printf	Prints a formatted string to the current standard output.
printLog	Prints a formatted string to the current application log.
procedureExists	Determines if the procedure exists.
productName	Returns the current product name.
removeList	Removes the specified element number from the list.
rnd	Generates a random number.
showStats	Prints the results to the current output stream.
sizeOf	Gets the number array elements for the specified expression.
sleep	Pauses execution for the specified number of seconds.
split	Parses a source string into a list of string tokens using the specified separator.
sprintf	Writes a formatted string to the specified output variable.
sqrt	Determines the square root value of a numeric expression.
startStats	Starts a counter that keeps track of elapsed and cpu time.

Function	Description
stopStats	Stops the counters from a previous call to startStats ().
strcat	Appends the input value to an output variable.
strcpy	Copies the input value to an output variable.
strlen	Gets the length of the source string.
substr	Extracts a substring from a source string.
toUpper	Converts the specified string to uppercase.
timeDiff	Gets the number of seconds between the two date / time expressions.
timeOffset	Gets the UTC zone offset in minutes.
timeZone	Gets the current time zone.
toDate	Converts the expression to a valid date and time.
today	Returns the current date and time in UTC.
toLower	Converts the specified string to lowercase.
toString	Converts the specified expression to a string.
Trim	Trims all leading and trailing whitespace from the string.
value	Clones the evaluated expression.
variableExists	Determines if the named variable exists.

1.2.7.2 Intrinsic Function Descriptions

abort (expr)

Logs a fatal error and then exits the process.

Parameters:

expr	The message to display upon exit.

Returns:

None

varying abs (arg)

Determines the absolute value of an expression.

Parameters:

arg	Argument to process.

Returns:

Absolute value. The data type will be identical to the original argument.

date addMinutes (date , minutes)

Adds the specified number of minutes to the date.

Parameters:

date	Date / time to amend.
minutes	Number of minutes to add.

Returns:

Modified date.

appendList (list , expr)

Appends the evaluated expression to the list.

Parameters:

list	List to receive a copy of the data.
expr	Expression that produces a value to append to the list. If the expression is a variable name, the variable is cloned to produce an independent copy of the data.

Returns:

None

string bitTest (number , offset)

Determines if the bit at the specified offset is set in the number.

Parameters:

number	The number to test.
offset	The relative bit number to test in the number. Offsets begin at zero.

Returns:

Zero if the bit is not set; otherwise, the value represented by the bit offset. For example, testing bit 2 in the number 6 will yield a return value of 2.

string chr (expr)

Converts the specified expression to a string.

Parameters:

expr	ASCII value expression to evaluate and convert.

Returns:

Converted string

clear (target-name [, count-expr])

Clears a target variable.

Parameters:

target-name	The variable to clear.
count-expr	An optional expression that specifies the number of array elements to clear.

Returns:

None

clearList (list)

Removes all elements from the list.

Parameters:

list	List to clear.

Returns:

None

string concat (arg1 , … , argn)

Concatenates all arguments into a single string.

Parameters:

argn	String to append to output string

Returns:

Concatenated string

string copyright ( )

Returns the product copyright declaration.

Returns:

Copyright as a string.

int countList (list)

Gets the length of the specified list.

Parameters:

list	List to count.

Returns:

Number of elements in the list.

execute (expr)

Dynamically executes a UXL expression.

Parameters:

expr	A string containing a valid **UXL** statement.

Returns:

None

float exp (arg)

Determines the exponential value of an expression.

Parameters:

arg	Argument to process.

Returns:

Exponential value.

string formatDate (fmt , date)

Formats a date and time value using formatting rules.

Parameters:

fmt	Date format mask. The following formatting options are supported: d The day as number without a leading zero (1 to 31) dd The day as number with a leading zero (01 to 31) M The month as number without a leading zero (1-12) MM The month as number with a leading zero (01-12) yy The year as two digit number (00-99) yyyy The year as four digit number These expressions may be used for the time: h The hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display) hh The hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display) m The minute without a leading zero (0 to 59) mm The minute with a leading zero (00 to 59) s The second without a leading zero (0 to 59) ss The second with a leading zero (00 to 59) z The milliseconds without leading zeroes (0 to 999) zzz The milliseconds with leading zeroes (000 to 999) AP Used for AM/PM display. *AP* will be replaced by either “AM” or “PM”. Ap Used for am/pm display. *Ap* will be replaced by either “am” or “pm”.
Date	Date to format.

Returns:

The formatted date as a string.

string getenv (expr)

Gets an environment variable.

Parameters:

expr	External environment variable to fetch.

Returns:

Variable contents.

varying getList (list , element)

Gets an item from the specified list.

Parameters:

list	List containing data.
element	The zero-based element number to retrieve. The maximum element number is the list size minus one.

Returns:

The retrieved element. The data type is based on the data type of the list element.

string hash (expr)

Computes a hash of the specified expression.

Parameters:

expr	Expression to evaluate.

Returns:

Hash value as a string.

int isByteArray (expr)

Tests the data type of the expression as a bytearray type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a bytearray. 0 if it is not a bytearray.

int isDate (expr)

Tests the data type of the expression as a date type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a date. 0 if it is not a date.

int isFloat (expr)

Tests the data type of the expression as a float type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a float. 0 if it is not a float.

int isInt (expr)

Tests the data type of the expression as an integer type.

Parameters:

expr	Expression to test

Returns:

1 if data type is an integer. 0 if it is not an integer.

int isList (expr)

Tests the data type of the expression as a list type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a list. 0 if it is not a list.

int isString (expr)

Tests the data type of the expression as a string type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a string. 0 if it is not a string.

int isStruct (expr)

Tests the data type of the expression as a structure type.

Parameters:

expr	Expression to test

Returns:

1 if data type is a structure. 0 if it is not a structure.

float ln (arg)

Determines the natural log value of an expression.

Parameters:

arg	Argument to process.

Returns:

Log value.

int locate (source , search)

Searches for the first occurrence of string in a source string.

Parameters:

source	Source string.
search	String to locate.

Returns:

The zero-based offset of the location. A -1 indicates not found.

float log (arg)

Determines the $\log^{10}$ value of an expression.

Parameters:

arg	Argument to process.

Returns:

Log value.

varying max (arg1 , … , argn)

Extracts the maximum value from a list of values.

Parameters:

arg	Argument to test.

Returns:

The largest value of the arguments. The data type will be identical to the original argument.

memcpy (target-name , src-name [, count-expr])

Copies source data to a target variable.

Parameters:

target-name	The variable to receive the data.
src-name	The variable containing the data to be copied.
count-expr	An optional expression that specifies the number of array elements to process.

Returns:

None

memset (target-name , fill-expr [, count-expr])

Fills the specified variable with an expression.

Parameters:

target-name	The variable to receive the fill characters.
fill-expr	An expression that will be copied to the specified variable.
count-expr	An optional expression that specifies the number of array elements in the target variable to receive the data.

Returns:

None

varying min (arg1 , … , argn)

Extracts the minimum value from a list of values.

Parameters:

arg	Argument to test.

Returns:

The smallest value of the arguments. The data type will be identical to the original argument.

string osPlatform ( )

Returns the current operating system name.

Returns:

Operating system name.

varying pow (num, exponent)

Calculates a number raised to the specified power.

Parameters:

num	Number to multiply.
exponent	Number used to raise.

Returns:

Result value.

print expr

Evaluates and prints the specified expression to standard output. This is more of a utility function to support **UXL** development.

Parameters:

expr	The command to execute.

Returns:

None

printf (format [, arg1 … , argn])

Prints a formatted string to the current standard output.

Parameters:

format	The format string containing argument placeholders. Unlike standard printf in the C-language, place holders take the form of %1, %2, etc. A maximum of five arguments is supported.
argn	An optional expression that specifies a substitution argument. Arg1 will be substituted for the placeholder %1. Arg2 will substitute into %2, etc.

Returns:

None

printLog(format [, arg1 … , argn])

Prints a formatted string to the current application log.

Parameters:

format	The format string containing argument placeholders. Unlike standard printf in the C-language, place holders take the form of %1, %2, etc. A maximum of five arguments is supported.
argn	An optional expression that specifies a substitution argument. Arg1 will be substituted for the placeholder %1. Arg2 will substitute into %2, etc.

Returns:

None

int procedureExists (name)

Determines if the procedure exists.

Parameters:

name	Procedure to find.

Returns:

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

string productName ( )

Returns the current product name.

Returns:

Product name.

removeList (list , element)

Removes the specified element number from the list.

Parameters:

list	List containing elements.
expr	The element number to remove. Lists are zero-based arrays, so the first element will be zero and the last element will be the list size minus 1.

Returns:

None

int rnd (low , high)

Generates a random number. The current time is the random number generator seed.

Parameters:

low	Low value for random number.
high	High value for random number.

Returns:

Random number

showStats ()

Prints the results to the current output stream.

Returns:

None

int sizeOf (source)

Gets the number array elements for the specified expression.

Parameters:

source	Variable to read.

Returns:

Number of elements.

sleep (expr)

Pauses execution for the specified number of seconds.

Parameters:

expr	The number of seconds to pause.

Returns:

None

list split (src , separator)

Parses a source string into a list of string tokens using the specified separator.

Parameters:

src	The source string to be parsed.
separator	The separator string use to break the source string into tokens.

Returns:

List of strings.

sprintf (outbuf , format [, arg1 … , argn])

Writes a formatted string to the specified output variable.

Parameters:

outbuf	The variable to which the formatted string will be written.
format	The format string containing argument placeholders. Unlike standard printf in the C-language, place holders take the form of %1, %2, etc. A maximum of five arguments is supported.
argn	An optional expression that specifies a substitution argument. Arg1 will be substituted for the placeholder %1. Arg2 will substitute into %2, etc.

Returns:

None

float sqrt (num)

Determines the square root value of a numeric expression.

Parameters:

num	Argument to process.

Returns:

Result value.

startStats ( )

Starts a counter that keeps track of elapsed and cpu time.

Returns:

None

stopStats ()

Stops the counters from a previous call to startStats ().

Returns:

None

strcat (output , input)

Appends the input value to an output variable.

Parameters:

output	Variable to receive data.
input	Data to append to the output variable data.

Returns:

None

strcpy (output , string)

Copies the input value to an output variable.

Parameters:

output	Variable to receive data.
input	Data to copy to the output variable data.

Returns:

None

int strlen (source)

Gets the length of the source string.

Parameters:

source	Source string.

Returns:

Length of string

string substr (source , start , length)

Extracts a substring from a source string.

Parameters:

source	Source string.
start	Offset into source string from which substring starts. Zero is the first character.
length	Number of characters to extract.

Returns:

Extracted substring.

int timeDiff (date1 , date2)

Gets the number of seconds between the two date / time expressions.

Parameters:

date1	First date / time.
date2	Second date / time.

Returns:

Number of seconds.

int timeOffset( )

Gets the UTC zone offset in minutes.

Parameters:

None

Returns:

Number of minutes.

string timeZone ( )

Gets the current time zone.

Parameters:

None

Returns:

Time zone as a string.

date toDate (expr)

Converts the expression to a valid date and time.

Parameters:

expr	Argument to process. If the expression is numeric, it attempts to convert the value from seconds to the date/time entity. If the expression is a string, then it attempts to create a date/time entity by parsing the value.

Returns:

Date value.

date today ( )

Returns the current date and time in UTC.

Returns:

The current date and time.

toLower (string)

Converts the specified string to lowercase.

Parameters:

string	String to convert.

Returns:

None

string toString (expr)

Converts the specified expression to a string.

Parameters:

expr	Expression to evaluate and convert.

Returns:

string	Converted string

toUpper (string)

Converts the specified string to uppercase.

Parameters:

string	String to convert.

Returns:

None

trim (string)

Trims all leading and trailing whitespace from the string.

Parameters:

string	String to trim.

Returns:

None

varying value (expr)

Clones the evaluated expression.

Parameters:

expr	Expression to clone.

Returns:

Temporary variable containing a copy of the data.

int variableExists (name)

Determines if the named variable exists.

Parameters:

name	Variable to find. The variable name must not be quoted.

Returns:

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

1.2.8 Scripting Functions

The **UXL Script Engine** is a command line interface to the **UXP Technology**. It supports the same syntax as the internal **P-code execution engine (KCL Code executor, UXP VM engine)**, but without support for any **UXP Object** authentication and access procedures and functions.

The following coding conventions are used:

- String literal values must be enclosed in double-quotes.
- One or more function arguments that are surrounded by square brackets are considered optional.

The following built-in scripting commands are supported:

@script-file

Executes the specified file as a list of one or more **UXL** script commands.

Note: Nested script files are supported.

The following functions provide a rich set of capabilities when working with **UXP Technology**. Many of the functions require package loading prior to execution.

The following packages can be loaded via the **x::loadPackage** function:

- agent
- file
- id
- sf

- sql
- str
- util

The package name corresponds the prefix of the function names.

1.2.8.1 Scripting Function Summary

Table 9 – Function Summary

Procedure	Description
agent::getProperty	Gets an agent property.
agent::isRunning	Determines if the **Agent** is running.
agent::setProperty	Sets an agent property.
agent::start	Starts the **Agent** as a background process.
agent::stop	Stops the **Agent**.
file::cd	Executes an external change current working directory command.
file::closes	Closes an open file.
file::dir	Gets a directory list for a specified path.
file::exists	Determines if the specified file exists.
file::isDir	Determines if the specified path is a directory.
file::join	Constructs a file from a set of split files.
file::mkdir	Creates the specified relative directory.
file::mkpath	Creates an entire directory path.
file::open	Opens the specified file for access.
file::parse	Parses a file specification into basic components.
file::pos	Gets the current position within an open file.
file::pwd	Executes an external pwd command.
file::read	Reads bytes from an open file.
file::readLine	Reads a line of data from an open file.
file::readAll	Reads an entire file into binary buffer.
file::remove	Removes the specified file.
file::rename	Renames the specified file.
file::seek	Sets the current position within an open file.
file::setPermissions	Sets the access permissions for the specified file.
file::size	Gets the size of the specified file.
file::split	Splits a specified file into sub-files.
file::splitSize	Gets the total size of the file split.

Procedure	Description
file::tempfile	Creates a unique temporary file.
file::touch	Changes the last modification date and time for a file.
file::validateSplit	Validates a current set of file splits.
file::write	Writes bytes to an open file.
file::writeAll	Writes a buffer to a file.
id::addConfigurations	Adds configurations to the **UXP ID** definition.
id::addUsers	Adds users to the **UXP ID** definition.
id::applyRules	Applies rules to the **UXP ID** definition.
id::applyRulesByName	Applies rules to the **UXP ID** definition, a preset.
id::closeExchangeSession	Closes a UXP **ID** exchange session.
id::closeSession	Closes a UXP **ID** session.
id::getConfiguration	Creates an XML document (ID definition) containing the current configuration.
id::getMessageProperty	Gets the specified **UXP Object**-protected data property.
id::getProperties	Creates an XML document (ID definition) containing the **UXP ID** properties.
id::getProperty	Gets the value of the specified **UXP ID** property.
id::getRuleParameter	Gets a rule parameter value from a UXP ID document.
id::getUsers	Creates an XML document (ID definition) containing the users from an existing **UXP ID**.
id::newAutoUnprotect	Creates a compiled **UXL script** that automatically unprotects data.
id::newDocument	Creates a new **UXP ID** definition XML document for designing a new **UXP ID**.
id::newDocumentFromMachine	Creates a UXP **ID** definition XML document that can be locked to a host system.
id::newIdFromDocument	Creates a new **UXP ID** from an XML document containing **UXP ID** metadata.
id::newIdFromKcl	Creates a UXP **ID** from an existing **KCL** file.
id::openExchangeSession	Opens a UXP **ID** exchange session.
id::openSession	Opens a UXP **ID** session.
id::readMessage	Extracts the contents of a UXP **ID**-protected file.
id::setRuleParameter	Sets a rule parameter within a UXP ID document.
id::showMessage	Shows an overview of the **UXP ID**-protected file.
id::transformID	Transforms a **UXP ID**.
id::transformIDbyName	Transforms a **UXP ID**.
id::validate	Validates and reformats **UXP ID** document.

Procedure	Description
id::writeMessage	Protects a file using the specified **UXP ID** and an optional exchange **UXP ID**.
sf::addFile	Adds one or more files into a protected **UXP Object**.
sf::alterFile	Alters the specified virtual file.
sf::alterIdentity	Updates a UXP with a new identity and governance.
sf::checkCompliance	Checks for a compliance violation within the **UXP Object**.
sf::closeDrive	Closes a **UXP** drive.
sf::closeIndex	Closes an open **UXP Object** index.
sf::closeUxp	Closes an open **UXP Object**.
sf::compareExternalFile	Compares a virtual file to an external file.
sf::compareFiles	Compares a virtual file to a virtual file within a second **UXP Object**.
sf::copyFile	Copies a virtual file to a second **UXP Object**.
sf::deleteFile	Deletes the specified virtual file from the **UXP Object**.
sf::directory	Prints directory of virtual files to the current output stream.
sf::exportFile	Exports one or more virtual files from the **UXP Object** to external files.
sf::extractFile	Extracts the contents of a virtual file for display.
sf::fetchIndex	Fetches index data from an open index.
sf::fileExists	Determines if a virtual file exists in the specified **UXP Object**.
sf::getAttribute	Gets the specified **UXP Object** attribute.
sf::getChallenges	Gets the current set of challenges during a **UXP** authentication session.
sf::getCompliance	Gets compliance properties.
sf::getDriveList	Gets a list of the open **UXP** drives.
sf::getEventList	Gets a list of recorded events from the UXP.
sf::getFileInfo	Gets information for a virtual file within a UXP.
sf::getIdentity	Gets the unique UXP identity.
sf::getPublicInfo	Gets public information for the specified **UXP**.
sf::join	Joins a group of **UXP Objects** into original clear file.
sf::newDirectory	Creates a new virtual directory or directory path within the **UXP Object**.
sf::newDrive	Creates a new **UXP Object** and opens it as a drive.
sf::newFile	Creates and opens a new virtual file for writing.
sf::newReality	Creates a new reality within the **UXP Object**.

Procedure	Description
sf::newUxp	Creates and opens a new **UXP Object**.
sf::openDrive	Mounts an existing **UXP** as a drive.
sf::openDriveSSO	Opens an existing **UXP** as a drive using the **UXP ID** single-sign-on protocol.
sf::openFile	Opens a virtual file for reading.
sf::openIndex	Opens an index that can be used to fetch data for an open **UXP Object**.
sf::openUxp	Opens an existing **UXP Object**.
sf::openUxpSSO	Opens an existing **UXP Object** using the **UXP ID** single-sign-on protocol.
sf::posFile	Gets the current read position on an open virtual file.
sf::quickProtect	Protects a file using the specified **UXP ID**.
sf::recordEvent	Record a custom event with the **UXP Object**.
sf::readFile	Reads data from a virtual file.
sf::renameFile	Renames the specified virtual file within the **UXP Object**.
sf::seekFile	Sets the current read position on an open virtual file.
sf::setAttribute	Sets the specified **UXP Object** attribute.
sf::setAutoOpen	Specifies a virtual file to automatically open within the **UXP Object** Assistant.
sf::setCallback	Sets the callback procedure when opening a **UXP Object**.
sf::setCreatorInfo	Sets the specified **UXP Object** creator.
sf::setCompliance	Sets compliance properties.
sf::setReality	Sets the current reality within the **UXP Object**.
sf::setResponses	Sets the challenge responses when opening a **UXP Object**..
sf::showAttributes	Shows the **UXP Object** attributes.
sf::showEvents	Shows the event information for the **UXP Object**.
sf::showSignatures	Shows the signatures for the **UXP Object**.
sf::showUxps	Shows a summary of open **UXP Objects**.
sf::split	Splits a specified file into **UXP Object** sub-files.
sf::touchUxp	Touches a **UXP Object**..
sf::validateUxp	Validates a **UXP Object**..
sf::viewFile	Displays the contents of the specified virtual file.
sf::writeFile	Writes data to a virtual file.
sql::bindCount	Gets the number of bind parameters in a prepared query.
sql::bindValue	Binds a value to a dynamic SQL parameter.

Procedure	Description
sql::columnCount	Gets the number of columns returned by a query.
sql::columnName	Gets the name of a column returned by a query.
sql::columnTable	Gets the parent table name for a selected column.
sql::columnValue	Gets a column value returned by a query.
sql::exec	Executes a query.
sql::execScript	Executes a buffer containing a SQL script.
sql::freeQuery	Deletes a query ID.
sql::newQuery	Creates a new query ID for executing SQL statements.
sql::next	Fetches the next row from an executed query.
sql::prepare	Prepares a SQL query for execution.
str::decode	Decodes a string that was encoded.
str::encode	Encodes a string by applying encryption.
str::replace	Replaces a token in a string with another string.
util::fromBase64	Decodes a Base64 buffer.
util::getEnv	Gets a system environment variable.
util::getHost	Gets the current system host name.
util::getPref	Gets a user preference.
util::getSysPref	Gets a system preference.
util::setPref	Sets a user preference.
util::setSysPref	Sets a system preference.
util::toBase64	Encodes a buffer to Base64.
workflow::absoluteConfigPath	Translates a workspace configuration name into a physical file specification.
workflow::activate	Activates the specified configuration.
workflow::applyPreset	Applies a preset to a **Task** definition.
workflow::configExists	Determines if the specified configuration exists.
workflow::createFolders	Creates underlying folders for a **Task**.
workflow::deleteConfig	Deletes a configuration.
workflow::deleteTask	Deletes an existing **Task** within a configuration setup.
workflow::duplicateConfig	Creates a copy of the specified configuration.
workflow::duplicateTask	Duplicates a **Task**.
workflow::executeTask	Manually execute a **Task**.

Procedure	Description
workflow::findScript	Finds the specified script file within the workflow system.
workflow::getConfigs	Gets a list of configuration names.
workflow::getProperty	Gets the specified workflow system property.
workflow::getPresets	Gets a list of valid **Task** preset names.
workflow::getTaskProperty	Gets a **Task** property.
workflow::getTasks	Gets a list of **Task** names within a configuration setup.
workflow::importConfig	Imports an external configuration XML into a configuration.
workflow::newConfig	Creates a new configuration setup.
workflow::newTask	Creates a new **Task** within a configuration setup.
workflow::renameConfig	Renames an existing configuration.
workflow::renameTask	Renames a **Task**.
workflow::setProperty	Sets the specified **Workflow** system property.
workflow::setTaskProperty	Sets a **task** property.
workflow::taskExists	Determines if the specified **Task** exists.
x::batch	Runs a script command in a separate process.
x::cancelOutput	Cancels the recording of operation output.
x::compile	Compiles the specified **UXL** source file.
x::executeFile	Executes a **UXL script** file.
x::exportConfiguration	Exports the current address configuration.
x::loadPackage	Loads optional functions.
x::printToLog	Prints a information to the current application log.
x::prompt	Prompts the user for input.
x::setOutput	Records any output to the specified file.
x::setProgress	Sets the level of progress when performing long operations.
x::shell	Executes an external shell command.
x::showConfiguration	Shows the current device and location configuration.
x::showLicense	Shows the attributes of the specified license.
x::showOutput	Shows the current file for all display output.
x::writeMachineCache	Writes a snapshot of current device in a private cache.

1.2.8.2 Scripting Function Descriptions

agent::getProperty(name, value)

Gets an **Agent** property value.

Parameters:

name	Specifies the property to set. Possible values: • Logging • Startup

Returns:

Property value.

agent::isRunning ()

Determines if the **Agent** is running.

Parameters:

None

Returns:

True if running.

agent::setProperty(name, value)

Sets an **Agent** property value.

Parameters:

name	Specifies the property to set. Possible values:
	• **Logging**
	  ○ Can be any value from 0 to 5, where 0 indicates no logging.
	• **Startup**
	  ○ Automatic
	  ○ Manual

value	Specifies the property value.

Returns:

True if successful.

agent::start ([force])

Starts the **Agent** as a back-group process. The **Agent** optimizes message delivery, license evaluation and performs workflow actions.

Parameters:

force	When true, the **Agent** will attempt a hard restart regardless of the current agent status. If false, the operation will not attempt to start the **Agent** if it is already running. **Default**: false

Returns:

True if successful.

agent::stop ()

Stops the **Agent**.

Parameters:

None

Returns:

True if successful.

file::cd (dir [, outbuf])

Executes an external change current working directory command.

Parameters:

dir	External directory to set as the current working directory.
outbuf	Buffer to receive full directory specification. If omitted, then the current working directory is printed to standard output.

Returns:

Current working directory

file::close (fp)

Closes an open file.

Parameters:

fp	File handle as returned by the **file::open function**.

Returns:

True if file successfully closed.

file::dir (path, &outlist)

Gets a directory list of the specified path.

Parameters:

path	Specifies the parent path to list. If empty, the current working directory will be used.

outlist	A list datatype to receive the directory entries. Each entry will be a fully qualified file specification.
filter	Optional wildcard-style filter to process files. Example: file::dir(“”, mylist, “*.h”);

Returns:

True if file successfully accessed.

file::exists (filename)

Determines if the specified file exists.

Parameters:

filename	Name of the file to test.

Returns:

True if file exists.

file::isDir (filespec)

Determines if the specified file is a directory.

Parameters:

filespec	File specification to test.

Returns:

True if file is a directory.

file::join (metafile, outfile[, options])

Joins a set of sub-files that were created by a **file::split** operation.

Parameters:

metafile	Specifies the metadata file necessary to reconstruct the data file.
outfile	Specifies the reconstructed output file.
options	Join options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Delete** Delete the sources files after a successful join operation. • **Replace** Replace the file if it already exists.

	Optional

Returns:

True if successful.

file::mkdir (name)

Creates the specified relative directory.

Parameters:

name	Name of the directory to create.

Returns:

True if successful.

file::mkpath (path)

Creates the entire directory path.

Parameters:

path	Directory path to create.

Returns:

True if successful

file::open (filename, mode)

Opens a file for access.

Parameters:

filename	File specification to open.
mode	Specifies the open mode. This is the same as the standard library fopen mode parameter.

Returns:

File handle.

file::parse (filespec, dir, basename, suffix)

Parses a file specification into its basic components.

Parameters:

filespec	File specification to parse.
dir	Receives the directory portion of the file specification.

basename	Receives the base name portion of the file specification.
suffix	Receives the suffix or file type of the file specification.

Returns:

True if successful.

file::pos (fp)

Gets the current position within an open file.

Parameters:

fp	Handle of the open file.

Returns:

Current file position of the file. -1 indicates an error.

file::pwd ([outbuf])

Executes an external pwd command. This will print out the current external working directory.

Parameters:

outbuf	Optional buffer to receive current working directory specification. If omitted, the directory specification is printed to standard output.

Returns:

True if successful.

file::read (buf, len, cnt, fp)

Reads bytes from an open file.

Parameters:

buf	Buffer to receive bytes.
len	Size of each element to read.
cnt	Number of elements to read.
fp	Handle of open file.

Returns:

Number of elements read.

file::readLine (buf, fp)

Reads a line of data from an open file. A line is determined to be a string of characters that is terminated with a newline character.

Parameters:

buf	Buffer to receive bytes.
fp	Handle of open file.

Returns:

True indicates a line was read. False indicates and end-of-file was reached.

file::readAll (filename)

Reads the entire file into a buffer.

Parameters:

filename	Name of the file to read.

Returns:

Buffer containing file contents.

file::remove (path)

Removes the specified file or directory.

Parameters:

path	Specification to remove.

Returns:

Number of files deleted.

file::rename (old, new)

Renames the specified file or directory.

Parameters:

old	Specification to rename.
new	New name

Returns:

True if successful

file::seek (fp, offset, origin)

Sets the current position within an open file.

Parameters:

fp	Handle of the open file.

offset	Number of bytes to shift relative to the origin.
origin	Specifies the starting point from which the shift will occur. Possible values are: • **SEEK_SET** • **SEEK_CUR** • **SEEK_END**

Example:

file::seek(fp, 10, "SEEK_CUR");

Returns:

Current file position of the file. -1 indicates an error.

file::setPermissions (fname, permissions)

Sets the access permissions for the specified file.

Parameters:

fname	Specification to modify.
permissions	Sets the access permissions for the specified file.

Returns:

True if successful

file::size (filespec)

Gets the current size of the specified file.

Parameters:

filespec	File specification to test.

Returns:

Current file size. -1 indiates an error.

file::split (infile, outfile, size-option, size[, options])

Splits a file into sub-files for optimized transport or possible obfuscation. Upon completion, the sub-files plus a metadata XML file can be used to reconstruct the original data file.

Parameters:

infile	Specifies the source file to split.
outfile	Specifies the output location, base file name and file type for the file splits. Each split will contain the base file name and file type, plus a unique identifier.

size-option	Specifies the sizing approach when splitting the file. Possible values: 0. The **size** parameter represents the maximum size of a file split. 1. The **size** parameter represents the number of desired file splits.
size	Specifies either the maximum size of a file split or the maximum number of splits.
options	Join options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Delete** Delete the sources file after a successful split operation. • **Replace** Replace the file if it already exists. Optional

Returns:

File specification of meta file.

file::splitSize (metafile)

Retrieves the total size of a file split that was created by a **file::split** operation.

Parameters:

metafile	Specifies the metadata file necessary to reconstruct the data file.

Returns:

Total size of files within the split.

file::tempfile (template)

Creates a unique temporary file.

Parameters:

template	File template. Include the characters XXXXXX to produce a unique file name.

Returns:

File name.

file::touch (filename)

Changes the last modification date and time for the specified file.

Parameters:

filename	File specification to update.

Returns:

True if file successfully updated.

file::validateSplit (metafile)

Validate a file split that was created by a **file::split** operation. Presently, the validate only determines if the necessary files are present. Actual validation of content can only be done by reconstruction.

Parameters:

metafile	Specifies the metadata file necessary to reconstruct the data file.

Returns:

Name of original file that was split. If empty, an error has occurred.

file::write (buf, len, cnt, fp)

Writes bytes to an open file.

Parameters:

buf	Buffer to write.
len	Size of each element to write.
cnt	Number of elements to write.
fp	Handle of open file.

Returns:

Number of elements written.

file::writeAll (filename, buffer [, options])

Creates a new file and writes buffer.

Parameters:

filename	Name of the file to create and write.
buffer	Buffer to write
options	Create options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Replace** Replace the file if it already exists. Optional

id::addConfigurations (doc, configs [, user])

Adds configurations to the specified **UXP ID** definition.

Parameters:

doc	**UXP ID** definition to update.
configs	XML document containing configurations.
user	Optional username under which the configurations will be added. If empty, the configurations will be added at the **UXP ID** definition level.

Returns:

Buffer containing updated **UXP ID** definition.

id::addUsers (doc, users, filter)

Adds users to the specified **UXP ID** definition.

Parameters:

doc	**UXP ID** definition to update.
users	XML document containing users.
filter	Optional comma-separated list of user names to include in the **UXP ID** definition. If empty, all users will be added.

Returns:

Buffer containing updated **UXP ID** definition.

id::applyRules (doc, policyXML [, users. policies])

Applies policies to the **UXP ID** definition

Parameters:

doc	**UXP ID** definition to update.
policyXML	XML document containing policy settings.
users	Optional list of users to which the settings will be applied. Multiple users must be separated by the '|' character.
policies	List of policies to apply. Multiple policies must be separated by the '| character. If empty or contains an asterisk, all relevant policies will be applied. Supported policy names: Access Alerts Approvals

	Configurations Events Privileges Restrictions Schedule UserApprovals UserConfigurations UserPrivileges UserSchedule

Returns:

Buffer containing updated **UXP ID** definition.

id::applyRulesByName (doc, preset [, users, policies])

Applies policies to the **UXP ID** definition using a defined policy preset.

Parameters:

doc	**UXP ID** definition to update.
preset	Policy preset name.
users	Optional list of users to which the settings will be applied. Multiple users must be separated by the '|' character.
policies	List of policies to apply. Multiple policies must be separated by the '| character. If empty or contains an asterisk, all relevant policies will be applied. Supported policy names: Access Alerts Approvals Configurations Events Privileges Restrictions Schedule UserApprovals UserConfigurations UserPrivileges UserSchedule

Returns:

Buffer containing updated **UXP ID** definition.

id::closeSession (session)

Closes a UXP **ID** session.

Parameters:

session	**UXP ID** session identifier.

id::closeExchangeSession (session, exchange)

Closes an **ID** exchange session.

Parameters:

session	**UXP ID** session identifier.
exchange	**UXP ID** exchange session identifier.

Returns:

True if successful.

id::closeSession (session)

Closes a UXP **ID** session.

Parameters:

session	**UXP ID** session identifier.

Returns:

True if successful.

id::getConfiguration ( )

Creates an XML document containing the current configuration.

Returns:

Buffer containing configuration.

id::getMessageProperty (id, filename, property)

Gets the specified **UXP Object-protected data** property.

Protection utilizes the **SmartMessage** protocol.

Parameters:

id	Exchange session identifier.
filename	Protected data file name.
property	Name of the data property. Possible values are: **CacheSize** **ExchangeIdentity** **Flags**

	Identity **IP** **Location** **Name** **PageSize** **PersonalName1** **Size** **Trusted** **User**

Returns:

Value of the property as a string.

id::getProperties (id, &outbuf)

Creates an XML document containing the **UXP ID** definition properties.

Parameters:

id	Buffer containing a valid binary **UXP ID** definition.
outbuf	Buffer or **UXL** list to receive data. If the output variable is a list, then the call returns the properties as a value/pair list. Otherwise, the buffer will be returned as an XML document.

Returns:

True if successful.

id::getRuleParameter (id, user, rule, parameter)

Gets the specified **UXP** rule parameter.

Parameters:

id	Definition document containing **UXP ID** metadata XML.
user	Optional username from which to fetch the parameter value.
rule	Rule name containing the desired parameter.
parameter	Name of the rule parameter name.

Returns:

Value of the parameter as a string.

id::getUsers (id [, usernames])

Creates an XML document containing the users from an existing **UXP ID** definition.

Parameters:

id	Buffer containing a valid binary **UXP ID** definition.

usernames	Optional list to receive usernames from the ID.

Returns:

Buffer containing users.

id::newAutoUnprotect (id, script)

Creates a compiled **UXL script** that automatically unprotects data. The **auto-unprotect** can only be created using a user specification that is local. By local, the user specification is in source form, complete with credential information. If a user specification is in protected form, the script creation will fail.

Parameters:

id	Definition document containing **UXP ID** metadata XML.
script	Name of the script to be used as a template for the compiled script. The script must be located in the scripts folder within the **Workflow** home.

Returns:

Buffer containing compiled script.

id::newDocument ( )

Creates a UXP **ID** definition XML document template for designing a new **UXP ID**.

Parameters:

None

Returns:

Buffer containing document template.

id::newDocumentFromMachine ( )

Creates a **UXP ID** definition XML document definition that can be locked to a host system. The document will contain an auto-generated user, the current configuration and the policies set to the policy preset **WorkflowMachine**.

Parameters:

None

Returns:

Buffer containing new **UXP ID** in XML format.

id::newIdFromDocument (id [, options])

Creates a new binary **UXP ID** definition from an XML document containing **UXP ID** metadata.

Parameters:

id	Document containing **UXP ID** metadata XML.
options	Create options Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **v2id** Creates a version 2 architecture ID. • **Compress** Reduces the size of a version 2 ID, but also slightly reduces the performance when opening a session or creating a UXP. Optional

Returns:

Buffer containing new **UXP ID**.

id::newIdFromKcl (code, name, expires [, options])

Creates a binary **UXP ID** definition from an existing **KCL** buffer. The **KCL** buffer must be valid and capable of being used to create a **UXP Object**. All users defined within the **KCL** module will be visible within the **UXP ID**.

*See **Sertainty Developer Guide** for a full description of **KCL**.*

Parameters:

code	Buffer containing **KCL** code.
name	Internal name to be given to the new **UXP ID**.
expires	Specifies a date upon which the new **UXP ID** will expire.
options	Create options Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Nooptimize** Disables **KCL code** optimization. Optional

Note: The new **UXP ID** cannot be imported into a **Workflow Assistant** nor any other **UXP ID**.

Returns:

Buffer containing new **UXP ID**.

id::openExchangeSession (session, filename)

Opens a UXP **ID** exchange session. An exchange session is used to protect data on behalf of another **UXP ID**.

Example: if you wish to create a UXP **ID-protected message** to be sent to a friend, one would acquire their **UXP ID** and use it to create the message. Then, when the friend receives the message, they would be able to open and decode it using their **UXP ID**.

Protection utilizes the **SmartMessage** protocol.

Parameters:

session	Current **ID** session identifier.
filename	**ID** file name.

Returns:

Exchange session identifier

id::openSession (filename, responses, share, stop_time)

Opens a UXP **ID** session. A session can be used to protect and access data external to a **UXP Object**.

Parameters:

filename	**ID** file name.
responses	Provides initial challenge-response pairs. If all required responses are found in the list, then the **UXP Object** will not prompt the user for additional challenge responses. The format of the list is: **name1=value1|name2=value2 ...**
share	True if the session is a shared session that permits other processes to see your **single-sign-on** session.
stop_time	Specifies the time the session will end. Possible values: • 0 indicates the session will end based on the default timeout for the **UXP ID**. • 1 disables timeouts The session must be ended by the caller. • Number of milliseconds into the future when the session will end.

Returns:

Session identifier

id::readMessage (session, infile, outfile [, options])

Extracts the contents of a UXP **ID** protected file.

Protection utilizes the **SmartMessage** protocol.

Parameters:

session	**UXP ID** session identifier as returned by the **sf::openExchangeSession** function.
infile	Protected file name.
outfile	Output file name to receive data.

options	Operation options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Replace** Replace any existing entries with the same name as the exported virtual files. Optional.

Returns:

True if successful.

id::setRuleParameter (id, user, rule, parameter)

Sets the specified **UXP** rule parameter.

Parameters:

id	Definition document containing **UXP ID** metadata XML.
user	Optional username from which to fetch the parameter value.
rule	Rule name containing the desired parameter.
parameter	Name of the rule parameter name.
value	Value to assigned to the rule parameter.

Returns:

Buffer containing updated **UXP ID** document.

id::showMessage (session, infile)

Shows an overview of the **UXP ID** protected message.

Parameters:

session	**UXP ID** session identifier as returned by the **sf::openExchangeSession** function.
infile	Protected file name.

Returns:

True if successful.

id::transformID (doc, policyXML [, users. policies])

Converts an IIC using the specified policies. The IIC must be a version 2 ID to convert. If no policies are specified, then, the operation is a straight copy.

Parameters:

doc	**UXP ID** definition to update.

policyXML	XML document containing policy settings.
users	Optional list of users to which the settings will be applied. Multiple users must be separated by the '|' character.
policies	List of policies to apply. Multiple policies must be separated by the '| character. If empty or contains an asterisk, all relevant policies will be applied. Supported policy names: Access Alerts Approvals Configurations Events Privileges Restrictions Schedule UserApprovals UserConfigurations UserPrivileges UserSchedule

Returns:

Buffer containing a new **binary ID**.

id::transformIDbyName (doc, preset [, users, policies])

Converts an IIC using the specified policies. The IIC must be a version 2 ID to convert. If no policies are specified, then, the operation is a straight copy.

Parameters:

doc	**UXP ID** definition to update.
preset	Policy preset name.
users	Optional list of users to which the settings will be applied. Multiple users must be separated by the '|' character.
policies	List of policies to apply. Multiple policies must be separated by the '| character. If empty or contains an asterisk, all relevant policies will be applied. Supported policy names: Access Alerts Approvals Configurations Events

	Privileges Restrictions Schedule UserApprovals UserConfigurations UserPrivileges UserSchedule

Returns:

Buffer containing a new **binary ID**.

id:validate (document)

Validates and reformats a UXP **ID** definition. Any warnings or errors will be formatted in the errorString variable.

Parameters:

document	**UXP ID** definition to validate and format.

Returns:

Formatted **UXP ID** definition buffer.

id:writeMessage (exchange, infile, outfile [, users, pagesize, cachesize, options])

Protects a file using the specified **UXP ID** definition and an optional exchange **UXP ID**. If an exchange session identifier is not provided, then the protected data is under the control of the current **UXP ID**. If an exchange session is provided, the data is under the control of the **UXP ID** that was opened as an exchange session.

Protection utilizes the **SmartMessage** protocol.

Parameters:

exchange	Exchange session identifier as returned from the **sf::openExchangeSession** function.
infile	File to protect
outfile	File containing protected data.
users	List of users who are permitted access to the protected entity. Multiple users can be separated using a '|' character, e.g. "joe|phil".
pagesize	Specifies a block size for storing protected data. Valid values are: -1 indicates default value A value greater than 512 and less than 8000000. Optional
cachesize	Specifies the number of cache buffers to use when reading a protected file. Value must be between 0 and 20. A zero disables caching.

	Optional
options	Command options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the data while protecting it. • **Minimal** When creating the protected entity, only a minimal amount of header information will be included. This will reduce the size of the entity by several hundred bytes. • **Replace** Replace the existing output file, if necessary. Optional.

Returns:

True if successful.

sf::addFile (id, name, filename [, pagesize, cachesize, options])

Adds one or more files into a protected **UXP Object**.

Parameters:

id	**Identifier representing an open UXP Object.**
name	Name to give the new virtual file. The name only applies to the top level virtual file.
filename	External file or directory name to add.
pagesize	Specifies a block size for storing protected data. Valid values are: -1 indicates default value A value greater than 512 and less than 8000000. Optional
cachesize	Specifies the number of cache buffers to use when reading a protected file. Value must be between 0 and 20. A zero disables caching. Optional
options	File options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the virtual files. The effectiveness of compress depends on the amount of white space and other repeating character sequences.

	• **Delete** Delete the source files after adding the files to the **UXP Object**. • **Recurse** Recursively include files within directories. Optional.

Returns:

True if successful.

sf::alterFile (id, name, item, data [, options])

Alters the specified virtual file. This function can only be utilized by the owner of the **UXP Object**.

Parameters:

id	**Identifier representing an open UXP Object.**
name	Name of the virtual file to alter.
item	Name of the attribute to change. Possible values are: • **CACHESIZE** Specifies a new cache size for virtual file data access. Value must be an integer between 0 and 20. • **FLAGS** Sets the global flags for virtual file or folder. Flags are represented by a single integer composed of bit values. Possible bit values are: o 0x0020 AlertEmail o 0x0040 AlertSMS o 0x0100 NoAccess o 0x2000 NoMove o 0x0002 ReadOnly o 0x0004 ReadWrite o 0x8000 RequireSignature • **ACL** Set the ACL membership as specified in the data parameter. An ACL name must be a user name or the wildcard *. The **UXP Engine** can then be used to filter files based on the ACL privileges. The format is: **name1|flags|starttime|endtime|siguser+** **name2|privs|starttime|endtime|siguser …** **nameN|privs|starttime|endtime|siguser**

	The **flags** value is a bitmask as defined above for the **FLAGS** attribute. The **starttime** and **endtime** values represent an access window. Each value is the number of milliseconds since epoch. The **siguser** is a user that must sign the virtual file in order to be visible by the named user. **Note:** ACL **privs** will supercede FLAGS settings. See **Sertainty Developer Guide** for more information on the **UXP Engine**.
data	Data that may be required by the attribute.
options	Alter options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Recurse** Apply the changes to all child entities. Optional

Returns:

True if successful.

sf::alterIdentity (id, iic)

Updates the **UXP Object** with a new identity and governance module.

Parameters:

id	**Identifier representing an open UXP Object.**
iic	Buffer containing IIC to apply to the open UXP.

Returns:

True if successful.

sf::checkCompliance (filename [, options])

Checks for a compliance violation within the **UXP Object**.

Parameters:

filename	**UXP Object** to check.
options	Command options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Destroy** Destroy the **UXP Object** if there is a violation. Optional.

Returns:

True if successful.

sf::closeIndex (index)

Closes an open **UXP Object** index.

Parameters:

index	Index identifier.

Returns:

True if successful.

sf::closeUxp (id)

Closes an open **UXP**.

Parameters:

id	**Identifier representing an open UXP Object.**

Returns:

True if successful.

sf::compareExternalFile (id1, name,1 name2)

Compares a virtual file to an external file.

Parameters:

id1	Source **Identifier representing an open UXP Object.**
name2	Name of the virtual file to compare.
name2	External file to compare.

Returns:

True if the files are identical

sf::compareFiles (id1, name,1 id2, name2)

Compares a virtual file to a virtual file within a second UXP entity object.

Parameters:

id1	Source **Identifier representing an open UXP Object.**
name2	Name of the virtual file to compare.
id2	Second **Identifier representing an open UXP Object.**
name2	Second file to compare.

Returns:

True if the files are identical

sf::copyFile (id, name, outid, outname [, options])

Copies a virtual file to a second **UXP Object** object.

Parameters:

id	Source **Identifier representing an open UXP Object.**
name	Name of the virtual file to copy.
outid	Destination **Identifier representing an open UXP Object.**
outname	Name of new virtual file copy.
options	Compile options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the new virtual file within the destination **UXP Object**. • **Reclaim** Reclaim the space in the destination **UXP Object** when deleting an existing file. • **Replace** Replace the output virtual file if it already exists. Optional

Returns:

Number of bytes copied

sf::deleteFile (id, filename [, options])

Deletes the specified virtual file from the **UXP Object**.

Parameters:

id	**UXP Object** identifier
name	Name of the virtual file to delete.
options	Delete options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the **UXP Object** after the delete operation to reclaim unused space and reduce the size of the **UXP Object**. For large **UXP Objects**, this may be time consuming.

	• **Shred** Performs a Shred2 operation on the data as part of the delete operation. This may be time consuming. Optional

Returns:

True if successful.

sf::directory (id, [filename, options])

Retrieves a directory of the current **UXP Object** and prints the results to the current output stream.

Parameters:

id	**UXP Object** identifier
filename	Virtual file name or directory to list. Optional.
options	Directory options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Full** Displays virtual file details. • **Hidden** Includes hidden files in the listing. • **Realities** Includes reality entries in the listing. • **Recurse** Recursively include child directories.

Returns:

True if successful.

sf::closeDrive (filename)

Closes an open drive.

Parameters:

filename	**UXP Object** specification or mountpoint. Specified * as the filename to close all open drives.

Returns:

True if successful.

sf::exportFile (id, name, filename [, options])

Exports one or more virtual files from the **UXP Object** to external files.

Parameters:

id	**UXP Object** identifier
name	Virtual file or directory to export.
outspec	File or directory name to receive the data.
options	Export options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Merge** Merge the exported files with existing file names. • **Recurse** Recursively include child directories. • **Replace** Replace any existing entries with the same name as the exported virtual files. Optional

sf::extractFile (id, filename, outbuf, offset, length_or_offset, options)

Extracts the contents of a virtual file into a buffer.

Parameters:

id	**UXP Object** identifier
name	Name of the virtual file to extract.
outbuf	Buffer to receive extracted data.
offset	Starting offset within the virtual file.
length_or_offset	Length or end offset within the virtual file.
options	Extract options. The following options are supported: • **End** Parameter p2 is treated as an end offset. • **Length** Parameter p2 is treated as the length of the data to be fetched.

Returns:

True if successful.

sf::fetchIndex (index, rec, &start, &end)

Fetches index data from an open index. The index must be a CSV file that is open via the **sf::openIndex** function.

Parameters:

index	Index identifier
rec	Record number to fetch.
start	Variable to receive the starting offset within the target virtual file.
end	Variable to receive the end offset within the target virtual file.

Returns:

True if found. False if at the end of the index.

sf::fileExists (id, name)

Determines if a virtual file exists in the specified **UXP Object**.

Parameters:

id	**UXP Object** identifier
name	Virtual file name to locate.

Returns:

True if found.

sf::getAttribute (id, name)

Gets the specified **UXP Object** attribute.

Parameters:

id	**UXP Object** identifier
name	Attribute name. Possible values are: • **COMPANY** • **DESCRIPTION** • **HOSTURL** • **LOCKED** • **NAME** • **OWNER**

Returns:

Value of the attribute as a string.

sf::getChallenges (id)

Retrieves the current set of challenges for a **UXP Object** authentication session.

Parameters:

id	**UXP Object** identifier

Returns:

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

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

sf::getCompliance (id, property)

Gets the specified **UXP Object** compliance property.

Parameters:

id	**UXP Object** identifier
property	Property name. Possible values are: • **Expiration – Sets the expiration date/time when UXP becomes inaccessible.** • **Locked – Permanently locks the compliance properties from further changes.** • **ReadOnly – Permanently sets the UXP to read-only.**

Returns:

Value of the property.

sf::getDriveList (&outlist)

Gets a list of open **UXP** drives.

Parameters:

outlist	A **UXL** list of drive items. Each list item is a value/pair containing the **UXP Object** name and mountpoint.

Returns:

True if successful.

sf::getEventList (id)

Gets the recorded events from the UXP in XML format.

Parameters:

id	**UXP Object** identifier

Returns:

XML document containing event information. If the current user does not have ReadEvent privileges, then an error will occur.

sf::getFileInfo (id, name)

Gets information for virtual file within the specified **UXP Object** in XML format.

Parameters:

id	**UXP Object** identifier
name	Virtual file name to locate.

Returns:

XML document containing file information. If the current user does not have owner privileges, then some items will be empty.

sf::getIdentity (id)

Gets the unique UXP identity.

Parameters:

id	**UXP Object** identifier

Returns:

UXP identity as a printable string.

sf::getPublicInfo (id_or_fname, &outlist)

Gets public information for the specified **UXP Object**.

Parameters:

id_or_fname	**UXP Object** identifier or **UXP Object** file specification. When an identifier is used, the **UXP Object** must be open.
outlist	A **UXL** list of public property items Each list item is a value/pair containing the property name and value.

Returns:

True if successful.

sf::join (metafile, outfile, responses[, options])

Joins a set of **UXP Object** sub-files that were created by a **sf::split** operation.

To open the **UXP Object** files, the caller must either provide all the necessary credentials or use **single-sign-on**.

Parameters:

metafile	Specifies the metadata file necessary to reconstruct the data file.

outfile	Specifies the reconstructed output file. If the output specification contains an **asterisk** as the filename, the original filename will be used for the actual output file.
responses	Authentication responses required to open the **UXP Object** splits. The format is: **name1=value1|name2=value2|...**
options	Join options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Delete** Delete the sources files after a successful join operation. • **Replace** Replace the file if it already exists. Optional

Returns:

File specification for reconstituted data. Empty indicates an error.

sf::newDirectory (id)

Creates a new virtual directory or directory path within the **UXP Object**.

Parameters:

id	**Identifier representing an open UXP Object.**

Returns:

True if successful.

sf::newDrive (filename , id [, mode, mountpoint])

Creates and opens a **UXP Object** as a drive. Once open, the **UXP Object** contents will be accessible by applications that are not aware of the **UXP Technology**.

Parameters:

filename	**UXP Object** file name.
id	Specifies a **UXP ID** to be used for creating the **UXP Object**.
mode	Sets the access mode. Possible values are: • **ShareAll** Opens the **UXP Object** for shared read and writing. This is the default mode. • **ShareReadOnly**

	Mounts the **UXP Object** in read-only mode. Will only be permitted if the **UXP Object** permits read-only access. Optional
mountpoint	Specifies a folder path or drive letter to associate the open **UXP Object**. The path or drive must not exist as the mount operation will create the path. If not specified, the mountpoint will be automatically generated. For non-Windows systems, the mountpoint will default to the folder **Sertainty Drives** under the current user's home folder. Optional

Returns:

The mountpoint of the open drive.

sf::newFile (id, newfile [, pagesize, cachesize, options])

Creates and opens a new virtual file for writing. Currently, virtual files can only be written sequentially.

Parameters:

id	**Identifier representing an open UXP Object.**
newfile	Virtual file name.
pagesize	Specifies a block size for storing protected data. Valid values are: -1 indicates default value A value greater than 512 and less than 8000000. Optional
cachesize	Specifies the number of cache buffers to use when reading a protected file. Value must be between 0 and 20. A zero disables caching. Optional
options	Create options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the data within the virtual file. • **ReadWrite** Creates a read-write architecture **UXP Object**. • **Reclaim** Reclaim space if deleting existing virtual file. • **Replace** Replace an existing file with the same name.

	Optional

Returns:

Virtual file handle

sf::newReality (id, name)

Creates a new reality within the **UXP Object**.

Parameters:

name	Name of the reality to create.
id	**UXP Object** identifier

Note: Only available with AltReality privileges.

Returns:

True if successful.

sf::newUxp (filename, code [, options, key1, key2, taskinfo])

Creates and opens a new **UXP Object**.

Parameters:

filename	**UXP** file name.
code	**UXL** or **ID** file name.
options	Create options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **UXLFile** Indicates the filename specified in the code parameter is a source or compiled **UXL** file. • **IDFile** Indicates the filename specified in the code parameter is an **ID** file. • **Journaling** Indicates the **UXP** will permit adding data without being authenticated. This implements a write-only journal mode. • **Nooptimize** Disables **UXL** code optimization • **ReadWrite**

	When creating the protected entity, the **ReadWrite** architecture will be used. • **Replace** Replace the existing output file, if necessary. • **WriteOnce** When creating the protected entity, the **WriteOnce** architecture will be used.
key1	Specifies an optional domain key or name. If empty, the public domain will be used and **key2** will be ignored. Optional
key2	Specifies part two of an optional domain key. If **key2** is zero, then **key1** is considered a domain name. Optional
taskinfo	Specifies Workflow task information that will be recorded in event records. Optional

Returns:

UXP Object identifier

sf::openFile (id, filename)

Opens a virtual file for reading.

Parameters:

id	**UXP Object** identifier
name	Name of the virtual file to open.

Returns:

Virtual file handle

sf::openIndex (filename [, maxrec])

Opens an index that can be used to fetch data for an open **UXP Object**. The index is a CSV file that must have the following columns:

1. Record number
2. Starting offset within the file
3. Ending offset within the file

Parameters:

filename	Index file name.

maxrec	Returns the maximum record number. Optional.

Returns:

Index identifier.

sf::openDrive (filename [, mode, mountpoint, responses])

Opens an existing **UXP Object** as a drive. Once open, the **UXP Object** contents will be accessible by applications that are not aware of the **UXP Technology**.

Parameters:

filename	**UXP Object** file name.
mode	Sets the access mode. Possible values are: • **ShareAll** Opens the **UXP Object** for shared read and writing. This is the default mode. • **ShareReadOnly** Mounts the **UXP Object** in read-only mode. Will only be permitted if the **UXP Object** permits read-only access. Optional
mountpoint	Specifies a folder path or drive letter to associate the open **UXP Object**. The path or drive must not exist as the mount operation will create the path. If not specified, the mountpoint will be automatically generated. For non-Windows systems, the mountpoint will default to the folder **Sertainty Drives** under the current user's home folder. Optional
responses	Provides initial challenge-response pairs. If all required responses are found in the list, then the **UXP Object** will not prompt the user for additional challenge responses. The format of the list is: **name1=value1|name2=value2 ...** Optional

Returns:

The mountpoint of the open drive.

sf::openDriveSSO (session, filename [, mode, mountpoint])

Opens an existing **UXP Object** as a drive using the **UXP ID single-sign-on** protocol. Once open, the **UXP** contents will be accessible by applications that are not aware of the **UXP Technology**.

Parameters:

session	Session identifier as returned from the **id::openSession** function.

filename	**UXP Object** file name.
mode	Sets the access mode. Possible values are: • **ShareAll** Opens the **UXP Object** for shared read and writing. This is the default mode. • **ShareReadOnly** Mounts the **UXP Object** in read-only mode. Will only be permitted if the **UXP Object** permits read-only access. Optional
mountpoint	Specifies a folder path or drive letter to associate the open **UXP Object**. The path or drive must not exist as the mount operation will create the path If not specified, the mountpoint will be automatically generated. For non-Windows systems, the mountpoint will default to the folder **Sertainty Drives** under the current user's home folder. Optional

Returns:

True if successful.

sf::openUxp (filename [, mode, responses, key1, key2, taskinfo])

Opens an existing **UXP**.

Parameters:

filename	**UXP Object** file name.
mode	Sets the access mode. Possible values are: • **ShareAnon** Opens the **UXP Object** for anonymous writes. This mode permits any user to add a file to the **UXP Object**. The user cannot read or modify existing objects, nor will it permit adding directories. Only top-level files may be anonymously added. For this mode, authentication is not performed. • **ShareAll** Opens the **UXP Object** for shared read and writing. This is the default mode. • **ShareNone** Opens the **UXP Object** for exclusive access. • **ShareReadOnly**

	Opens the **UXP Object** in read-only mode. Will only be permitted if the **UXP Object** permits read-only access. Optional
responses	Provides initial challenge-response pairs. If all required responses are found in the list, then the **UXP Object** will not prompt the user for additional challenge responses. The format of the list is: **name1=value1|name2=value2 ...** Optional
key1	Specifies an optional domain key or name. If empty, the public domain will be used and **key2** will be ignored. Optional
key2	Specifies part two of an optional domain key. If **key2** is zero, then **key1** is considered a domain name. Optional
taskinfo	Specifies Workflow task information that will be recorded in event records. Optional

Returns:

Identifier representing an open UXP Object.

sf::openUxpSSO (session, filename [, mode])

Opens an existing **UXP Object** using the **UXP ID single-sign-on** protocol.

Parameters:

session	Session identifier as returned from the **id::openSession** function.
filename	**UXP Object** file name.
mode	Sets the access mode. Possible values are: • **ShareAll** Opens the **UXP Object** for shared read and writing. This is the default mode. • **ShareNone** Opens the **UXP Object** for exclusive access. • **ShareReadOnly** Opens the **UXP Object** in read-only mode. Will only be permitted if the **UXP Object** permits read-only access. Optional

Returns:

Identifier representing an open UXP Object.

sf::posFile (id, handle)

Gets the current read position on an open virtual file.

Parameters:

id	**Identifier representing an open UXP Object.**
handle	Virtual file handle.

Returns:

Current position.

sf::quickProtect (ID, infile, outfile [, users, pagesize, cachesize, options])

Protects a file using the specified **UXP ID**. The protected data is under the control of the current **UXP ID**.

Parameters:

ID	**UXP ID** file specification used to protect data.
infile	File to protect
outfile	File containing protected data.
users	List of users who are permitted access to the protected entity. Multiple users can be separated using a '|' character, e.g. "joe|phil".
pagesize	Specifies a block size for storing protected data. Valid values are: -1 indicates default value A value greater than 512 and less than 8000000. Optional
cachesize	Specifies the number of cache buffers to use when reading a protected file. Value must be between 0 and 20. A zero disables caching. Optional
options	Command options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress** Compress the data while protecting it. • **ReadWrite** When creating the protected entity, the **ReadWrite** architecture will be used. • **Replace** Replace the existing output file, if necessary.

	• **WriteOnce** When creating the protected entity, the **WriteOnce** architecture will be used. Optional.

Returns:

True if successful.

sf::recordEvent (id, 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:

id	**UXP Object** identifier
action	Event action code. Must be a value between 24 and 50.
status	Status of action. A one indicates success; all other values indicate non-success.
msg	A user-supplied message for 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

Returns:

True if successful.

sf::readFile (id, handle, outbuf, len)

Reads data from a virtual file at the current logical position within the virtual file. Once the data has been read, the logical position is incremented by the number of bytes read.

Parameters:

id	**UXP Object** identifier
handle	Virtual file handle.
outbuf	Buffer to receive data.
len	Number of bytes to read from the current logical position within the virtual file.

Returns:

Number of bytes read. A zero indicates the end of the virtual file has been reached.

sf::renameFile (id, name, newname)

Renames the specified virtual file within the **UXP Object**.

Parameters:

id	**UXP Object** identifier
name	Name of the virtual file.
newname	New name of the virtual file.

Returns:

True if successful.

sf::seekFile (id, handle, pos)

Sets the current read position on an open virtual file.

Parameters:

id	**Identifier representing an open UXP Object.**
handle	Virtual file handle.
pos	Desired position to set read marker.

Returns:

Current position.

sf::setAttribute (id, name, value)

Sets the specified **UXP Object** attribute.

Parameters:

id	**UXP Object** identifier
name	Attribute name. Possible values are: • **COMPANY** • **DESCRIPTION** • **HOSTURL** • **NAME** • **OWNER**
value	Value to assign to the attribute.

sf::setAutoOpen (id, name)

Specifies a virtual file to automatically open within the **UXP Object Assistant**.

Parameters:

id	**UXP Object** identifier
name	Virtual file to open

Returns:

True if successful.

sf::setCallback (procedure-name)

Declares the specified procedure as a callback procedure for **UXP Object** authentication. The procedure must exist and will be called during **UXP Object** authentication if a challenge is requested.

Parameters:

procedure-name	Name of an existing procedure. The procedure should expect no arguments. The procedure can get the current **UXP Object** identifier from the built-in parameter p1. Upon completion, the procedure should set the built-in parameter p2 to either true or false. True indicates the user wishes to continue with authentication. A false value will trigger a cancel.

Example:

```
/**
 * Callback function to simulate a human response
 * Parameters:
 *    p1 must contain the UXP identifier
 *    p2 should be set to the status of the call. True indicates
 *    the responses have been provided.  False indicates cancel.
 */
replace procedure callback()
{
  int sf = p1;
  string buf = sf::getChallenges(sf);
  list list2, challenges = split(buf,"||");
  int i, cnt2, cnt = countList(challenges);
  string ch, name, prompt, response;
  string outbuf;

  for (i = 0; i < cnt; i++)
  {
    list2 = split(getList(challenges, i), '|');
    cnt2 = countList(list2);

    name = getList(list2, 0);
    prompt = getList(list2, 1);

    if (strlen(outbuf) > 0)
      outbuf = concat(outbuf, "||");

    if (name == "USERNAME")
    {
      response = "Admin";
    }
    else
    {
      response = substr(name, 10, 2);
```

```
    response = concat("Response ", response);
   }

   outbuf = concat(outbuf, name, "|", prompt, "|", response);
  }

  sf::setResponses(sf, outbuf);

  p2 = true;
 }
```

Returns:

True if successful.

sf::setCompliance (id, property, value)

Gets the specified **UXP Object** attribute.

Parameters:

id	**UXP Object** identifier
property	Property name. Possible values are: • **Expiration – Sets the expiration date/time when UXP becomes inaccessible.** • **Locked – Permanently locks the compliance properties from further changes.** • **ReadOnly – Permanently sets the UXP to read-only.**
value	Property value.

Returns:

True if successful.

sf::setCreatorInfo (id, name)

Sets the specified **UXP Object** creator.

Parameters:

id	**UXP Object** identifier
name	Creator name.

Returns:

True if successful.

sf::**setReality (id, name)**

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

Parameters:

id	**UXP Object** identifier
name	Name of the reality.

Note: Only available with AltReality privileges.

Returns:

True if successful.

sf::setResponses (id, responses)

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

Parameters:

id	**UXP Object** identifier
Responses	Packed list of responses for the current authentication session. The format of the response list is: **Name1|Prompt1|Response1||Name2|Prompt2|Response2 …** Where: • Name is the challenge name. • Prompt is the challenge prompt. • Response is the user-supplied response value..

Returns:

True if successful.

sf::showAttributes (id)

Shows the **UXP Object** attributes.

Parameters:

id	**UXP Object** identifier

Returns:

True if successful.

sf::showEvents (id)

Shows the event information for the **UXP Object**.

Parameters:

id	**Identifier representing an open UXP Object.**

Returns:

True if successful.

sf::showSignatures (id)

Shows the signatures for the **UXP Object**.

Parameters:

id	**Identifier representing an open UXP Object.**

Returns:

True if successful.

sf::showUxps ()

Shows a summary of open **UXP Objects**.

Parameters:

None

Returns:

True if successful.

sf::split (infile, id, outfile, size-option, size[, options])

Splits the specified file into chunks and creates a **UXP Object** for each segment. The segments can be accessed by an authenticated user; however, to reconstruct the segments into the original clear file, the metadata file must be utilized.

Parameters:

infile	Specifies the source file to split.
id	Specifies the **UXP ID** file uses to create the individual **UXP Object** split.
outfile	Specifies the output location, base file name and file type for the file splits. Each split will contain the base file name and file type, plus a unique identifier.
size-option	Specifies the sizing approach when splitting the file. Possible values: **2.** The size parameter represents the maximum size of a file split. **3.** The size parameter represents the number of desired file splits.
size	Specifies either the maximum size of a file split or the maximum number of splits.
options	Join options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compress**

	Compress data within **UXP Object**. • **Delete** Delete the sources file after a successful split operation. • **Replace** Replace the file if it already exists. Optional

Returns:

Buffer containing metafile document.

sf::touchUxp (filename)

Touches a **UXP Object**. A touch operation may execute user-independent rules that test global schedules, location, device, etc. A touch is a fast, simple mechanism for testing a without authenticating the user.

Parameters:

filename	**UXP Object** file name.

Returns:

True if **UXP Object** still permits access.

sf::validateUxp (filename)

Validates a **UXP**.

Parameters:

filename	**UXP Object** to validate.

Returns:

True if successful.

sf::viewFile (id, filename)

Displays the contents of the specified virtual file within the **UXP Object**.

Parameters:

id	**UXP Object** identifier
filename	Virtual file name.
options	View options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **noprompt**

	Disables prompting after a page of data has been displayed. Optional.

Returns:

True if successful.

sf::writeFile (id, handle, buf)

Writes data to a virtual file. Currently, only new virtual files can be written.

Parameters:

id	**UXP Object** identifier
handle	Virtual file handle.
buf	Buffer to save.

Returns:

True if successful.

Table 2

sql::bindCount	**Gets the number of bind parameters in a prepared query.**
sql::bindValue	Binds a value to a dynamic SQL parameter.
sql::columnCount	Gets the number of columns returned by a query.
sql::columnName	Gets the name of a column returned by a query.
sql::columnTable	Gets the parent table name for a selected column.
sql::columnValue	Gets a column value returned by a query.
sql::exec	Executes a query.
sql::execScript	Executes a buffer as a script of SQL statements.
sql::freeQuery	Deletes a query ID.
sql::newQuery	Creates a new query ID for executing SQL statements.
sql::next	Fetches the next row from an executed query.
sql::prepare	Prepares a SQL query for execution.

sql::bindValue (query-id, col, value)

Binds a value to a dynamic SQL parameter.

Parameters:

query-id	Query identifier.

col	Column number to bind. Column numbers are relative offsets within the source statement starting a zero. A bind parameter is represented in the statement by a question mark character. Example: select * from t1 where c1 = ?
value	A data value to be inserted as a parameter.

Returns:

True if successful.

sql::columnCount (query-id)

Gets the number of columns returned by a query.

Parameters:

query-id	Query identifier.

Returns:

Number of columns returned by the executed query.

sql::columnName (query-id, col)

Gets the name of a column returned by a query..

Parameters:

query-id	Query identifier.
col	Column number offset. Offsets begin at zero and represent the relative column number returned by the executed query.

Returns:

Column name.

sql::columnTable (query-id, col)

Gets the parent table name for a selected column.

Parameters:

query-id	Query identifier.
col	Column number offset. Offsets begin at zero and represent the relative column number returned by the executed query.

Returns:

Parent table name of the column.

sql::columnValue (query-id, col)

Gets a column value returned by a query.

Parameters:

query-id	Query identifier.
col	Column number offset. Offsets begin at zero and represent the relative column number returned by the executed query.

Returns:

Column value.

sql::exec (query-id, [sql])

Executes a query.

Parameters:

query-id	Query identifier.
sql	Optional SQL statement. If omitted, the SQL statement must be prepared using the **sql::prepare** function.

Returns:

True if successful.

sql::execScript (query-id, script-buffer)

Executes a buffer as a script of SQL statements.

Parameters:

query-id	Query identifier.
script-buffer	Buffer containing one or more SQL statements. Each statement must be terminated with a semi-colon. In addition to SQL statements, the script executor supports the SQL comment "--" as the first characters on a line. A print command is also supported: print "your-text";

Returns:

True if successful.

sql::freeQuery (query-id)

Deletes a query ID.

Parameters:

query-id	Query identifier to delete.

Returns:

True if successful.

sql::newQuery (uxp-id)

Creates a new query ID for executing SQL statements.

Parameters:

uxp-id	UXP file ID as provided by the sf::openUxp or sf::newUxp functions. The open UXP must be authenticated and have a ReadWrite format.

Returns:

Query ID.

sql::next (query-id)

Fetches the next row from an executed query. The query must be executed by the sql::exec function prior to retrieving rows.

Parameters:

query-id	Query identifier.

Returns:

True if a row has been fetched. False if no more rows exist.

sql::prepare (query-id, sql)

Prepares a SQL query for execution.

Parameters:

query-id	Query identifier.
sql	SQL statement to prepare for execution. The statement must be a valid SQL statement and should not be terminated with a semi-colon.

Returns:

True if successful.

str::decode (source [, key])

Decodes a string that was previously encoded.

Parameters:

source	String to be decoded. If the source string was not decoded, the source string will be returned by the call.
key	Optional key to use when encoding. If empty, a default key will be use.

Returns:

Decoded string.

str::encode (source [, key])

Encodes the specified source string using encryption.

Parameters:

source	String to be encoded.
key	Optional key to use when encoding. If empty, a default key will be use.

Returns:

Encoded string.

str::replace (&source, search, replace)

Replaces a token string with another string.

Parameters:

source	String containing characters to replace. The ampersand is required, otherwise, the source will be passed by value.
search	String to replace.
replace	String to insert into source string.

Returns:

Source string.

util::fromBase64 (buffer)

Decodes a Base64 buffer.

Parameters:

filename	Name of the file to read.

Returns:

Buffer containing decoded data.

util::toBase64 (buffer)

Encodes a buffer to Base64.

Parameters:

buffer	Buffer to write

Returns:

Buffer containing encoded data.

util::getEnv (var [, def])

Gets a system environment variable.

Parameters:

var	Variable to read.
def	Value to return if variable was not found or is empty.

Returns:

Value of variable as a string.

util::getHost ( )

Gets the current system host name.

Parameters:

None

Returns:

Host name as a string.

util::getPref (pref [, def])

Gets a user preference for the **UXP** environment.

Parameters:

pref	Preference to read.
def	Value to return if preference was not found.

Returns:

Value of preference as a string.

util::getSysPref (pref [, def])

Gets a system preference for the **UXP** environment.

Parameters:

pref	Preference to read.
def	Value to return if preference was not found.

Returns:

Value of preference as a string.

util::setPref (pref, value)

Sets a user preference for the **UXP** environment.

Parameters:

pref	Preference to set.
value	Value to save

Returns:

True if operation was successful.

util::setSysPref (pref, value)

Sets a system preference for the **UXP** environment.

Parameters:

pref	Preference to set.
value	Value to save

Returns:

True if operation was successful.

workflow::absoluteConfigPath (host, name)

Translates a workspace configuration name into its actual file specification.

Parameters:

host	Specifies the host containing the configuration.
name	Specifies the workspace configuration name. The actual configuration does not have to exist.

Returns:

File specification of the actual or proposed library configuration.

workflow::activate (host, name)

Activates the specified host and configuration. An active configuration will be used by current **Sertainty Agent**.

Parameters:

host	Specifies the host containing the configuration.

name	Specifies the name of the configuration.

Returns:

True if successful.

workflow::applyPreset (taskname, name)

Applies the specified preset to an existing **Task**.

Parameters:

taskname	Specifies the **Task** name.
name	Specifies the preset to apply.

Returns:

True if successful.

workflow::configExists (name)

Determines if the specified configuration exists.

Parameters:

name	Specifies the name of the configuration to test.

Returns:

True if exists.

workflow::createFolders (config, task])

Creates the underlying folders for the specified **Tasks**.

Parameters:

config	Specifies the name of the configuration. If empty, all configurations and all **Tasks** will be scanned for missing folders.
task	Specifies the **Task** name. If empty, all **Tasks** for the specified configuration will be scanned for missing folders.

Returns:

True if operation was successful.

workflow::deleteConfig (name)

Deletes the specified configuration.

Parameters:

name	Specifies the name of the configuration.

Returns:

True if successful.

workflow::deleteTask (config , name)

Deletes an existing **Task** definition within a configuration.

Parameters:

config	Specifies the configuration name.
name	Specifies the name of an existing **Task**.

Returns:

True if successful.

workflow::duplicateConfig (name [, newname])

Duplicates the specified configuration within the **Workflow** system.

Parameters:

name	Specifies the name of the configuration.
newname	Specifies the name of the duplicate configuration.

Returns:

Name of the new configuration. Empty if there was an error.

workflow::duplicateTask (config, task, newconfig [, newtask, options])

Duplicates a **Task** within the same configuration or in a different host or configuration.

Parameters:

config	Specifies the configuration name.
task	Specifies the original **Task** name.
newconfig	Specifies new configuration name.
newtask	Specifies the new **Task** name. If empty, the original host name plus a suffix will be generated.
options	Specifies options. Possible values: • **Replace** Will replace an existing **Task** with same name.

Returns:

Name of the new **Task**. Empty if there was an error.

workflow::executeTask (config, taskname [, path])

Manually executes a **Task**. This will execute the defined **Task** regardless of the **Agent** status.

Parameters:

config	Specifies the configuration name.
taskname	Specifies the **Task** to execute.
path	Optional folder path to pass to the **Task**. If empty, the default folder specified in the **Task** definition will be used.

Returns:

True if successful.

workflow::findScript (script-name)

Finds the specified script by first looking in the script folder of the active configuration and then in the shared script folder.

Parameters:

script-name	Specifies the script name to find.

Returns:

Full script file specification. The specification will be empty if the script was not found.

workflow::getConfigs (list)

Gets a list of valid configuration names. The **Workflow** system contains configuration files that can be deployed as **Workflow** configurations.

Parameters:

list	Specifies the list to receive the configuration names.

Returns:

True if successful.

workflow::getPresets (list)

Gets a list of valid **Task** preset names.

Parameters:

list	Specifies the list to receive the preset names.

Returns:

True if successful.

workflow::getProperty (name)

Gets a **Workflow** system property.

Parameters:

name	Specifies the property to read. Possible values: • ActiveConfig • ApplicationName • ApplicationVersion • TasksEnabled

Returns:

Property value.

workflow::getTaskProperty (config , task, name)

Gets a **Task** property from a **Task**.

Parameters:

config	Specifies the configuration name.
task	Specifies the **Task** name.
name	Specifies the property to read. Possible values: • Name • Type • Enabled • Notes • Executable • ExecutableType • LogFile • FileScanner • Folder • MaxFiles • Include • Exclude • ScanInterval • StartHour • StartMinute • Parameter_1 • Parameter_2 • Parameter_3 • Parameter_4 • Parameter_5 • Parameter_6 • Parameter_7 • Parameter_8 • WatchInterval

Returns:

Property value.

workflow::getTasks (config , list)

Gets a list of **Task** names from within a configuration.

Parameters:

config	Specifies the configuration name.
list	Specifies the list to receive the **Task** names.

Returns:

True if successful.

workflow::importConfig (infile, name, options)

Imports an external configuration XML file into the workflow system. The **Workflow** system contains configuration files that can be deployed as **Workflow** configurations.

Parameters:

infile	Specifies the XML file containing a valid configuration.
name	Specifies the name of the imported configuration. The name must be unique among configurations.
options	Specifies options. Possible values: • **Replace** Will replace an existing configuration.

Returns:

True if successful.

workflow::newConfig (name [, options])

Opens a new configuration setup.

Parameters:

name	Specifies the name of the new configuration. The name must be unique among host configurations.
options	Specifies options. Possible values: • **Replace** Will replace an existing configuration.

Returns:

True if successful.

workflow::newTask (config , name)

Creates a new **Task** definition within a configuration.

Parameters:

config	Specifies the configuration name.
name	Specifies the name of the new **Task**. The **Task** name must be unique within the specified configuration.
options	Specifies options. Possible values: • **Replace** Will replace an existing **Task**.

Returns:

True if successful.

workflow::renameConfig (name, newname)

Renames the specified configuration.

Parameters:

name	Specifies the name of the configuration to rename.
newname	Specifies the new configuration name.

Returns:

True if successful.

workflow::setProperty (name, value)

Sets a workflow system property.

Parameters:

name	Specifies the property to save. Possible values:
value	Specifies the property value to save. Possible values: • ActiveConfig • ApplicationName • ApplicationVersion • TasksEnabled

Returns:

True if successful.

workflow::setTaskProperty (config, task , name, value)

Sets a **Task** property.

Parameters:

config	Specifies the configuration name.
task	Specifies the **Task** name.
name	Specifies the property to save. Possible values: • Name • Type • Enabled • Notes • Executable • ExecutableType • LogFile • FileScanner • Folder • MaxFiles • Include • Exclude • ScanInterval • StartHour • StartMinute • Parameter_1 • Parameter_2 • Parameter_3 • Parameter_4 • Parameter_5 • Parameter_6 • Parameter_7 • Parameter_8 • WatchInterval
value	Specifies the property value to save.

Returns:

True if successful.

x::batch (cmd [, log-prefix])

Runs a script command in a separate process. Once the command completes, the process will terminate and leave a log in a local file with the template scriptXXXXXX.log.

Parameters:

cmd	Script command to execute.
log-prefix	Optional prefix to use for the standard output of the batch operation. **Default:** script

Returns:

True if successful.

x::cancelOutput ()

Cancels the recording of operation output.

Parameters:

None

Returns:

True if successful.

x::compile (infile, outfile [, macros, options])

Compiles the specified **UXL** source file.

Parameters:

infile	**UXL** source file.
outfile	Output file specification.
macros	A list of optional macros, each separated by a comma character. Optional
options	Compile options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **IncScript** Include scripting code in compilation. Without this flag, the compiler will skip any statement that is not within a rule or procedure. • **NoOptimize** Optimization will eliminate unnecessary code. It will also mangle variable names, which will make it difficult to interpret runtime errors. • **Replace** Replace the output file if it already exists. Optional

Note: Only available with Developer privileges.

Returns:

True if successful.

x::executeFile (infile [, options])

Executes the specified compiled **UXL** file.

Parameters:

infile	UXL file.
options	Execution options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Compiled** Execute the file as compiled **UX**L. • **Source** Execute the file as **UX**L source. This is the default setting. Optional

Returns:

True if successful.

x::loadPackage (name [, library])

Loads optional functions.

Parameters:

name	Name of the optional package. Possible values: • **agent** • **file** • **id** • **sf** • **str** • **util** Package names are used as the prefix to respective function names. **Example**: workflow::openConfiguration(). **Note:** An asterisk (*) will load all available packages.
library	Shared library name. The library must reside in the same folder as the Sertainty core library. **Optional**

Returns:

True if successful.

x::exportConfiguration (name, outfile [, options])

Exports the current address configuration to a protected configuration file. An address configuration consists of the current operating device and the network location. An exported configuration file can be imported into a UXP **ID** profile within the **UXP Object Assistant**.

Parameters:

name	User-supplied name to be assigned to the configuration.
outfile	Output file specification.
options	Compile options. Multiple options can be separated using a '|' character, e.g. "full|recurse". The following keywords are supported: • **Replace** Replace the output file if it already exists. Optional

Returns:

True if successful.

x::printToLog (message)

Prints an information message to the current application log.

Parameters:

message	Message to print.

Returns:

True if successful.

x::prompt (prompt [, defvalue])

Prompts the user for input.

Parameters:

prompt	Specifies the prompt to display to the user.
defvalue	Default value when user presses Enter without providing data. Optional.

Returns:

User value

x::setOutput (filename)

Records any output to the specified file.

Parameters:

filename	The file to receive output from all subsequent operations.

Returns:

True if successful.

x::setProgress (level)

Sets the level of progress when performing long operations.

Parameters:

level	Specifies the level. A zero disables progress messages. The larger the value, the more verbose the messages are.

Returns:

True if successful.

x::shell (cmd)

Executes an external shell command.

Parameters:

cmd	Command to execute.

Note: Only available with Developer privileges.

Returns:

True if successful.

x::showConfiguration ( )

Shows the current device and location configuration.

Parameters:

None

Returns:

True if successful.

x::showLicense ([filename])

Shows the attributes of the specified license.

Parameters:

filename	Shows the license file contents as specified in the filename parameter. If data is not specified, it will show the current license.

Returns:

True if successful.

x::showOutput ()

Shows the current file for all display output.

Parameters:

None

Returns:

True if successful.

x::writeMachineCache () (Linux-only)

Writes a snapshot of current device in a private cache. The machine cache is optional; however, in order for the **UXP** system to retrieve privileged machine information (Linux), the cache can be populated by a process with root privileges. The cache can then be read by non-privileged **UXP**-based applications.

Parameters:

None

Example: populate the cache with necessary data:

sudo /opt/Sertainty/bin/SertaintyScript.sh ‘x::writeMachineCache();’ -b

Returns:

True if successful.