USP SECURE ENTRY SERVER®

Documentation series

UNITED SECURITY PROVIDERS

Secure Login Service

Scripting Guide

Version 5.19.0.4

United Security Providers AG
www.united-security-providers.ch
info@united-security-providers.ch

Headquarter Stauffacherstrasse 65/15 CH-3014 Bern Tel. +41 31 959 02 02
Baslerpark Mürtschenstrasse 27 CH-8048 Zürich Tel. +41 44 496 61 11

UNITED SECURITY PROVIDERS

Contents

Chapter 1

AbstractCertificateHolder

Base class for the support of signing and signature verification in JEXL. It can be extended for support of custom security providers, types of certificates and formats. This prefix is usually NOT used directly! Instead, other JEXL functions such as "function.getCertificate()" will return an object of this type, on which all these functions can be invoked.

1.1 addFingerprint()

- **Signature**:

```
public void addFingerprint(String key, String value)
```

Add fingerprint. * **parameter**: key * **parameter**: value

1.2 addSignature()

- **Signature**:

```
public String addSignature(String dataToSign, String signaturePrefix, String ↩
    signatureSuffix)
```

Creates and adds a signature to the given data string.

- **parameter**: dataToSign

The data string to which to add a signature.

- **parameter**: signaturePrefix

The prefix string which separates the signature from the data. If it is null, none will be added.

- **parameter**: signatureSuffix

The optional suffix string. If it is null, none will be added.

- **return**:

The data string and the added signature.

1.3 getCertificate()

- **Signature**:

```
public Certificate getCertificate()
```

Certificate getter

- **return**:

The certificate/

1.4 getFingerPrint()

- **Signature**:

```
public String getFingerPrint(String hashType)
```

Returns a fingerprint string of this certificate.

- **parameter**: hashType

The hashing algorithm of the fingerprint, like "MD5" or "SHA1".

- **return**:

The fingerprint string.

1.5 getLastFileChange()

- **Signature**:

```
public long getLastFileChange()
```

Returns the last file change timestmap.

- **return**:

The last file change timestamp.

1.6 getPassword()

- **Signature**:

```
public char[] getPassword(String alias)
```

Returns the password configured for the certificate with the given alias.

- **parameter**: alias

The alias of the certificate.

- **return**:

password The password as an array of 'char's.

1.7 getPrivateKey()

- **Signature**:

```
public PrivateKey getPrivateKey()
```

Private key getter.

- **return**:

The private key of this certificate.

1.8 getPublicKey()

- **Signature**:

```
public PublicKey getPublicKey()
```

Public key getter.

- **return**:

The public key of this certificate.

1.9 setLastFileChange()

- **Signature**:

```
public void setLastFileChange(long lastFileChange)
```

Sets the last file change.

- **parameter**: lastFileChange

The timestamp of the last file change.

1.10 signData()

- **Signature**:

```
public String signData(String dataToSign)
```

Signs the given data using the private key.

- **parameter**: dataToSign

The data string for which to create a signature.

- **return**:

the signed data, if it succeed. The unsigned data, otherwise.

1.11 verifyData()

- **Signature**:

```
public boolean verifyData(String dataToVerify, String entriesSeparator, String ↩
    keyValuePairSeparator, String signatureKey)
```

Verifies the given data and signature. The method expects the data to consist of key-/value pairs. It will parse the string based on the given entry and key-/value separator characters, extract the signature based on the given signature key ID, and then verify it for the preceding data.

- **parameter**: dataToVerify

The key-/value pair string, e.g. *user=xy,age=35,timestamp=183025012011,sig=HA23SD...*.

- **parameter**: entriesSeparator

The character which separates the various key-/value pair entries from each other, e.g. *,*.

- **parameter**: keyValuePairSeparator

The character which separates a key from the corresponding value, e.g. *=*.

- **parameter**: signatureKey

The name (ID) of the key which contains the signature, e.g. *sig*.

- **return**:

true if the signature is valid, false if not.

1.12 verifySignature()

- **Signature**:

```
public boolean verifySignature(String data, String signature)
```

Verifies the signature using the public key, extracted from the certificate.

- **parameter**: data

The signed data that must match the given signature.

- **parameter**: signature

The signature that must be verified.

- **return**:

true if the signature was correct; false, otherwise.

Chapter 2

apidoc_saml_attribute

SAML: Documents the methods of the AttributeWrapper, which is returned, for example, by sp_assertion.getAssertionAttribute(name). Note that the prefix "apidoc_saml_attribute" cannot be used in JEXL expressions, it only serves as an ID in documentation.

2.1 getBase64DecodedValue()

- **Signature**:

```
public byte[] getBase64DecodedValue();
```

Returns the base64 decoded string value of the first attribute value or null if no values.

- **return**:

Base64 decoded first attribute value or null if no values.

- **since**: 4.32.0

2.2 getBase64DecodedValues()

- **Signature**:

```
public List<byte[]> getBase64DecodedValues();
```

Returns the base64 decoded string values.

- **return**:

List of base64 decoded string values (can be empty, never null).

- **since**: 4.32.0

2.3 getFriendlyName()

- **Signature**:

```
public String getFriendlyName();
```

Returns the attribute friendly name.

Returns the short name, typically a short string like "uid".

- **return**:

The attribute friendly name.

- **since**: 4.32.0

2.4 getName()

- **Signature**:

```
public String getName();
```

Returns the attribute name.

Returns the full/long name, typically a url like "urn:oid:0.9.2342.19200300.100.1.1".

- **return**:

The attribute name.

- **since**: 4.32.0

2.5 getNameFormat()

- **Signature**:

```
public String getNameFormat();
```

Returns the attribute name format.

Returns the full/long attribute name format, a urn like "urn:oasis:names:tc:SAML:2.0:attrname-format:basic".

- **return**:

The attribute name format.

- **since**: 4.32.0

2.6 getOpenSamlAttribute()

- **Signature**:

```
public Attribute getOpenSamlAttribute();
```

Returns a reference to the OpenSAML-API object instance which represents the attribute.

- **return**:

An instance of the class "org.opensaml.saml2.core.Attribute".

- **since**: 4.32.0

2.7 getShortNameFormat()

- **Signature**:

```
public String getShortNameFormat();
```

Returns the SLS' short attribute name format, "basic" or "uri" or null for other name formats.

This is the short notation for the name format used in idp JEXL functions and in IdP configuration properties.

- **return**:

The short attribute name format or null.

- **since**: 4.32.0

2.8 getShortValueType()

- **Signature**:

```
public String getShortValueType();
```

Returns the SLS' short value type of the attribute, e.g "string" or "base64" or null if something else.

This is the short notation for the value type used in idp JEXL functions and in IdP configuration properties.

- **return**:

The short value type (null if not known or no attribute values).

- **since**: 4.32.0

2.9 getValue()

- **Signature**:

```
public String getValue();
```

Returns the string value of the first attribute value or null if no values.

If base64 encoded, the encoded value is retured.

- **return**:

First attribute value or null if no values.

- **since**: 4.32.0

2.10 getValueType()

- **Signature**:

```
public String getValueType();
```

Returns the value type of the attribute, e.g "xs:string" or "xs:base64Binary".

- **return**:

The value type (null if no attribute values).

- **since**: 4.32.0

2.11 getValues()

- **Signature**:

```
public List<String> getValues();
```

Returns the string values.

If base64 encoded, the encoded values are retured.

- **return**:

List of string values (can be empty, never null).

- **since**: 4.32.0

Chapter 3

apidoc_saml_idp_entry

SAML: Documents the methods of the OpenSAML IDPEntry class, which is returned, for example, by idp_authn_request.getIdpListEntry(providerID). Note that the prefix "apidoc_saml_idp_entry" cannot be used in JEXL expressions, it only serves as an ID in documentation.

3.1 getLoc()

- **Signature**:

```
public String getLoc();
```

Gets the Loc value (the URL for authentication at the IdP, optional).

- **return**:

the Loc value or null if none

- **since**: 4.32.0

3.2 getName()

- **Signature**:

```
public String getName();
```

Gets the Name value (a human readable string that identifies the IdP, optional).

- **return**:

the Name value or null if none

- **since**: 4.32.0

3.3 getProviderID()

- **Signature**:

```
public String getProviderID();
```

Gets ProviderID URI (the IdP EntityID, mandatory).

- **return**:

the ProviderID URI

- **since**: 4.32.0

3.4 setLoc()

- **Signature**:

```
public void setLoc(String newLoc);
```

Sets the Loc value (the URL for authentication at the IdP, optional).

- **parameter**: newLoc

the new Loc value

- **since**: 4.32.0

3.5 setName()

- **Signature**:

```
public void setName(String newName);
```

Sets the Name value (a human readable string that identifies the IdP, optional).

- **parameter**: newName

the Name value

3.6 setProviderID()

- **Signature**:

```
public void setProviderID(String newProviderID);
```

Sets the ProviderID URI (the IdP EntityID, mandatory).

- **parameter**: newProviderID

the new ProviderID URI

- **since**: 4.32.0

Chapter 4

cache

Provides functions for using global caches (i.e. not per user/session).

4.1 clear()

- **Signature**:

```
public void clear(String cacheName)
```

Clear all objects from cache.

- **parameter**: cacheName

Name of the cache.

```
${cache.clear('mycache')}
```

- **since**: 4.26.0

4.2 createCache()

- **Signature**:

```
public void createCache(String cacheName, int maxElementsInMemory, long ↩
    timeToLiveSeconds, long timeToIdleSeconds)
```

Create cache with given name and parameters, unless there is already a cache with the given name. If there is already a cache with the same name, nothing is done, i.e. parameters of the existing cache are not changed.

- **parameter**: cacheName

Name of the cache.

- **parameter**: maxElementsInMemory

Maximal number of objects to store in memory (0 means no limit).

- **parameter**: timeToLiveSeconds

Time in seconds after which objects are removed from the cache in any case.

- **parameter**: timeToIdleSeconds

Time in seconds after which objects are removed from the cache if not accessed or modified.

```
${cache.createCache('mycache', 5000, 7200, 3600)}
```

- **since**: 4.26.0

4.3 destroyCache()

- **Signature**:

```
public void destroyCache(String cacheName)
```

Destroy cache with given name, has no effect if there is no cache with the given name.

- **parameter**: cacheName

Name of the cache.

```
${cache.destroyCache('mycache')}
```

- **since**: 4.26.0

4.4 get()

- **Signature**:

```
public Object get(String cacheName, String key)
```

Get object from cache.

- **parameter**: cacheName

Name of the cache.

- **parameter**: key

The key that identifies the object in the cache.

- **return**:

The value from the cache or null if none in the cache

```
${function.setVariable('some.value', cache.get('mycache', 'mykey'))}
```

- **since**: 4.26.0

4.5 put()

- **Signature**:

```
public void put(String cacheName, String key, Object value)
```

Put object into cache.

- **parameter**: cacheName

Name of the cache.

- **parameter**: key

The key that will identify the object in the cache.

- **parameter**: value

The value to store in the cache.

```
${cache.put('mycache', 'mykey', some.value)}
```

- **since**: 4.26.0

4.6 remove()

- **Signature**:

```
public Object remove(String cacheName, String key)
```

Remove object from cache.

- **parameter**: cacheName

Name of the cache.

- **parameter**: key

The key that identifies the object in the cache.

- **return**:

The value from the cache or null if none was in the cache

```
${cache.remove('mycache', 'mykey')}
```

- **since**: 4.26.0

Chapter 5

crypto

Contains JEXL/Groovy functions for "crypto." prefix

5.1 base32decode()

- **Signature**:

```
public String base32decode(String input)
```

Decodes a base32 string using UTF-8 byte to character encoding.

- **parameter**: input A Base32 string to decode.
- **return**:

A decoded string representation.

- **since**: 5.4.0.0

5.2 base32decode()

- **Signature**:

```
public String base32decode(String input, String charEncoding)
```

Decodes a base32 string using the given byte to character encoding.

- **parameter**: input A Base32 string to decode.
- **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".
- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.3 base32decodeToBytes()

- **Signature**:

```
public byte[] base32decodeToBytes(String input)
```

Decodes a base32 string to a byte array.

- **parameter**: input A Base32 string to decode.
- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.4 base32encode()

- **Signature**:

```
public String base32encode(String input)
```

Encodes a string to base32 using UTF-8 character to byte encoding.

- **parameter**: input A string to encode.
- **return**:

The encoded String.

- **since**: 5.4.0.0

5.5 base32encode()

- **Signature**:

```
public String base32encode(String input, String charEncoding)
```

Encodes a string to base32 using given character to byte encoding.

- **parameter**: input A string to encode.
- **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".
- **return**:

The encoded String.

- **since**: 5.14.0.0

5.6 base32encode()

- **Signature**:

```
public String base32encode(byte[] input)
```

Encodes a byte array to base32.

- **parameter**: input A byte array to encode.
- **return**:

The encoded String.

- **since**: 5.4.0.0

5.7 base32toHex()

- **Signature**:

```
public String base32toHex(String base32)
```

Converts the given base32 data to a hex string.

- **parameter**: base32

The base32 string to convert.

- **return**:

The resulting hex string.

- **since**: 5.4.0.0

5.8 base64UrlDecode()

- **Signature**:

```
public String base64UrlDecode(String input)
```

Decodes a base64URL string using UTF-8 byte to character encoding.

- **parameter**: input

A Base64URL string to decode.

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.9 base64UrlDecode()

- **Signature**:

```
public String base64UrlDecode(String input, String charEncoding)
```

Decodes a base64URL string using the given byte to character encoding.

- **parameter**: input

A Base64URL string to decode. * **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.10 base64UrlDecodeToBytes()

- **Signature**:

```
public byte[] base64UrlDecodeToBytes(String input)
```

Decodes a base64URL string to a byte array.

- **parameter**: input

A Base64URL string to decode.

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.11 base64UrlEncode()

- **Signature**:

```
public String base64UrlEncode(String input)
```

Encodes a string to base64URL using UTF-8 character to byte encoding.

- **parameter**: input

A string to encode.

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.12 base64UrlEncode()

- **Signature**:

```
public String base64UrlEncode(String input, String charEncoding)
```

Encodes a string to base64URL using given character to byte encoding.

- **parameter**: input

A string to encode. * **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.13 base64UrlEncode()

- **Signature**:

```
public String base64UrlEncode(byte[] input)
```

Encodes a byte array to base64.

- **parameter**: input

A byte array to encode.

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.14 base64decode()

- **Signature**:

```
public String base64decode(String input)
```

Decodes a base64 string using UTF-8 byte to character encoding.

- **parameter**: input

A Base64 string to decode.

- **return**:

A decoded string representation.

- **since**: 5.4.0.0

5.15 base64decode()

- **Signature**:

```
public String base64decode(String input, String charEncoding)
```

Decodes a base64 string using the given byte to character encoding.

- **parameter**: input A Base64 string to decode.
- **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".
- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.16 base64decodeToBytes()

- **Signature**:

```
public byte[] base64decodeToBytes(String input)
```

Decodes a base64 string to a byte array.

- **parameter**: input A Base64 string to decode.
- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.17 base64encode()

- **Signature**:

```
public String base64encode(String input)
```

Encodes a string to base64 using UTF-8 character to byte encoding.

- **parameter**: input

A string to encode.

- **return**:

The encoded String.

- **since**: 5.4.0.0

5.18 base64encode()

- **Signature**:

```
public String base64encode(String input, String charEncoding)
```

Encodes a string to base64 using given character to byte encoding.

- **parameter**: input

A string to encode. * **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.19 base64encode()

- **Signature**:

```
public String base64encode(byte[] input)
```

Encodes a byte array to base64.

- **parameter**: input

A byte array to encode.

- **return**:

The encoded String.

- **since**: 5.4.0.0

5.20 base64toHex()

- **Signature**:

```
public String base64toHex(String base64)
```

Converts the given base64 data to a hex string. This can be useful with the "md5" or "sha?" functions, which create Base64 data.

- **parameter**: base64

The base64 string to convert.

- **return**:

The resulting hex string.

- **since**: 5.4.0.0

5.21 createSecureRandomGenerator()

- **Signature**:

```
public SecureRandom createSecureRandomGenerator()
```

Creates an instance of the Java "SecureRandom" class. This allows to use the methods of that object (such as ".nextInt()" etc.) to generate pseudo-random values.

Please consult the official Java API documentation for the class "java.security.SecureRandom" to see all available methods.

```
#{rnd = crypto.createSecureRandomGenerator()}
```

- **return**:

The "SecureRandom" instance.

- **since**: 5.4.0.0

5.22 createSecureRandomGenerator()

- **Signature**:

```
public SecureRandom createSecureRandomGenerator(byte[] seed)
```

Creates an instance of the Java "SecureRandom" class, initialized with a given seed. This allows to use the methods of that object (such as ".nextInt()" etc.) to generate pseudo-random values with a sequence of values pre-defined by the seed.

Please consult the official Java API documentation for the class "java.security.SecureRandom" to see all available methods.

```
#{rnd = crypto.createSecureRandomGenerator(seed)}
```

- **parameter**: seed

An array of bytes used as seed for the randomizer.

- **return**:

The "SecureRandom" instance.

- **since**: 5.4.0.0

5.23 decrypt()

- **Signature**:

```
public byte[] decrypt(String alias, byte[] encryptedData)
```

Decrypts the given encrypted data by performing the decryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". Any salt in the encrypted data will be removed from the result.

The incoming data is a byte array that is expected to be structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes, after being decrypted, are expected to contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> This function handles all that and only returns the decrypted payload data.

```
action.1=#{setVar('plain', crypto.decrypt('alias', encData))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: encryptedData

The encrypted data.

- **return**:

The decrypted plain data, without salt - or null, if the given data was empty or invalid.

- **since**: 5.4.0.0

5.24 decryptToBytes()

- **Signature**:

```
public byte[] decryptToBytes(String alias, String encryptedDataBase64)
```

Decrypts the given encrypted Base64 data by first decoding the Base64 data to retrieve the encrypted data, and then performing the decryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias".

The incoming Base64 data contains a byte array that is expected to be structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes, after being decrypted, are expected to contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> This function handles all that and only returns the decrypted payload data.

```
action.1=#{setVar('plain', crypto.decryptToBytes('alias', base64))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: encryptedDataBase64

The encrypted Base64 data.

- **return**:

The decrypted plain data (may be empty, but never null), without salt. @throws Exception If the decryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.25 decryptToString()

- **Signature**:

```
public String decryptToString(String alias, String encryptedDataBase64)
```

Decrypts the given encrypted Base64 data by first decoding the Base64 data to retrieve the encrypted data, and then performing the decryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". The string created from the encrypted data assumes that the econding of the encrypted data was UTF-8. If that is not the case, use the alternative function of the same name that allows to also specify the encoding of the incoming data.

The incoming Base64 data contains a byte array that is expected to be structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes, after being decrypted, are expected to contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> This function handles all that and only returns the decrypted payload data.

```
action.1=#{setVar('text', crypto.decryptToString('alias', base64))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: encryptedDataBase64

The encrypted Base64 data.

- **return**:

The decrypted plain data as a string (created with UTF-8 encoding), without salt. @throws Exception If the decryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.26 decryptToString()

- **Signature**:

```
public String decryptToString(String alias, String encryptedDataBase64, String ↩
    encoding)
```

Decrypts the given encrypted Base64 data by first decoding the Base64 data to retrieve the encrypted data, and then performing the decryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". The string created from the encrypted data is created with the specified encoding (so make sure that the incoming data really matches this encoding).

The incoming Base64 data contains a byte array that is expected to be structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes, after being decrypted, are expected to contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> This function handles all that and only returns the decrypted payload data.

```
action.1=#{setVar('text', crypto.decryptToString('alias', base64))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: encryptedDataBase64

The encrypted Base64 data.

- **parameter**: encoding

The encoding of the encrypted string data.

- **return**:

The decrypted plain data as a string, without salt - or null, if the given data was empty or invalid. @throws SLSJexlRuntimeException If the selected encoding was not supported by the JVM.

- **since**: 5.4.0.0

5.27 encrypt()

- **Signature**:

```
public byte[] encrypt(String alias, byte[] plainData)
```

Executes the encryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". Returns the encrypted data as a raw byte array.

The resulting data is a byte array that is structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> Any function that receives this data and attempts to decrypt it must take this data structure into consideration. The decryption functions in this API do all that automatically.

```
action.1=#{setVar('data', crypto.encrypt('alias', plainData))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: plainData

The plain data to encrypt.

- **return**:

The encrypted data (may be empty, but never null). @throws Exception If the encryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.28 encryptToBase64()

- **Signature**:

```
public String encryptToBase64(String alias, byte[] plainData)
```

Executes the encryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". Returns the encrypted data as a Base64 string.

The resulting Base64 data contains a byte array that is structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> Any function that receives this data and attempts to decrypt it must take this data structure into consideration. The decryption functions in this API do all that automatically.

```
action.1=#{setVar('data', crypto.encryptToBase64('alias', plainData))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: plainData

The plain data to encrypt.

- **return**:

The encrypted data (may be empty, but never null). @throws Exception If the encryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.29 encryptToBase64()

- **Signature**:

```
public String encryptToBase64(String alias, String plainData)
```

Executes the encryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". Returns the encrypted data as a Base64 string. The string within the Base64 data is UTF-8 encoded.

The resulting Base64 data contains a byte array that is structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> Any function that receives this data and attempts to decrypt it must take this data structure into consideration. The decryption functions in this API do all that automatically.

```
action.1=#{setVar('data', crypto.encryptToBase64('alias', plainData))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: plainData

The plain data to encrypt.

- **return**:

The encrypted data (may be empty, but never null). @throws Exception If the encryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.30 encryptToBase64()

- **Signature**:

```
public String encryptToBase64(String alias, String plainData, String encoding)
```

Executes the encryption operation defined by the configuration properties with the prefix "crypto.", followed by the given "alias". Returns the encrypted data as a Base64 string. In addition, it allows to define the encoding of the data. NOTE: The incoming string in the parameter "plainData" is a Java Unicode string; it will be converted to a string with the given encoding before being encrypted.

The resulting Base64 data contains a byte array that is structured as follows: <pre> [IV bytes (if configured)]<encrypted data>.

</pre> And the "<encrypted data>" bytes contain the salt bytes at the beginning: <pre> <salt bytes><payload data>

</pre> Any function that receives this data and attempts to decrypt it must take this data structure into consideration. The decryption functions in this API do all that automatically.

```
action.1=#{setVar('data', crypto.encryptToBase64('alias', plainData))}
```

- **parameter**: alias

The alias of the configuration properties for this operation.

- **parameter**: plainData

The plain data to encrypt.

- **parameter**: encoding

The target encoding of the string data in the base64 data.

- **return**:

The encrypted data (may be empty, but never null). @throws Exception If the encryption operation failed (which is usually caused by a configuration problem, or by invalid input data).

- **since**: 5.4.0.0

5.31 getSecureRandomBoolean()

- **Signature**:

```
public boolean getSecureRandomBoolean()
```

Generates a random boolean value.

```
#{randomBoolean = crypto.getSecureRandomBoolean()}
```

- **return**:

The random boolean value.

- **since**: 5.4.0.0

5.32 getSecureRandomBytes()

- **Signature**:

```
public byte[] getSecureRandomBytes(int numberOfBytes)
```

Generates a byte array with a number of random bytes in it.

```
#{randomBytes = crypto.getSecureRandomBytes(20)}
```

- **parameter**: numberOfBytes
- **return**:
- **since**: 5.4.0.0

5.33 getSecureRandomDouble()

- **Signature**:

```
public double getSecureRandomDouble()
```

Returns a random double value, generated with the Java "SecureRandom" class.

```
#{randomDouble = crypto.getSecureRandomDouble()}
```

- **return**:

The next random double value between 0.0 and 1.0.

- **since**: 5.4.0.0

5.34 getSecureRandomFloat()

- **Signature**:

```
public float getSecureRandomFloat()
```

Returns a random float value, generated with the Java "SecureRandom" class.

```
#{randomFloat = crypto.getSecureRandomFloat()}
```

- **return**:

The next random float value between 0.0 and 1.0.

- **since**: 5.4.0.0

5.35 getSecureRandomInt()

- **Signature**:

```
public int getSecureRandomInt()
```

Generates a random integer value.

```
#{randomInt = crypto.getSecureRandomInt()}
```

- **return**:

The random integer value.

- **since**: 5.4.0.0

5.36 getSecureRandomLong()

- **Signature**:

```
public long getSecureRandomLong()
```

Generates a random long value.

```
#{randomLong = crypto.getSecureRandomLong()}
```

- **return**:

The random long value.

- **since**: 5.4.0.0

5.37 getSecureRandomNumber()

- **Signature**:

```
public int getSecureRandomNumber(int numberOfDigits)
```

Generate a random integer number with a certain number of digits.

```
#{challenge = crypto.getSecureRandomNumber(6)}
```

- **parameter**: numberOfDigits

The number of digits.

- **return**:

The integer value.

- **since**: 5.4.0.0

5.38 getSecureRandomString()

- **Signature**:

```
public String getSecureRandomString(int numberOfCharacters, boolean ↩
    numbersOnly)
```

Generates a text string containing random ascii characters and / or numeric digits.

Note
The string will never contain lowercase "L" ("l") or uppercase "o" ("O") characters, as they are often easily confused with the number digits for one and zero.

- **parameter**: numberOfCharacters

The number of characters (length) for the resulting string.

- **parameter**: numbersOnly

True if the string should contain number digits only.

- **return**:

The resulting random string.

- **since**: 5.4.0.0

5.39 hexDecode()

- **Signature**:

```
public String hexDecode(String input)
```

Decodes a hex string using UTF-8 byte to character encoding.

- **parameter**: input

A hex string to decode.

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.40 hexDecode()

- **Signature**:

```
public String hexDecode(String input, String charEncoding)
```

Decodes a hex string using the given byte to character encoding.

- **parameter**: input

A hex string to decode. * **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.41 hexDecodeToBytes()

- **Signature**:

```
public byte[] hexDecodeToBytes(String input)
```

Decodes a hex string to a byte array.

- **parameter**: input

A hex string to decode.

- **return**:

A decoded string representation.

- **since**: 5.14.0.0

5.42 hexEncode()

- **Signature**:

```
public String hexEncode(String input)
```

Encodes a string to hex using UTF-8 character to byte encoding.

- **parameter**: input

A string to encode.

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.43 hexEncode()

- **Signature**:

```
public String hexEncode(String input, String charEncoding)
```

Encodes a string to hex using given character to byte encoding.

- **parameter**: input

A string to encode. * **parameter**: charEncoding Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.44 hexEncode()

- **Signature**:

```
public String hexEncode(byte[] input)
```

Encodes a byte array to hex.

- **parameter**: input

A byte array to encode.

- **return**:

The encoded String.

- **since**: 5.14.0.0

5.45 hexToBase32()

- **Signature**:

```
public String hexToBase32(String hex)
```

Converts the given hex data to a base32 string.

- **parameter**: hex

The hex string to convert.

- **return**:

The resulting base32 string.

- **since**: 5.14.0.0

5.46 hexToBase64()

- **Signature**:

```
public String hexToBase64(String hex)
```

Converts the given hex data to a base64 string.

- **parameter**: hex

The hex string to convert.

- **return**:

The resulting base64 string.

- **since**: 5.14.0.0

5.47 hmacSHA256()

- **Signature**:

```
public String hmacSHA256(String key, String data)
```

Returns a hex-encoded HmacSHA256 encrypted hash for the given key and data.

- **parameter**: key

key to use

- **parameter**: data

data to hash and encrypt

- **return**:

hex-encoded hmacSHA256

- **since**: 5.4.0.0

5.48 hmacSHA256bytes()

- **Signature**:

```
public byte[] hmacSHA256bytes(String key, String data)
```

Returns a byte array containing a HmacSHA256 encrypted hash for the given key and data.

- **parameter**: key

key to use

- **parameter**: data

data to hash and encrypt

- **return**:

The byte array with the hmacSHA256 data

- **since**: 5.4.0.0

5.49 isMobileIDSignatureOK()

- **Signature**:

```
public boolean isMobileIDSignatureOK(String signedData, String signature)
```

Verifies the CMS/PKCS7 signature in the response received from the MobileID backend. The truststore used for this verification must either be configured using the properties "mobileid.truststore.path", "mobileid.truststore.pwd" and "mobileid.truststore.type" (defaults to "JKS"), or by setting the Java system properties "javax.net.ssl.trustStore", "javax.net.ssl.trustStorePassword" and "javax.net.ssl.trustStoreType" (defaults to "JKS").

- **parameter**: signedData

The data that the user had to sign.

- **parameter**: signature

The base64 signature string received in the JSON field *MSS_StatusResp.MSS_Signature*. @throws Exception If the signature could not be successfully verified.

- **since**: 5.14.0.0

5.50 sha256()

- **Signature**:

```
public byte[] sha256(byte [] input)
```

This method creates a SHA256 hash from a given byte array.

- **parameter**: input

Any string.

- **return**:

The SHA256 hash bytes.

- **since**: 5.14.0.0

5.51 sha256()

- **Signature**:

```
public String sha256(String input)
```

This method creates a SHA256 hash from a given string (UTF-8 encoded to bytes).

- **parameter**: input

Any string.

- **return**:

The base64 encoded SHA256 hash.

- **since**: 5.4.0.0

5.52 sha256hex()

- **Signature**:

```
public String sha256hex(String input)
```

This method creates a SHA256 hash from a given string (UTF-8 encoded to bytes).

- **parameter**: input

Any string.

- **return**:

The hex encoded SHA512 hash.

- **since**: 5.4.0.0

5.53 sha512()

- **Signature**:

```
public byte[] sha512(byte [] input)
```

This method creates a SHA512 hash from a given byte array.

- **parameter**: input

Any string.

- **return**:

The SHA512 hash bytes.

- **since**: 5.14.0.0

5.54 sha512()

- **Signature**:

```
public String sha512(String input)
```

This method creates a base64 encoded SHA512 hash from a given string (UTF-8 encoded to bytes).

- **parameter**: input

Any string.

- **return**:

The base64 encoded SHA512 hash.

- **since**: 5.4.0.0

5.55 sha512hex()

- **Signature**:

```
public String sha512hex(String input)
```

This method creates a hex-encoded SHA512 hash from a given string (UTF-8 encoded to bytes).

- **parameter**: input

Any string.

- **return**:

The hex encoded SHA512 hash.

- **since**: 5.4.0.0

Chapter 6

function

Provides general utility functions.

6.1 addSig()

- **Signature**:

```
public String addSig(String data)
```

This function adds a signature to the given data string in the correct format to be parsed by the J2EE filter or Tomcat authenticator. <p> IMPORTANT: If the given value should also contain a timestamp that is going to be validated by a servlet filter or the SLS Tomcat authenticator, the JEXL function "addSigAndTime" should be used instead. This way, proper formatting of timestamp and signature can be ensured. <p> NOTE: This function requires two configuration properties: <ol> ses.ticketapi.keyIndexFile=/..path../list.keyindex ses.ticketapi.keyId=key_alias The first defines the path of the SES Ticket API key index file, which contains the actual file path of the signing key referenced by the key <code>key_alias</code>. Please see the "SLS Administration Guide" chapter about "SES Ticket API" and "SSO Integration", specifically the "regex mode example", for more detailled configuration information.

```
app.header.userinfo=${function.addSig('User: ' + parameter.userid)}
```

- **parameter**: data The data to be signed
- **return**:

A new string with the data and the new created signature

- **since**: 4.5.4

6.2 addSigAndTime()

- **Signature**:

```
public String addSigAndTime(String data)
```

This function adds a signature and a timestamp to the given data. <p> NOTE: This function requires two configuration properties: <ol> ses.ticketapi.keyIndexFile=/..path../list.keyindex ses.ticketapi.keyId=key_alias The first defines the path of the SES Ticket API key index file, which contains the actual file path of the signing key referenced by the key <code>key_alias</code>. Please see the "SLS Administration Guide" chapter about "SES Ticket API" and "SSO Integration", specifically the "regex mode example", for more detailled configuration information.

- **parameter**: data

The plaintext data for which to create a signature.

- **return**:

The data with a timestamp and a signature. The timestamp is part of the signed data.

- **since**: 4.5.4

6.3 addTimestamp()

- **Signature**:

```
public String addTimestamp(String data)
```

This function adds a timestamp to the given data string.

- **parameter**: data

The plaintext data for which to create a signature.

- **return**:

the given data with a timestamp appended.

- **since**: 4.5.4

6.4 addTimestamp()

- **Signature**:

```
public String addTimestamp(String data, String timestampFormat, String ↩
    timestampPrefix, String timestampSuffix)
```

This function adds a timestamp to the given data string.

```
${function.addTimestamp(mydata, ';Timestamp=', ';')}
```

- **parameter**: data

The plaintext data for which to create a signature.

- **parameter**: timestampFormat

The Java text format definition of the timestamp.

- **parameter**: timestampPrefix

Defines the prefix (the *key*) of the timestamp entry to be added. If null, none will be added.

- **parameter**: timestampSuffix

Defines the end character for the timestamp entry. If null, none will be added.

- **return**:

the given data with a timestamp appended.

- **since**: 4.9.2

6.5 addToArray()

- **Signature**:

```
public Object[] addToArray(Object[] array, Object toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.3.10

6.6 addToArray()

- **Signature**:

```
public byte[] addToArray(byte[] array, byte toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.7 addToArray()

- **Signature**:

```
public short[] addToArray(short[] array, short toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.8 addToArray()

- **Signature**:

```
public int[] addToArray(int[] array, int toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.9 addToArray()

- **Signature**:

```
public long[] addToArray(long[] array, long toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.10 addToArray()

- **Signature**:

```
public float[] addToArray(float[] array, float toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.11 addToArray()

- **Signature**:

```
public double[] addToArray(double[] array, double toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.12 addToArray()

- **Signature**:

```
public char[] addToArray(char[] array, char toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.13 addToArray()

- **Signature**:

```
public boolean[] addToArray(boolean[] array, boolean toAdd)
```

Adds an object to an array. The return value is a new array containing the additional value as its last entry.

```
${list = function.addToArray(list, someObject)}
```

- **parameter**: array

The array.

- **parameter**: toAdd

The object to add to the array.

- **return**:

The complete array.

- **since**: 4.24.0

6.14 arrayContains()

- **Signature**:

```
public boolean arrayContains(Object obj, Object value)
```

Checks if the given object (usually a string) contains a certain value. The reason why this function is called "arrayContains()", even though it does not take an array as an input parameter, is the following scenario: When, for example, an LDAP attribute variable with user memberships is created as a single string variable instead of an array, just because the user in question is only member of one single group, then the JEXL parser will invoke this function instead of the one with the array parameter. So, in a script or condition, it's all completely transparent. The check still works as expected.

- **parameter**: obj

The object to search. * **parameter**: value The value to look for.

- **return**:

True, if the object contains (equals) the value.

- **since**: 4.30.0

6.15 arrayContains()

- **Signature**:

```
public boolean arrayContains(Object[] array, Object value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.3.6

6.16 arrayContains()

- **Signature**:

```
public boolean arrayContains(byte[] array, byte value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.17 arrayContains()

- **Signature**:

```
public boolean arrayContains(short[] array, short value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.18 arrayContains()

- **Signature**:

```
public boolean arrayContains(int[] array, int value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.19 arrayContains()

- **Signature**:

```
public boolean arrayContains(long[] array, long value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.20 arrayContains()

- **Signature**:

```
public boolean arrayContains(float[] array, float value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.21 arrayContains()

- **Signature**:

```
public boolean arrayContains(double[] array, double value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.22 arrayContains()

- **Signature**:

```
public boolean arrayContains(char[] array, char value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.23 arrayContains()

- **Signature**:

```
public boolean arrayContains(boolean[] array, boolean value)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array The array to search.
- **parameter**: value The value to look for.
- **return**:

True, if the array contains the value.

- **since**: 4.24.0

6.24 arrayMatches()

- **Signature**:

```
public boolean arrayMatches(String holder, String regex)
```

Checks if the given object (usually a string) contains a certain value which matches the given regular expression. The reason why this function is called "arrayContains()", even though it does not take an array as an input parameter, is the following scenario: When, for example, an LDAP attribute variable with user memberships is created as a single string variable instead of an array, just because the user in question is only member of one single group, then the JEXL parser will invoke this function instead of the one with the array parameter. So, in a script or condition, it's all completely transparent. The check still works as expected.

- **parameter**: holder

The value to search.

- **parameter**: regex

The regular expression with which to match the values.

- **return**:

True, if there was a value in the holder which matched the expression.

- **since**: 4.30.0

6.25 arrayMatches()

- **Signature**:

```
public boolean arrayMatches(String[] array, String regex)
```

Checks if the given array contains a certain value. An array is a variable with multiple values, where a single value would be referenced like this: variable[0]

- **parameter**: array

The array of values to search.

- **parameter**: regex

The regular expression with which to match the values.

- **return**:

True, if there was a value in the array which matched the expression.

- **since**: 4.30.0

6.26 base32decode()

- **Signature**:

```
public String base32decode(String input)
```

Decodes a base32 string.

- **parameter**: input A Base32 string to decode.
- **return**:

A decoded string representation. @deprecated Use "crypto.base32decode(String)" instead

- **since**: 4.39.0

6.27 base32encode()

- **Signature**:

```
public String base32encode(String input)
```

Encodes a string to base32.

- **parameter**: input A string to encode.
- **return**:

The encoded String. @deprecated Use "crypto.base32encode(String)" instead

- **since**: 4.39.0

6.28 base32encode()

- **Signature**:

```
public String base32encode(byte[] input)
```

Encodes a byte array to base32.

- **parameter**: input A byte array to encode.
- **return**:

The encoded String. @deprecated Use "crypto.base32encode(byte[])" instead

- **since**: 4.39.0

6.29 base32toHex()

- **Signature**:

```
public String base32toHex(String base32)
```

Converts the given base32 data to a hex string.

- **parameter**: base32

The base32 string to convert.

- **return**:

The resulting hex string. @deprecated Use "crypto.base32toHex(String)" instead

- **since**: 4.39.0

6.30 base64decode()

- **Signature**:

```
public String base64decode(String input)
```

Decodes a base64 string.

- **parameter**: input A Base64 string to decode.
- **return**:

A decoded string representation. @deprecated Use "crypto.base64decode(String)" instead

- **since**: 4.0.12

6.31 base64encode()

- **Signature**:

```
public String base64encode(String input)
```

Encodes a string to base64.

- **parameter**: input A string to encode.
- **return**:

The encoded String. @deprecated Use "crypto.base64encode(String)" instead

- **since**: 4.0.12

6.32 base64encode()

- **Signature**:

```
public String base64encode(byte[] input)
```

Encodes a byte array to base64.

- **parameter**: input A byte array to encode.
- **return**:

The encoded String. @deprecated Use "crypto.base64encode(byte[])" instead

- **since**: 4.0.12

6.33 base64toHex()

- **Signature**:

```
public String base64toHex(String base64)
```

Converts the given base64 data to a hex string. This can be useful with the "md5" or "sha?" functions, which create Base64 data.

- **parameter**: base64

The base64 string to convert.

- **return**:

The resulting hex string. @deprecated Use "crypto.base64toHex(String)" instead

- **since**: 4.11.1

6.34 bytesToString()

- **Signature**:

```
public String bytesToString(byte[] bytes)
```

Convert byte array to string using UTF-8 encoding.

- **parameter**: bytes

Byte array

- **return**:

String

- **since**: 5.14.0.0

6.35 bytesToString()

- **Signature**:

```
public String bytesToString(byte[] bytes, String charEncoding)
```

Convert byte array to string using given encoding.

- **parameter**: bytes

Byte array

- **parameter**: charEncoding

Byte to character encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

String

- **since**: 5.14.0.0

6.36 clearAllAdapterAttributes()

- **Signature**:

```
public void clearAllAdapterAttributes(String adapterType)
```

Clears all attributes collected by an adapter. Since any attribute created by an adapter in any previous back-end call during the same SLS session, this function allows to start from scratch by clearing out all current attributes.

- **parameter**: adapterType

The adapter type, such as "radius" or "ldap" etc.

- **since**: 4.7.3

6.37 clearErrors()

- **Signature**:

```
public void clearErrors()
```

Clears the current error message, so that it won't be displayed in a JSP. NOTE: After invoking this function, the information about the error is lost, so other functions like "getSlsClientErrorMessageKey()" won't work after this function has been invoked.

- **since**: 5.1.0.0

6.38 clearVariable()

- **Signature**:

```
public void clearVariable(String name)
```

Removes a JEXL variable, if it exists. Groovy shortcut: <code>clearVar(name)</code>.

- **parameter**: name

The name of the variable.

- **since**: 4.1.0

6.39 clearVariables()

- **Signature**:

```
public void clearVariables(String prefix)
```

Removes all JEXL variables with a certain name prefix, if they exist. If the prefix matches no names (or is empty), no variables will be deleted. Groovy shortcut: <code>clearVars(prefix)</code>.

- **parameter**: prefix

The prefix to match for the variable names.

- **since**: 5.4.0.0

6.40 contains()

- **Signature**:

```
public boolean contains(Object obj, Object value)
```

Checks if the given objects contains a certain value. If the object is an array of objects, all entries of the array checked if they equal the value. If the object is not an array it is compared for equality. Arrays of primitive types are not supported by this function.

```
...state.50.action.1.if=${function.contains(someList, 'certain value')}
```

- **parameter**: obj The object to search in.
- **parameter**: value The value to look for.
- **return**:

{@code true} if the object contains the value, {@code false} if the value could not be found or an array of primitive types was used.

- **since**: 4.7.0

6.41 createClassInstance()

- **Signature**:

```
public void createClassInstance(String prefix, String className)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details). <p> The message text may contain variables as supported by the Java message those variables in the order of their numbers (first entry in the parameter

- **parameter**: messageId

The ID of the custom log message.

- **parameter**: params

The parameters for the message text.

- **since**: 4.27.0.3

6.42 createCounter()

- **Signature**:

```
public void createCounter(String name)
```

Creates a new counter object as a variable with the given name in the current session.

- **parameter**: name

The name for the counter variable.

- **since**: 4.1.19

6.43 createEncryptedTicket()

- **Signature**:

```
public String createEncryptedTicket(String ticketName, String username) throws ↩
    TicketException
```

Creates an encrypted SES Ticket. The ticket will be encrypted with the key defined by the "encryptionKey"-Property in the SES ticket definition. Example:

<code>ses.ticket.user.realm=acme</code> <code>ses.ticket.user.lifetime=300</code> <code>ses.ticket.user.keyIndexFile=/opt/usp/sls/keys/list.keyindex</code> <code>ses.ticket.user.private.key=key_1</code> <code>ses.ticket.user.public.key=key_2</code> <code>ses.ticket.user.encryption.key=key_3</code>

- **parameter**: ticketName The configuration name of the ticket.
- **parameter**: username The login username.
- **return**:

A string representation of the SES Ticket. @throws TicketException

- **since**: 4.3.9

6.44 createNevisToken()

- **Signature**:

```
public String createNevisToken()
```

Creates an SSO token for the Nevis authentication service. The token is an XML string that can then be used as the content of a cookie or header etc. <p> NOTE 1: This function requires four configuration properties to be set in "sls.properties": <ul> certificate.file.nevisSigner - Path of the certificate file. certificate.key.file.nevisSigner - Path of the private key file. certificate.key.password.nevisSigner - The private key password. Both file paths can be specified as absolute paths, or relative to the SLS "WEB-INF" directory. <p> NOTE 2: The actual contents of the security token have to be defined in the XML include file "NevisAuthTokenAttributes.xml" in the subdirectory "WEB-INF/templates" of the SLS web application.

- **return**:

The Nevis token XML string.

- **since**: 4.8.0

6.45 createObjectInstance()

- **Signature**:

```
public void createObjectInstance(String prefix, String className)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details). <p> The message text may contain variables as supported by the Java message those variables in the order of their numbers (first entry in the parameter

- **parameter**: messageId

The ID of the custom log message.

- **parameter**: params

The parameters for the message text.

- **since**: 4.27.0.3

6.46 createTicket()

- **Signature**:

```
public String createTicket(String ticketName, String username) throws ↩
    TicketException
```

Creates a SES Ticket.

- **parameter**: ticketName The configuration name of the ticket.
- **parameter**: username The login username.
- **return**:

A string representation of the SES Ticket. @throws TicketException

- **since**: 4.0.12

6.47 currentDate()

- **Signature**:

```
public Date currentDate()
```

Returns the current time as a <code>java.util.Date</code> object.

```
${function.currentDate().after(otherDate)}
```

- **return**:

The current time as a Date object.

- **since**: 4.9.2
- **since**: use 1.0.5

6.48 currentSeconds()

- **Signature**:

```
public long currentSeconds()
```

Returns the current time as number of seconds since midnight, January 1, 1970 UTC. The result is a numerical value (not a text string), so it can be used for calculation purposes.

- **return**:

The number of seconds.

- **since**: 4.7.8

6.49 dataProtectorDecrypt()

- **Signature**:

```
public String dataProtectorDecrypt(String encrypted, String encoding)
```

Uses the DataProtector/Seal to decrypt a given encrypted text. Note that the result of the decryption depends on the key in the DataProtector/Seal keystore file, as configured in the "sls.properties" configuration file (see property "dataprotector.keystore").

```
${plain = function.dataProtectorDecrypt(encryptedData, 'UTF-8')}
```

- **parameter**: encrypted

The encrypted text.

- **parameter**: encoding

The encoding of the original string (defaults to "UTF-8", if the parameter is *null*).

- **return**:

A string with the decrypted plain text.

- **since**: 4.3.10

6.50 dataProtectorEncrypt()

- **Signature**:

```
public String dataProtectorEncrypt(String plain, String encoding)
```

Uses the DataProtector/Seal to encrypt a given plain text. Note that the result of the encryption depends on the key in the DataProtector/Seal keystore file, as configured in the "sls.properties" configuration file (see property "dataprotector.keystore").

```
${encrypted = function.dataProtectorEncrypt('some text', 'UTF-8')}
```

- **parameter**: plain

The plain text to encrypt.

- **parameter**: encoding

The encoding of the string (defaults to "UTF-8", if the parameter is *null*).

- **return**:

A base64 string with the encrypted text.

- **since**: 4.3.10

6.51 date()

- **Signature**:

```
public String date(String format)
```

This method returns the current date/time formatted in the given pattern.

- **parameter**: format The pattern to format the current date/time. For example: "yyMMddHHmmss"
- **return**:

A string with the current date/time, e.g. "150328140123". @see java.text.SimpleDateFormat

- **since**: 4.0.12

6.52 dateRfc822()

- **Signature**:

```
public String dateRfc822()
```

This method returns the current date/time formatted in RFC 822 format, as needed e.g. for HTTP date headers, e.g. "Wed, 02 Oct 2002 15:00:00 +0200".

- **return**:

A string with the current date/time. @see java.text.SimpleDateFormat

- **since**: 4.0.12

6.53 deleteOpticalTokenImage()

- **Signature**:

```
public void deleteOpticalTokenImage()
```

Deletes the local token image file, created by the JEXL function "storeOpticalTokenImage()". Invoke this function before completing the login procedure, e.g. in a JEXL action during a "do.generic" state just before the "do.success" state.

- **since**: 4.34.0.1

6.54 enableSesTraceLog()

- **Signature**:

```
public void enableSesTraceLog()
```

It sets a header to indicate to the SES that the trace log has to be dynamically activated, for the defined users for that trigger.

- **since**: 4.8.0

6.55 evalExpr()

- **Signature**:

```
public String evalExpr(String expr, String doEncode)
```

Evaluates a given string, and any JEXL expression to be found in it. This function can be useful if values are to be used in the config that may contain variables, but are coming from a source that is not evaluated automatically. A typical example is an adapter response attribute value;

6.56 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithAuthenticationError('INVALID_NEW_PWD_SAME_AS_OLD')}
```

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **since**: 5.10.0.0

6.57 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithAuthenticationError('ERR_INTERNAL_STATE','Fail!')}
```

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1

- **since**: 4.1.28

6.58 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1

- **parameter**: arg2

- **since**: 4.1.28

6.59 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2, String arg3)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1
- **parameter**: arg2
- **parameter**: arg3
- **since**: 4.1.28

6.60 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2, String arg3, String arg4)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file.

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1
- **parameter**: arg2
- **parameter**: arg3
- **parameter**: arg4
- **since**: 4.1.28

6.61 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2, String arg3, String arg4, String arg5)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1
- **parameter**: arg2
- **parameter**: arg3
- **parameter**: arg4
- **parameter**: arg5
- **since**: 4.1.28

6.62 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2, String arg3, String arg4, String arg5, String arg6)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1

- **parameter**: arg2
- **parameter**: arg3
- **parameter**: arg4
- **parameter**: arg5
- **parameter**: arg6
- **since**: 4.1.28

6.63 failWithAuthenticationError()

- **Signature**:

```
public void failWithAuthenticationError(String messageId, String arg1, String ↩
    arg2, String arg3, String arg4, String arg5, String arg6, String arg7)
```

Creates an authentication error. As a result, the model will switch to the appropriate failed state, and an error message will be displayed in the current page. The resulting message depends on the configured mapping of the given message ID in the "sls-errormap.properties" file. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

- **parameter**: messageId

The message ID of the error (see "sls-log-messages.pdf" for a complete list of error IDs).

- **parameter**: arg1
- **parameter**: arg2
- **parameter**: arg3
- **parameter**: arg4
- **parameter**: arg5
- **parameter**: arg6
- **parameter**: arg7
- **since**: 4.1.28

6.64 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog)
```

Creates an authentication error with the internal SLS error ID <code>CUSTOM_JEXL_ERROR</code>. The first parameter allows to define the text that will be displayed to the user (instead of using the error-mapping mechanism through the "sls-errormap.properties" file). It must be a valid key of a text in the message resource file(s). <p> NOTE: The message text in the resource file should always begin the internal error ID (<code>CUSTOM_JEXL_ERROR</code>) will not be displayed on the JSP if <code>error.details</code> has been set to <code>true</code> in the configuration. Example: <p> <pre> </pre> <p> The second parameter is only written to the SLS log file as the error reason. <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('custom.error.text', 'Invalid user!')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **since**: 4.5.0

6.65 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1)
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('custom.error.text', 'Invalid user!', 'fox')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **since**: 4.30.0

6.66 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1,
String messageArg2)
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('err.txt','Fail!','A','B')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **parameter**: messageArg2

in the message displayed to the user.

- **since**: 4.30.0

6.67 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1,
String messageArg2, String messageArg3)
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('err.txt','Fail!','A','B','C')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **parameter**: messageArg2

in the message displayed to the user.

- **parameter**: messageArg3

in the message displayed to the user.

- **since**: 4.30.0

6.68 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1,
String messageArg2, String messageArg3, String messageArg4)
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('err.txt','Fail!','A','B','C','D')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **parameter**: messageArg2

in the message displayed to the user.

- **parameter**: messageArg3

in the message displayed to the user.

- **parameter**: messageArg4

in the message displayed to the user.

- **since**: 4.30.0

6.69 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1,
String messageArg2, String messageArg3, String messageArg4, String messageArg5 ↩
    )
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('err.txt','Fail!','A','B','C','D','E')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **parameter**: messageArg2

in the message displayed to the user.

- **parameter**: messageArg3

in the message displayed to the user.

- **parameter**: messageArg4

in the message displayed to the user.

- **parameter**: messageArg5

in the message displayed to the user.

- **since**: 4.30.0

6.70 failWithCustomError()

- **Signature**:

```
public void failWithCustomError(String messageId, String reasonToLog, String ↩
    messageArg1,
String messageArg2, String messageArg3, String messageArg4, String messageArg5 ↩
    ,
String messageArg6)
```

Same function as "failWithCustomError(String, String)", but allows to define additional variable values for the text message. Example: <p> <pre> </pre> <p> NOTE: When this function is used in the "action" of a JSP state like "get.cred", it will (a) be executed only AFTER the JSP has been displayed, and once the model advances to the next state, and (b) will result in a 500 error response from the SLS, because the internal error handling is different for the JSP states than for other non-JSP states like "do.auth".

```
${function.failWithCustomError('err.txt','Fail!','A','B','C','D','E','F')}
```

- **parameter**: messageId

Key of a text entry in the message resource file(s) to display to the user.

- **parameter**: reasonToLog

Information about what caused the error (used only for the log record).

- **parameter**: messageArg1

in the message displayed to the user.

- **parameter**: messageArg2

in the message displayed to the user.

- **parameter**: messageArg3

in the message displayed to the user.

- **parameter**: messageArg4

in the message displayed to the user.

- **parameter**: messageArg5

in the message displayed to the user.

- **parameter**: messageArg6

in the message displayed to the user.

- **since**: 4.30.0

6.71 formatCurrentDate()

- **Signature**:

```
public String formatCurrentDate(String format)
```

Creates a formatted string for the current date, so this is essentially a timestamp function.

- **parameter**: format

The string that describes the format of the resulting string.

- **return**:

The date string or an empty string, if there was a formatting problem.

- **since**: 4.1.29

6.72 formatCurrentDate()

- **Signature**:

```
public String formatCurrentDate()
```

Creates a formatted string for the current date, so this is essentially a timestamp function.

- **return**:

The date string or an empty string, if there was a formatting problem.

- **since**: 4.1.0

6.73 formatDate()

- **Signature**:

```
public String formatDate(Date date, String format)
```

Creates a formatted string for a given Date object.

- **parameter**: date

The date object to create a string for.

- **parameter**: format

The string that describes the format of the resulting string.

- **return**:

The date string or an empty string, if there was a formatting problem.

- **since**: 4.1.29

6.74 formatDate()

- **Signature**:

```
public String formatDate(Date date)
```

Creates a formatted string for a given Date object.

- **parameter**: date

The date object to create a string for.

- **return**:

The date string or an empty string, if there was a formatting problem.

- **since**: 4.1.0

6.75 generateChallenge()

- **Signature**:

```
public String generateChallenge()
```

Generates a cryptographically strong random challenge (based on java.security.SecureRandom) of 10 characters length, where each character comes from a set of characters that contains all upper and lower case letters and all numbers, except "1Il0O", i.e. without the number one, the upper case letter "i", the lower case letter "L", the number zero, and the upper case letter "o".

- **return**:

The generated challenge.

- **since**: 5.8.0.0

6.76 generateChallenge()

- **Signature**:

```
public String generateChallenge(int len)
```

Generates a cryptographically strong random challenge (based on java.security.SecureRandom) of given length, where each character comes from a set of characters that contains all upper and lower case letters and all numbers, except "1Il0O", i.e. without the number one, the upper case letter "i", the lower case letter "L", the number zero, and the upper case letter "o".

- **parameter**: len

The desired length of the challenge.

- **return**:

The generated challenge.

- **since**: 5.8.0.0

6.77 generateChallenge()

- **Signature**:

```
public String generateChallenge(int len, String chars)
```

Generates a cryptographically strong random challenge (based on java.security.SecureRandom) of given length, where each character comes from the given set of characters.

- **parameter**: len

The desired length of the challenge.

- **parameter**: chars

The desired characters in the challenge.

- **return**:

The generated challenge.

- **since**: 5.8.0.0

6.78 generateRandomPassword()

- **Signature**:

```
public String generateRandomPassword() throws ConfigurationException
```

Generates a random password according to the rules defined in the "password-policy.xml" file.

- **return**:

The generated random password. @throws ConfigurationException If there was a problem with the password policy configuration.

- **since**: 4.1.24

6.79 getAdapterAttribute()

- **Signature**:

```
public String getAdapterAttribute(String adapterType, String attributeName)
```

Returns the value of a certain attribute of the adapter (if that attribute exists, after the last call-out).

- **parameter**: adapterType

The adapter type, such as "radius" or "ldap" etc.

- **parameter**: attributeName

The name of the attribute, without any prefixes, like "mailAddress" (or, for RADIUS attributes, a number like "18").

- **return**:

The value, or an empty string (but never null).

- **since**: 4.0.18

6.80 getAdapterType()

- **Signature**:

```
public String getAdapterType(String function)
```

Allows to check which adapter type was used for a certain function. This corresponds to the configuration settings in the "sls.properties" file. For example, if the following configuration property is set: <pre> adapter.authentication = ldap </pre> then the following function call will return "ldap": <pre> function.getAdapterType(*authentication*);

6.81 getArraySize()

- **Signature**:

```
public int getArraySize(Object array)
```

Allows to determine the size of an array, for example in a condition.

```
${function.getArraySize(myArray, -1) > 3}
```

- **parameter**: array

The array.

- **return**:

The number of entries in the array, or -1 if the size of the array could not be determined.

- **since**: 4.11.3

6.82 getArraySize()

- **Signature**:

```
public int getArraySize(Object array, int defaultSize)
```

Allows to determine the size of an array, for example in a condition.

```
${function.getArraySize(myArray, -1) > 3}
```

- **parameter**: array

The array.

- **parameter**: defaultSize

If the array size cannot be determined for some reason, the value of this parameter is returned (for example, if the given array is null).

- **return**:

The number of entries in the array, or the given default size, if the size of the array could not be determined.

- **since**: 4.8.6
- **since**: use 1.0.4

6.83 getAuthorizations()

- **Signature**:

```
public String getAuthorizations(String separator)
```

DEPRECATED: Use "session.getAuthorizations(String)" instead.

@deprecated Use {@link Session#getAuthorizations(String)} function instead. * **parameter**: separator The string separating the different authorisations.

- **return**: A String containing all authorisations.
- **since**: 4.6.0

6.84 getBasicAuthPassword()

- **Signature**:

```
public String getBasicAuthPassword()
```

Get basic auth password if a basic auth header has been received in the current request.

- **return**:

password or null if could not obtain for any reason

- **since**: 5.18.0.0

6.85 getBasicAuthUsename()

- **Signature**:

```
public String getBasicAuthUsename()
```

Get basic auth username if a basic auth header has been received in the current request.

- **return**:

username or null if could not obtain for any reason

- **since**: 5.18.0.0

6.86 getCertificate()

- **Signature**:

```
public AbstractCertificateHolder getCertificate(String alias)
```

Returns a holder (wrapper) for the given certificate. The certificate is defined by the alias, which refers to a configuration property "certificate.file.alias=filename". <p> The "CertificateHolder" object returned by this function provides the following methods that can be invoked on it: <ul> getCertificate() - Returns a <code>java.security.cert.Certificate</code> instance of the X509 certificate. getFingerPrint(hashType) - Returns the fingerprint string for the given hash type (such as *MD5* or *SHA1*). Please refer to the "SLS Administration Guide" for detailled explanations of the corresponding configuration properties.

```
${function.getCertificate('mycert').getCertificate().toString()}
```

- **parameter**: alias

The alias that refers to a configuration property with the filename.

- **return**:

The certificate holder instance.

- **since**: 4.6.0

6.87 getCertificateFingerprint()

- **Signature**:

```
public String getCertificateFingerprint(String alias, String hashType)
```

Returns a fingerprint string for the given certificate. The certificate itself is defined by the alias (the first parameter), which refers to a configuration property "certificate.file.<alias>=<filename>". <p> Please refer to the "SLS Administration Guide" for detailled explanations of the corresponding configuration properties.

```
${function.getCertificateFingerprint('mycert', 'sha1')}
```

- **parameter**: alias The alias that refers to a configuration property with the filename.
- **parameter**: hashType The hash type ("MD5" or "SHA1").
- **return**:

The fingerprint string, like "A1:A4:9F:07:B5:CC:C3:B7:1B:78:17:53:CE:94:AA:22"

- **since**: 4.6.0

6.88 getClientCertificateExtension()

- **Signature**:

```
public String getClientCertificateExtension(String extensionOID)
```

Returns the string value of an extension of the current client certificate (in a PKI adapter login setup).

- **parameter**: extensionOID

The OID of the extension.

- **return**:

The string value, or null, if the certificate did not have such an extension, or no certificate existed in the session, or the given

- **since**: 4.30.0

6.89 getCompleteUrl()

- **Signature**:

```
public String getCompleteUrl(String url) throws SLSException
```

Creates a finalized, correct, safe redirect URL based on the incoming URL which may be relative or contain invalid host/port information.

- **parameter**: url The incoming URL to be completed.
- **return**:

The complete and correct URL. @throws SLSException

- **since**: 4.1.28

6.90 getConfigPropertiesWithPrefix()

- **Signature**:

```
public List<String> getConfigPropertiesWithPrefix(String prefix)
```

Returns the values of all configuration properties with the given prefix. NOTE: If the properties use a numbering scheme in their name, i.e. "my.prop.1", "my.prop.2" etc. for a prefix "my.prop", the resulting list will use an ascending order based on the numbers. So the first entry in the list would be the value of "my.prop.1", then the value of "my.prop.2" and so on.

- **parameter**: prefix

The configuration property name prefix to look for.

- **return**:

A list of strings with the property values (may be empty, but not null).

- **since**: 4.40.0.0

6.91 getConfigProperty()

- **Signature**:

```
public Object getConfigProperty(String name)
```

Returns an SLS configuration property value.

- **parameter**: name

The name of the configuration propery.

- **return**:

The property value (may be an empty string if no value was found).

- **since**: 4.29.0

6.92 getConfigProperty()

- **Signature**:

```
public Object getConfigProperty(String name, String defaultValue)
```

Returns an SLS configuration property value.

- **parameter**: name

The name of the configuration propery.

- **parameter**: defaultValue

The default value of the configuration property if not configured.

- **return**:

The property value (or the given default value if no value was found).

- **since**: 4.29.0

6.93 getConfigPropertyEntriesWithPrefix()

- **Signature**:

```
public List<ConfigEntry> getConfigPropertyEntriesWithPrefix(String prefix,  ↩
    boolean evaluate)
```

Returns property entries for all configuration properties with the given prefix. NOTE: If the properties use a numbering scheme in their name, i.e. "my.prop.1", "my.prop.2" etc. for a prefix "my.prop", the resulting list will use an ascending order based on the numbers. So the first entry in the list would be the value of "my.prop.1", then the value of "my.prop.2" and so on.

<p> The property entries have the following getters/attributes: <ul> getSuffix() - Returns the suffix of this configuration property. This is the part of the property name after the prefix, usually some kind of grouping id, a number etc. getPrefix() - Returns the prefix of this configuration entry. This is the prefix that was used for the search for entries. getKey() - Returns the key (property) name of this configuration entry. getValue() - Returns the value of this configuration entry. getOriginalSuffix() - Like getSuffix(), but returns the original, non-lowercased suffix part of the property name. <p> Note that in Groovy you can write e.g. "{entry.key}" instead of "{entry.getKey()}".

- **parameter**: prefix

The configuration property name prefix to look for.

- **parameter**: evaluate

Whether to evaluate Groovy/JEXL expressions in property values.

- **return**:

A list of property entries (may be empty, but not null).

- **since**: 4.40.0.0

6.94 getConfigPropertyNamesWithPrefix()

- **Signature**:

```
public List<String> getConfigPropertyNamesWithPrefix(String prefix)
```

Returns the property names of all configuration properties with the given prefix. NOTE: If the properties use a numbering scheme in their name, i.e. "my.prop.1", "my.prop.2" etc. for a prefix "my.prop", the resulting list will use an ascending order based on the numbers. So the first entry in the list would be the value of "my.prop.1", then the value of "my.prop.2" and so on.

- **parameter**: prefix

The configuration property name prefix to look for.

- **return**:

A list of strings with the property names (may be empty, but not null).

- **since**: 4.40.0.0

6.95 getCookieValue()

- **Signature**:

```
public String getCookieValue(String cookieName)
```

Returns the value of the specified cookie, if it exists.

```
${cookieValue = function.getCookieValue('MyCookie')}
```

- **parameter**: cookieName

The name of the cookie.

- **return**:

The value (may be an empty string).

- **since**: 4.3.9

6.96 getCred()

- **Signature**:

```
public String getCred(String name)
```

DEPRECATED: Use "session.getCred(String)" instead.

@deprecated Use {@link Session#getCred(String)} function instead. * **parameter**: name The name of the credential.

- **return**:

The value of the credential or {@code null} if it doesn't exist.

- **since**: 4.6.0

6.97 getCurrentRequest()

- **Signature**:

```
public HttpServletRequest getCurrentRequest()
```

Allows to access the <code>javax.servlet.HttpServletRequest</code> object of the current request.

Note that starting with SLS 5.10.0.0, a script variable named <code>request.raw</code> is created which already contains this request, and that for many properties there are specific variables, e.g. instead of <code>function.getCurrentRequest.getRequestURI()</code> simply use the variable <code>request.uri</code>. See theSLS Administration Guide* in the chapter "JEXL Expressions" for a detailed description of all request-specific variables.

- **return**:

The request reference.

- **since**: 4.9.0

6.98 getCurrentYear()

- **Signature**:

```
public Integer getCurrentYear()
```

This method returns the current year.

- **return**:

An integer with the current year

- **since**: 4.1.17

6.99 getGroovyConfig()

- **Signature**:

```
public ConfigObject getGroovyConfig(String configFilePath)
```

Gets a Groovy ConfigObject from a Groovy config file, as it would be obtained by Groovy's ConfigSlurper, but caches previous results, with automatic update if file content has changed since the last call.

This performs better than using ConfigSlurper because the latter always compiles the configuration to a class; and in the past there were also memory leaks with some combinations of Java VM and Groovy version, so using this function is more safe.

- **parameter**: configFilePath

Absolute path or path relative to the SLS webapp of the Groovy config file.

- **return**:

A ConfigObject, the same type of object that GroovySlurper.parse() returns.

- **since**: 5.1.0.0

6.100 getLocalizedMessage()

- **Signature**:

```
public String getLocalizedMessage(String key, String defaultValue)
```

Returns a message string from the language resource files of the SLS web application (the <code>sls-resources_XX.properties</code> file(s) in the <code>WEB-INF/classes</code> subdirectory). <p> The localization depends on the language (and tenant) active in the current session.

- **parameter**: key

The message resource key (property name).

- **parameter**: defaultValue

The default value to return, if no message was found for the given key.

- **return**:

The localized message string or null, if no string was found.

- **since**: 4.40.0.0

6.101 getLocalizedMessage()

- **Signature**:

```
public String getLocalizedMessage(String key)
```

Returns a message string from the language resource files of the SLS web application (the <code>sls-resources_XX.properties</code> file(s) in the <code>WEB-INF/classes</code> subdirectory). <p> The localization depends on the language (and tenant) active in the current session.

- **parameter**: key

The message resource key (property name).

- **return**:

The localized message string or null, if no string was found.

- **since**: 4.1.23

6.102 getLogMessageParameterValue()

- **Signature**:

```
public Object getLogMessageParameterValue(String name)
```

Returns the value of a certain parameter of the current SLS log message. * This function is usually meant to be used when custom log variables are defined using the properties with the prefix "log.variables.*", and the value should access one of the parameters of that message.

- **parameter**: name

The name of the message parameter.

- **return**:

The String value (may be empty string, but not null).

- **since**: 5.1.0.0

6.103 getModelState()

- **Signature**:

```
public String getModelState()
```

DEPRECATED: Use "session.getModelState()" instead.

@deprecated Use {@link Session#getModelState()} function instead.

- **return**:

The current state of the SLS model (corresponds to the state names as defined in the "sls.properties" file).

- **since**: 4.6.0

6.104 getOriginalRequestMethod()

- **Signature**:

```
public String getOriginalRequestMethod()
```

Gets the original HTTP request method with which the request came into the SLS. The original method is formally overwritten in the SLS because since Tomcat 8 JSPs only allow the officially supported methods for JSPs, namely GET, HEAD and POST.

This function allows basic method-specific handling of other HTTP methods like PUT or DELETE, etc. in login models and JSPs.

- **return**:

The original method

- **since**: 5.1.0.0

6.105 getPasswordPolicy()

- **Signature**:

```
public ScriptingPasswordPolicy getPasswordPolicy()
```

Returns an SLS password policy holder.

- **return**:

An instance of a password policy holder. See function list of prefix "passwordpolicy" for information about available methods on the object.

- **since**: 4.27.0.3

6.106 getPreferenceValue()

- **Signature**:

```
public String getPreferenceValue(String key)
```

Returns the value of a preference key, typically something defined by the user (such as his preferred GUI layout or language etc.). <p> This function is most useful when used with the SLS JSP tag "getJexl", in order to display something according to the user's preferences in an HTML page.

- **parameter**: key

The name of the preference value.

- **return**:

The value as a string (may be an empty string but not null).

- **since**: 4.0.17

6.107 getQRCode()

- **Signature**:

```
public String getQRCode(String data, int sizeInPixels)
```

Renders the given string to QR Code PNG image data, base64-encoded.

Can be e.g. be included in HTML pages as an inline image like this:

The generated image is a PNG with a size of sizeInPixels x sizeInPixels pixels, which limits the length of the data that can be successfully parsed from the generated PNG.

- **parameter**: data

String to render to QR Code

- **parameter**: sizeInPixels

length of the QR Code image square in pixels

- **return**:

Base64-encoded QR Code for the given data string

- **since**: 5.16.0.1

6.108 getQRCode()

- **Signature**:

```
public String getQRCode(String data)
```

Renders the given string to QR Code PNG image data, base64-encoded.

Can be e.g. be included in HTML pages as an inline image like this:

The generated image is a PNG with a size of 125 x 125 pixels, which limits the length of the data that can be successfully parsed from the generated PNG.

- **parameter**: data

String to render to QR Code

- **return**:

Base64-encoded QR Code for the given data string

- **since**: 5.16.0.1

6.109 getReqParameterValue()

- **Signature**:

```
public String getReqParameterValue(String key)
```

Returns the value of a field of the *req* parameter. This will only return a value if the current request had such a *req* parameter.

- **parameter**: key

The name of the field.

- **return**:

The value, or an empty string.

- **since**: 4.0.17

6.110 getSlsClientErrorMessage()

- **Signature**:

```
public String getSlsClientErrorMessage()
```

Returns the error message of the current error in the session. This is the localized error message that would be displayed in the JSP page using the "global.error.message" key.

- **return**:

The localized client-side error message.

- **since**: 4.15.0

6.111 getSlsClientErrorMessageKey()

- **Signature**:

```
public String getSlsClientErrorMessageKey()
```

Returns the message resource key of the error message of the current error in the session. This is the localized error message that would be displayed in the JSP page using the "global.error.message" key.

- **return**:

The message resource key for the localized client-side error message.

- **since**: 4.15.0

6.112 getSlsErrorCategory()

- **Signature**:

```
public String getSlsErrorCategory()
```

Returns the category of the current error in the session (if there is one). This can be used in conditions in the model to switch to a certain failedState based on the type of error. <p> See "sls-log-messages.pdf" for a complete list of message IDs and the category of each message. <p> There are three categories: "AUDIT" for audit messages, "SYSTEM" for (usually technical) errors in the SLS, "SYSTEM_BACKEND" for technical errors in the backend server system, and "USER" for user-related errors, such as invalid credentials.

```
${function.getSlsErrorCategory() eq 'SYSTEM')}
```

- **return**:

The message category or an empty string (''), if there was no error.

- **since**: 4.26.0.2

6.113 getSlsErrorId()

- **Signature**:

```
public String getSlsErrorId()
```

Returns the SLS message ID of the current error in the session (if there is one). This can be used in conditions in the model to switch to a certain failedState based on the type of error. <p> See "sls-log-messages.pdf" for a complete list of message IDs.

```
${function.getSlsErrorId() eq 'ERR_INTERNAL_STATE')}
```

- **return**:

The message ID or an empty string (''), if there was no error.

- **since**: 4.1.30

6.114 getSlsErrorMessage()

- **Signature**:

```
public String getSlsErrorMessage()
```

Returns the exception message of the current error in the session (if there is one). This can be used in conditions in the model to switch to a certain failedState based on the type of error.

```
${function.printToConsole(function.getSlsErrorMessage())}
```

- **return**:

The message text or an empty string (''), if there was no error.

- **since**: 4.1.30

6.115 getStorageValue()

- **Signature**:

```
public Object getStorageValue(String name)
```

Returns an object from the thread local storage by its name.

- **parameter**: name

The name of the storage value.

- **return**:

The Object value (may be null).

- **since**: 4.7.8

6.116 getSystemProperty()

- **Signature**:

```
public Object getSystemProperty(String name)
```

Returns a Java system property by its name.

- **parameter**: name

The name of the system property.

- **return**:

The String value (may be null).

- **since**: 4.7.8

6.117 getTemplateContent()

- **Signature**:

```
public String getTemplateContent(String alias)
```

Evaluates the content of a file translating all JEXL expressions inside to their corresponding value. The location of the file is defined by a configured property: template.file.<alias>=<filepath>. The file path is always relative to WEB-INF/templates.

@see Template#getContent(String) * **parameter**: alias The alias of the property defining the location of the file.

- **return**:

The translated content of the file.

- **since**: 4.7.0

6.118 getTemplateContent()

- **Signature**:

```
public String getTemplateContent(String alias, String newline)
```

Evaluates the content of a file translating all JEXL expressions inside to their corresponding value. The location of the file is defined by a configured property: template.file.<alias>=<filepath>. The file path is always relative to WEB-INF/templates.

@see Template#getContent(String) * **parameter**: alias The alias of the property defining the location of the file. * **parameter**: newline What kind of line separator character should be used (unix, windows, system, none).

- **return**:

The translated content of the file.

- **since**: 4.7.0

6.119 getValueFromFile()

- **Signature**:

```
public String getValueFromFile(String filename)
```

Returns the first non-comment line of the given file as a string. Comment lines are lines where the first non-whitespace character is #. Useful, for example, for reading property values from a file (e.g. passwords).

```
${function.getValueFromFile('/var/data/passwordfile.txt')}
```

- **parameter**: filename

path and name of the the file

- **return**:

line, empty string if file is empty except for comments, never null

- **since**: 4.12.1

6.120 getVariable()

- **Signature**:

```
public Object getVariable(String name)
```

Returns a named JEXL variable from the session. This is useful for variables whose name ends with "." followed by a number, because the regular JEXL expression resolver has problems with this kind of variables (they are treated as array accesses internally). Groovy shortcut: <code>var(name)</code>.

- **parameter**: name

The name of the JEXL variable.

- **return**:

The String value (may be null or empty).

- **since**: 4.7.0

6.121 getVariable()

- **Signature**:

```
public Object getVariable(String name, String defaultValue)
```

Returns a named JEXL variable from the session. This is useful for variables whose name ends with "." followed by a number, because the regular JEXL expression resolver has problems with this kind of variables (they are treated as array accesses internally). Groovy shortcut: <code>var(name, defaultValue)</code>.

- **parameter**: name

The name of the JEXL variable.

- **parameter**: defaultValue

The default value to use if the variable does not exist. NOTE: If the variable exists, it's value will be returned, even if it is an empty string, or a null reference.

- **return**:

The String value (may be null or empty).

- **since**: 5.1.0.0

6.122 getVariableNames()

- **Signature**:

```
public List<String> getVariableNames()
```

Returns a sorted list of the names of all JEXL variables. Groovy shortcut: <code>getVarNames()</code>.

- **return**:

Sorted list of the names of all JEXL variables, never null.

- **since**: 5.16.0.0

6.123 getVerifiedCred()

- **Signature**:

```
public String getVerifiedCred(String name)
```

DEPRECATED: Use "session.getVerifiedCred(String)" instead.

@deprecated Use {@link Session#getVerifiedCred(String)} function instead. @deprecated Use "Session" function instead. * **parameter**: name The name of the credential.

- **return**:

The value of the credential or null if it doesn't exist.

- **since**: 4.6.0

6.124 hasAdapterAttribute()

- **Signature**:

```
public boolean hasAdapterAttribute(String adapterType, String attributeName)
```

Checks if the adapter has a certain attribute available after the last call-out.

- **parameter**: adapterType

The adapter type, such as "radius" or "ldap" etc.

- **parameter**: attributeName

The name of the attribute, without any prefixes, like "mailAddress" (or, for RADIUS attributes, a number like "18").

- **return**:

True if such an attribute exists, false if not.

- **since**: 4.0.18

6.125 hasConfigProperty()

- **Signature**:

```
public boolean hasConfigProperty(String name)
```

Allows to check if a certain configuration property exists (in any of the property files in the SLS "WEB-INF" directory).

- **parameter**: name

The name of the configuration property.

- **return**:

True if the property exists, false if not.

- **since**: 4.40.0.0

6.126 hasNonEmptyVariable()

- **Signature**:

```
public boolean hasNonEmptyVariable(String name)
```

Checks if a certain JEXL/Groovy variable exists and has a value that is neither null nor (after trimming) an empty string. Groovy shortcut: <code>hasNonEmptyVar(name)</code>.

- **parameter**: name

The name of the variable.

- **return**:

"true" if such a variable exists and is neither null nor (after trimming) an empty string, "false" otherwise.

- **since**: 4.6.0

6.127 hasVariable()

- **Signature**:

```
public boolean hasVariable(String name)
```

Checks if a certain JEXL variable exists. Groovy shortcut: <code>hasVar(name)</code>.

- **parameter**: name

The name of the variable.

- **return**:

"true" if such a variable exists, "false" if not.

- **since**: 4.0.18

6.128 hmacSHA256()

- **Signature**:

```
public String hmacSHA256(String key, String data)
```

Returns a hex-encoded HmacSHA256 encrypted hash for the given key and data.

- **parameter**: key

key to use

- **parameter**: data

data to hash and encrypt

- **return**:

hex-encoded hmacSHA256 @deprecated Use "crypto.hmacSHA256(String, String)" instead.

- **since**: 4.39.0.0

6.129 hmacSHA256bytes()

- **Signature**:

```
public byte[] hmacSHA256bytes(String key, String data)
```

Returns a byte array containing a HmacSHA256 encrypted hash for the given key and data.

- **parameter**: key

key to use

- **parameter**: data

data to hash and encrypt

- **return**:

The byte array with the hmacSHA256 data @deprecated Use "crypto.hmacSHA256bytes(String, String)" instead.

- **since**: 4.39.0.2

6.130 htmlDecode()

- **Signature**:

```
public String htmlDecode(String input)
```

Decodes a HTML-encoded string.

- **parameter**: input A HTML-string to decode.
- **return**:

A decoded string representation.

- **since**: 4.0.18

6.131 htmlEncode()

- **Signature**:

```
public String htmlEncode(String input)
```

Encodes a string to HTML-encoding.

- **parameter**: input A string to encode.
- **return**:

The encoded String.

- **since**: 4.0.18

6.132 http64decode()

- **Signature**:

```
public String http64decode(String input)
```

Decodes a http64 string.

- **parameter**: input A httpBase64 string to decode.
- **return**:

A decoded string representation.

- **since**: 4.0.13

6.133 http64encode()

- **Signature**:

```
public String http64encode(String input)
```

Encodes a string to http64.

- **parameter**: input A string to encode.
- **return**:

The encoded string.

- **since**: 4.0.13

6.134 isArray()

- **Signature**:

```
public boolean isArray(Object variable)
```

Allows to check if the given variable is an array.

```
if(function.isArray(myVars)) { foreach(myVar in myVars) { ... } }
```

- **parameter**: variable

The JEXL variable to check.

- **return**:

True if it's an array.

- **since**: 4.12.1

6.135 isBrowserOnBlacklist()

- **Signature**:

```
public boolean isBrowserOnBlacklist()
```

Checks if the browser client of the current user is blacklisted, based on the "User-Agent" header.

- **return**:

True if the browser is blacklisted, false if not.

- **since**: 4.0.17

6.136 isClientIpTrusted()

- **Signature**:

```
public boolean isClientIpTrusted(List<String> trustedNetworks)
```

Determines if the IP of the client is in any of the given trusted networks.

Supports IPv4 IPs embedded into IPv6 addresses for the client IP, more precisely "compat", "6to4" and "terero" formats, but not "ISATAP" as that is easy to spoof; see the JavaDoc of Guava's InetAddresses class for more details. Networks are always considered either IPv4 or IPv6, never both; for embedded IPv4 networks test against the network in both IPv4 and IPv6 formats.

- **parameter**: trustedNetworks

List of trusted networks (IPv4 or IPv6), each e.g. "10.21.228.0/22" or "fd99:0:0:3::10/64".

- **return**:

True of the client IP could be determined as an IPv4 address and the IP is in any of the trusted networks, false otherwise.

```
#{if(function.isClientIpTrusted(['1.2.3.0/24','5.6.7.8/32'])...}
```

- **since**: 5.8.0.0

6.137 isConfigPropertyOn()

- **Signature**:

```
public boolean isConfigPropertyOn(String name, boolean defaultValue)
```

Allows to check if a "boolean" type configuration property is on, which means that it contains on of the values "true" / "on" / "yes", or if it is off, which is signaled by one of the values "false" / "off" / "no". NOTE: If the property contains a value that does not equal one of the previously mentioned values, e.g. "xyz", the function will return false.

- **parameter**: name

The name of the configuration property.

- **parameter**: defaultValue

The default value to use if the property does not exist.

- **return**:

The boolean value (true or false).

- **since**: 4.40.0.0

6.138 isHspSessionExpired()

- **Signature**:

```
public boolean isHspSessionExpired()
```

Allows to check if the HSP session is in the expired state. This could be the case, for example, if an authenticated client was inactive for too long.

```
${function.isHspSessionExpired()}
```

- **return**:

True if the HSP session has expired.

- **since**: 4.8.9

6.139 isHspSessionStepUp()

- **Signature**:

```
public boolean isHspSessionStepUp()
```

Allows to determine if the client is trying to perform a step-up authentication of the HSP session, in other words: Increasing the authentication level. <p> Typically this means that a client which has been authenticated on "Member" level is now trying to access an application that requires "Customer" authentication level, so an additional credential verification step may be performed. <p> This function could be used in conditions either within a model, in order to perform or leave out some steps, or in a JEXL expression to define a model trigger.

```
${function.isHspSessionStepUp()}
```

- **return**:

True if the client need to perform a step-up.

- **since**: 4.8.9

6.140 isIpInAnyNetworks()

- **Signature**:

```
public boolean isIpInAnyNetworks(String ip, List<String> networks)
```

Determines if the given IP is in any of the given networks.

Supports IPv4 IPs embedded into IPv6 addresses for the given IP, more precisely "compat", "6to4" and "terero" formats, but not "ISATAP" as that is easy to spoof; see the JavaDoc of Guava's InetAddresses class for more details. Networks are always considered either IPv4 or IPv6, never both; for embedded IPv4 networks test against the network in both IPv4 and IPv6 formats.

- **parameter**: ip

IP (IPv4 or IPv6 including IPv4 embedded in IPv6), e.g. "10.21.229.5" or "fd99:0:0:3::10".

- **parameter**: networks

List of networks (IPv4 or IPv6, embedded IPv4 treated as IPv6), each e.g. "10.21.228.0/22" or "fd99:0:0:3::10/64".

- **return**:

True if the IP is in any of the networks, false otherwise.

```
#{if(function.isIpInAnyNetworks('1.2.3.4',['5.6.7.0/24','1.2.3.0/24'])).../}
```

- **since**: 5.8.0.0

6.141 isIpInNetwork()

- **Signature**:

```
public boolean isIpInNetwork(String ip, String network)
```

Determines if the given IP is in the given network.

Supports IPv4 IPs embedded into IPv6 addresses for the given IP, more precisely "compat", "6to4" and "terero" formats, but not "ISATAP" as that is easy to spoof; see the JavaDoc of Guava's InetAddresses class for more details. Networks are always considered either IPv4 or IPv6, never both; for embedded IPv4 networks test against the network in both IPv4 and IPv6 formats.

- **parameter**: ip

IP (IPv4 or IPv6, including IPv4 embedded in IPv6), e.g. "10.21.229.5" or "fd99:0:0:3::10".

- **parameter**: network

Network (IPv4 or IPv6, embedded IPv4 treated as IPv6), e.g. "10.21.228.0/22" or "fd99:0:0:3::10/64".

- **return**:

True if the IP is in the network, false otherwise.

```
#{if(function.isIpInNetwork('1.2.3.4','1.2.3.0/24')).../}
```

- **since**: 5.8.0.0

6.142 isUserAllowedByFilter()

- **Signature**:

```
public boolean isUserAllowedByFilter()
```

Checks if the current user (a "username" credential with a value must exist in the session) is allowed by the "User Filter". This depends of course on the user filtering settings, if the feature is enabled at all. If the feature is disabled, the function always returns true.

```
if=${function.isUserAllowedByFilter()}
```

- **return**:

True if the current user is allowed, false if not.

- **since**: 5.3.0.0

6.143 isVerifiedCred()

- **Signature**:

```
public boolean isVerifiedCred(String semanticType)
```

DEPRECATED: Use "session.isVerifiedCred(String)" instead.

@deprecated Use {@link Session#isVerifiedCred(String)} function instead. * **parameter**: semanticType The type of the credential (password, username...).

- **return**:

"true" if it is verified "false" if not

- **since**: 4.6.0

6.144 logAudit()

- **Signature**:

```
public void logAudit(String text)
```

Writes the given text string to the audit log file with log level INFO.

- **parameter**: text

The text to write to the log file.

- **since**: 4.1.28

6.145 logCustomMessage()

- **Signature**:

```
public void logCustomMessage(String messageId)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details).

- **parameter**: messageId

The ID of the custom log message.

- **since**: 4.26.0

6.146 logCustomMessage()

- **Signature**:

```
public void logCustomMessage(String messageId, String parameter)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details). <p> The message text may contain variables as supported by the Java message

- **parameter**: messageId

The ID of the custom log message.

- **parameter**: parameter

- **since**: 4.27.0.3

6.147 logCustomMessage()

- **Signature**:

```
public void logCustomMessage(String messageId, String parameter1, String ↩
    parameter2)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details). <p> The message text may contain variables as supported by the Java message those variables in the order of their numbers (first entry in the parameter

- **parameter**: messageId

The ID of the custom log message.

- **parameter**: parameter1

- **parameter**: parameter2

- **since**: 4.27.0.3

6.148 logCustomMessage()

- **Signature**:

```
public void logCustomMessage(String messageId, Object ... params)
```

Writes a custom log message, specified by the custom log message properties with the prefix "log.message." (see "SLS Administration Guide" for details). <p> The message text may contain variables as supported by the Java message those variables in the order of their numbers (first entry in the parameter

- **parameter**: messageId

The ID of the custom log message.

- **parameter**: params

The parameters for the message text.

- **since**: 4.27.0.3

6.149 logInfo()

- **Signature**:

```
public void logInfo(String text)
```

Writes the given text string to the SLS log file with log level INFO.

- **parameter**: text

The text to write to the log file.

- **since**: 4.1.0

6.150 logToLog4j()

- **Signature**:

```
public void logToLog4j(String namedLogger, String level, String text)
```

Generic Log4j logging function, which allows to write a log message to a particular named logger. Check your SLS "log4j.xml" configuration file for "logger" tags to see all available loggers. The "SLS Administration Guide" lists all the usually available named loggers in the chapter "SLS Named Loggers".

- **parameter**: namedLogger

The name, like "audit", "exception" or "com.usp".

- **parameter**: level

The log level; must be either *TRACE*, *DEBUG*, *INFO*, *WARN* or *ERROR*.

- **parameter**: text

The text to be written to the log file.

- **since**: 4.11.1

6.151 md5()

- **Signature**:

```
public String md5(String input)
```

This method creates an md5 hash from a given string.

- **parameter**: input Any string.
- **return**:

An MD5 hash. @deprecated Use {@link #sha256(String)} or {@link #sha512(String)} instead.

- **since**: 4.7.2

6.152 md5hex()

- **Signature**:

```
public String md5hex(String input)
```

This method creates an md5 hash from a given string.

- **parameter**: input Any string.
- **return**:

An MD5 hash as a hex string. @deprecated Use {@link #sha256(String)} or {@link #sha512(String)} instead.

- **since**: 4.11.1

6.153 mergeArrays()

- **Signature**:

```
public Object[] mergeArrays(Object[] arrayOne, Object[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.3.6

6.154 mergeArrays()

- **Signature**:

```
public byte[] mergeArrays(byte[] arrayOne, byte[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.155 mergeArrays()

- **Signature**:

```
public short[] mergeArrays(short[] arrayOne, short[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.156 mergeArrays()

- **Signature**:

```
public int[] mergeArrays(int[] arrayOne, int[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.157 mergeArrays()

- **Signature**:

```
public long[] mergeArrays(long[] arrayOne, long[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.158 mergeArrays()

- **Signature**:

```
public float[] mergeArrays(float[] arrayOne, float[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.159 mergeArrays()

- **Signature**:

```
public double[] mergeArrays(double[] arrayOne, double[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.160 mergeArrays()

- **Signature**:

```
public char[] mergeArrays(char[] arrayOne, char[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.161 mergeArrays()

- **Signature**:

```
public boolean[] mergeArrays(boolean[] arrayOne, boolean[] arrayTwo)
```

Merges the given arrays into one array.

```
${newList = function.mergeArrays(listOne, listTwo)}
```

- **parameter**: arrayOne The first array.
- **parameter**: arrayTwo The second array.
- **return**:

The merged values.

- **since**: 4.24.0

6.162 nodeToXmlString()

- **Signature**:

```
public String nodeToXmlString(Node node)
```

Converts the given node to an XML string, including the leading <?xml ... ?> declaration.

Note that the string usually spans more than one text line.

The output of this function can e.g. be used to create a Groovy GPathResult object, which is much more elegant to use than a Node object.

```
#{def gpath = new XmlSlurper().parseText(function.nodeToXmlString(node))}
```

- **parameter**: node

The node to convert to an XML string.

- **return**:

The XML string or null if the node is null or the string could not be created.

- **since**: 4.27.0.0

6.163 numberToString()

- **Signature**:

```
public String numberToString(int number)
```

Converts a given numeric value to a string.

- **parameter**: number

The number to convert to a string.

- **return**:

The string representing the numerical value.

- **since**: 4.1.0

6.164 obfuscate()

- **Signature**:

```
public Object obfuscate(String obfuscatorId, String data)
```

Obfuscates a given data string based on configured properties in a reproducible (non-random) manner.

Sample properties: <pre> sls.obfuscator.<obfuscatorId>.item.1=9382134845
sls.obfuscator.<obfuscatorId>.item.2=${function.getVariable(*xy*)} </pre>

The properties values plus the given data are concatenated and then an MD5 hash is calculated and converted to a hex string and returned. The first item is used as a random salt value. All following items make up the actual hashed values.

Sample configuration for obfuscating user IDs in a SAML identity broker: <pre> sls.obfuscator.id.item.1=432452342563463
sls.obfuscator.id.item.2=${session.getCred(*username*)} sls.obfuscator.id.item.3=${sp.getSamlMessageIssuer()} </pre>

Sample returned string: "C9059EC129F3807F2F85D02EC6137442".

```
${obfuscatedId = function.obfuscate('id', id)}
```

- **parameter**: obfuscatorId

The obfuscator ID used to identify the obfuscator properties.

- **parameter**: data

The data to obfuscate.

- **return**:

Obfuscated data.

- **since**: 4.32.0

6.165 parseDate()

- **Signature**:

```
public Date parseDate(String value)
```

Parses a string value and returns an appropriate java.util.Date object, which could be used in a condition to perform a certain date / time comparison check etc. <p> See the Javadoc API description for "java.util.SimpleDateFormat" for detailed information about the functionality of that class.

- **parameter**: value The string value containing a date string.
- **return**:

The Date representing the string value (will be null if there was a parsing problem).

- **since**: 4.1.0

6.166 parseDate()

- **Signature**:

```
public Date parseDate(String value, String format)
```

Parses a string value and returns an appropriate java.util.Date object, which could be used in a condition to perform a certain date / time comparison check etc. <p> See the Javadoc API description for "java.util.SimpleDateFormat" for detailed information about the functionality of that class.

- **parameter**: value The string value containing a date string.
- **parameter**: format The string that describes the format of the *value* parameter.
- **return**:

The Date representing the string value (will be null if there was a parsing problem).

- **since**: 4.1.29

6.167 parseJson()

- **Signature**:

```
public Map<String,Object> parseJson(String json)
```

Parses the given JSON string using Groovy's JsonSlurper to a map with string keys and object values.
See also the chapter "Groovy Scripting" in the SLS Admin Guide for some samples how to elegantly modify the returned map.

- **parameter**: json

JSON string

- **return**:

JSON map with string keys and object values

- **since**: 5.15.0.0

6.168 prettyPrintJson()

- **Signature**:

```
public String prettyPrintJson(String json)
```

Renders the given JSON string to a multi-line JSON string for readable "pretty" output.

- **parameter**: json

JSON string (single-line or multi-line)

- **return**:

multi-line JSON string

- **since**: 5.15.0.0

6.169 printToConsole()

- **Signature**:

```
public void printToConsole(String text)
```

Prints the given text string to the console (the standard output of the Java process).

- **parameter**: text

The text to print.

- **since**: 4.1.0

6.170 random()

- **Signature**:

```
public String random()
```

This method returns a random string consisting of numbers.

- **return**:

A random string.

- **since**: 4.0.12

6.171 regex()

- **Signature**:

```
public String regex(String regex, String input)
```

Parse an input string with a regular expression and return the first group if it matches. The regular expression must have a group specified e.g. *(.)_**[a-z]**@acme.com* where (.*) is the first group

- **parameter**: regex A regular expression to apply.
- **parameter**: input The input string.
- **return**:

The first group of the input string if matched.

- **since**: 4.3.2

6.172 registerClassPrefix()

- **Signature**:

```
public void registerClassPrefix(String prefix, String className)
```

Allows to invoke static methods of Java classes by creating an instance of the class, with a given prefix. Then, all the static methods of that class can be invoked from that prefix. <p> For example, in order to use static methods of the Java class "java.lang.String", invoke the method with the prefix "String" and the class name "java.lang.String". Then, invoke static String methods by using the prefix "String".

```
${function.registerClassPrefix('Int','java.lang.Integer')}
```

- **parameter**: prefix
- **parameter**: className
- **since**: 4.9.7

6.173 renderJson()

- **Signature**:

```
public String renderJson(Map<String,Object> map)
```

Renders the given JSON map to a single-line JSON string suitable for logging/debugging.

- **parameter**: map

JSON map, may be null

- **return**:

single-line JSON string, or null if the given map was null

- **since**: 5.15.0.0

6.174 renderJson()

- **Signature**:

```
public String renderJson(Map<String,Object> map, boolean ↩
    heuristicallyBase64UrlEncodeByteArrays)
```

Renders the given JSON map to a single-line JSON string suitable for logging/debugging.

- **parameter**: map

JSON map * **parameter**: heuristicallyBase64UrlEncodeByteArrays If true heuristically tries to detect byte arrays and renders them as base64url-encoded string values; e.g. useful for logging/debugging WebAuthn messages.

- **return**:

single-line JSON string, or null if the given map was null

- **since**: 5.15.0.0

6.175 replace()

- **Signature**:

```
public String replace(String input, String oldStr, String newStr)
```

This method replaces a character sequence from a given string with a new sequence.

- **parameter**: input A given input string.
- **parameter**: oldStr The sequence of char values to be replaced.
- **parameter**: newStr The replacement sequence of char values.
- **return**:

A new string.

- **since**: 4.6.0

6.176 sendMail()

- **Signature**:

```
public void sendMail(String receiver, String sender, String subject, String ↩
    body, String mimeType)
```

Sends a eMail with the method parameters.

- **parameter**: receiver

email address of receiver(s), comma-separated list (mandatory)

- **parameter**: sender

email address of sender (mandatory)

- **parameter**: subject

email subject (mandatory)

- **parameter**: body

email body (mandatory)

- **parameter**: mimeType

MIME type of the mail (optional, defaults to *text/plain*).

- **since**: 4.11.5

6.177 sendMail()

- **Signature**:

```
public void sendMail(String receiver, String sender, String subject, String ↩
    body)
```

Sends a eMail with the method parameters.

- **parameter**: receiver

email address of receiver(s), comma-separated list (mandatory)

- **parameter**: sender

email address of sender (mandatory)

- **parameter**: subject

email subject (mandatory)

- **parameter**: body

email body (mandatory)

- **since**: 4.11.5

6.178 setVariable()

- **Signature**:

```
public void setVariable(String name, Object value)
```

Sets a custom JEXL variable in the current session. Groovy shortcut: <code>setVar(name, value)</code>.

- **parameter**: name

The name of the variable.

- **parameter**: value

The value of the variable.

- **since**: 4.7.0

6.179 sha1()

- **Signature**:

```
public String sha1(String input)
```

This method creates an SHA1 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA1 hash as a base64 string. @deprecated Use "crypto.sha256(String)" or "crypto.sha512(String)" instead.

- **since**: 4.7.1

6.180 sha1hex()

- **Signature**:

```
public String sha1hex(String input)
```

This method creates an SHA1 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA1 hash as a hex string. @deprecated Use {@link #sha256(String)} or {@link #sha512(String)} instead.

- **since**: 4.11.1

6.181 sha256()

- **Signature**:

```
public String sha256(String input)
```

This method creates an SHA256 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA256 hash as a base64 string. @deprecated Use "crypto.sha256(String)" instead.

- **since**: 4.7.1

6.182 sha256hex()

- **Signature**:

```
public String sha256hex(String input)
```

This method creates an SHA256 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA256 hash as a hex string. @deprecated Use "crypto.sha256hex(String)" instead.

- **since**: 4.11.1

6.183 sha512()

- **Signature**:

```
public String sha512(String input)
```

This method creates a base64 encoded SHA512 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA512 hash as a base64 string. @deprecated Use "crypto.sha512(String)" instead.

- **since**: 4.7.1

6.184 sha512hex()

- **Signature**:

```
public String sha512hex(String input)
```

This method creates a hex-encoded SHA512 hash from a given string.

- **parameter**: input Any string.
- **return**:

An SHA512 hash as a hex string. @deprecated Use "crypto.sha512hex(String)" instead.

- **since**: 4.11.1

6.185 signWithCertificate()

- **Signature**:

```
public String signWithCertificate(String certificateAlias, String dataToSign)
```

Creates a signature for the given data, using the certificate defined by the alias. Note: The certificate must have configured a valid private key, or it cannot be used to create signatures.

```
${signature = function.signWithCertificate('mycert', 'some data')}
```

- **parameter**: certificateAlias The alias that refers to a configuration property with the filename.
- **parameter**: dataToSign The data for which to create a signature.
- **return**:

The signature as a base64 string.

- **since**: 4.6.0

6.186 sleep()

- **Signature**:

```
public void sleep(int milliseconds)
```

Stops processing within the current thread for the specified number of milliseconds. This method should be used only for testing purposes, and with care, because it will block the current connection for the same time.

- **parameter**: milliseconds

The number of milliseconds to wait.

- **since**: 4.1.28

6.187 sortArray()

- **Signature**:

```
public Object[] sortArray(Object[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.3.6

6.188 sortArray()

- **Signature**:

```
public byte[] sortArray(byte[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.189 sortArray()

- **Signature**:

```
public short[] sortArray(short[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.190 sortArray()

- **Signature**:

```
public int[] sortArray(int[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.191 sortArray()

- **Signature**:

```
public long[] sortArray(long[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.192 sortArray()

- **Signature**:

```
public float[] sortArray(float[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.193 sortArray()

- **Signature**:

```
public double[] sortArray(double[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.194 sortArray()

- **Signature**:

```
public char[] sortArray(char[] array)
```

Sorts the given array in its natural order (for text strings, this is usually the alphabetical order).

```
${sorted = function.sortArray(someList)}
```

- **parameter**: array The array to sort.
- **return**:

The sorted array.

- **since**: 4.24.0

6.195 storeOpticalTokenImage()

- **Signature**:

```
public void storeOpticalTokenImage(String base64)
```

Stores the given base64 encoded image in the current session, so that in can later be displayed by a request sent to the "/auth/image" location of the SLS.

- **parameter**: base64

The base64 encoded image to store in the session.

- **since**: 4.27.0.3

6.196 stringToBytes()

- **Signature**:

```
public byte[] stringToBytes(String string)
```

Convert string to byte array using UTF-8 encoding.

- **parameter**: string

String

- **return**:

Byte array

- **since**: 5.14.0.0

6.197 stringToBytes()

- **Signature**:

```
public byte[] stringToBytes(String string, String charEncoding)
```

Convert string to byte array using given encoding.

- **parameter**: string

String

- **parameter**: charEncoding

Character to byte encoding, e.g. "UTF-8" or " ISO-8859-1".

- **return**:

Byte array

- **since**: 5.14.0.0

6.198 stringToNumber()

- **Signature**:

```
public int stringToNumber(String value)
```

Converts a given String with a numerical value into an integer number. The result of this method could then be used in calculations, counters etc.

- **parameter**: value

The string value with a number.

- **return**:

The number or zero, if the string could not be converted to a number.

- **since**: 4.1.0

6.199 stringTokenizer()

- **Signature**:

```
public String[] stringTokenizer(String value, String separator)
```

Parses a given string which consists of a list of values, separated by a specified separator character. The result is a String array which can then be used as input for all functions that can process arrays.

- **parameter**: value

The string with the values, separated by the given separator character.

- **parameter**: separator

The character that separates the values.

- **return**:

an Array of Strings that were separated by the <code>separator</code>

- **since**: 4.0.18

6.200 timestamp()

- **Signature**:

```
public String timestamp()
```

This function returns the current timestamp in a fix format ("yyyyMMddHHmmss").

- **return**:

A string with the current date/time, e.g. "20140325180247".

- **since**: 4.5.4

6.201 timestamp()

- **Signature**:

```
public String timestamp(String dateFormat)
```

This function returns the current timestamp in a fix format.

- **parameter**: dateFormat

The date format string, as documented in the Java "SimpleDateFormatter" class. See: http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html

- **return**:

A string with the current date/time, e.g. "20140325180247".

- **since**: 4.5.4

6.202 toLowerCase()

- **Signature**:

```
public String toLowerCase(String input)
```

Transforms all characters in a string to lowercase.

- **parameter**: input Any string.
- **return**:

The string in lowercase characters.

- **since**: 4.0.12

6.203 toUpperCase()

- **Signature**:

```
public String toUpperCase(String input)
```

Transforms all characters in a string to uppercase.

- **parameter**: input Any string.
- **return**:

The string in uppercase characters.

- **since**: 4.0.12

6.204 updateVarsFromJson()

- **Signature**:

```
public void updateVarsFromJson(String templateAlias, String json)
```

Creates / updates variables from the given JSON input string (usually a JSON response body from an HTTP/WS request)

```
function.updateVarsFromJson('mytemplate', response.content)
```

- **parameter**: templateAlias

The alias of the template with the variable definitions.

- **parameter**: json

The JSON data to process (e.g. the content of the response body of the last HTTP adapter request)

- **since**: 5.10.0.0

6.205 updateVarsFromXml()

- **Signature**:

```
public void updateVarsFromXml(String templateAlias, String xml)
```

Creates / updates variables from the given XML input string (usually an XML response body from an HTTP/WS request)

```
function.updateVarsFromXml('mytemplate', response.content)
```

- **parameter**: templateAlias

The alias of the template with the variable definitions.

- **parameter**: xml

The XML data to process (e.g. the content of the response body of the last HTTP adapter request)

- **since**: 5.10.0.0

6.206 urlDecode()

- **Signature**:

```
public String urlDecode(String input)
```

Decodes a URL-encoded string with charset UTF-8.

- **parameter**: input

The URL encoded string.

- **return**:

The URL decoded string (or empty if it could not be decoded).

- **since**: 4.0.13

6.207 urlDecode()

- **Signature**:

```
public String urlDecode(String input, String charset)
```

Decodes a URL-encoded string with given charset.

- **parameter**: input

The URL encoded string.

- **return**:

The URL decoded string (or empty if it could not be decoded).

- **since**: 5.13.0.0

6.208 urlEncode()

- **Signature**:

```
public String urlEncode(String input)
```

URL-encodes a plain string with UTF-8 charset.

- **parameter**: input

The plain string.

- **return**:

The URL encoded string (or empty if it could not be encoded).

- **since**: 4.0.13

6.209 urlEncode()

- **Signature**:

```
public String urlEncode(String input, String charset)
```

URL-encodes a plain string with given charset

- **parameter**: input

The plain string.

- **return**:

The URL encoded string (or empty if it could not be encoded).

- **since**: 5.13.0.0

6.210 xPathGetNodeList()

- **Signature**:

```
public NodeList xPathGetNodeList(String xml, String xpathExpression)
```

Allows to access a list of nodes from a given XML structure. If the XML contains namespaces, they must be coded into the XPath expression as well, like *//pre:myNode*. <p> See http://www.w3.org/TR/xpath/ for detail about XPath expression syntax.

```
function.xPathGetNodeList(xmlData,'//book[author=\"MrSmith\"]')
```

- **parameter**: xml

The XML data that should be processed.

- **parameter**: xpathExpression

The XPath expression to use.

- **return**:

An <code>org.w3c.dom.NodeList</code> object (or null).

- **since**: 4.8.10
- **since**: use 1.0.4

6.211 xPathGetNumber()

- **Signature**:

```
public double xPathGetNumber(String xml, String xpathExpression)
```

Allows to access an XML node which contains an integer number. If the XML contains namespaces, they must be coded into the XPath expression as well, like *//pre:myNode*. <p> See http://www.w3.org/TR/xpath/ for detail about XPath expression syntax.

```
function.xPathGetNumber(xmlData,'//book[author=\"MrSmith\"]/isbn')
```

- **parameter**: xml

The XML data that should be processed.

- **parameter**: xpathExpression

The XPath expression to use.

- **return**:

A Java "double" value.

- **since**: 4.8.10
- **since**: use 1.0.4

6.212 xPathGetSingleNode()

- **Signature**:

```
public Node xPathGetSingleNode(String xml, String xpathExpression)
```

Allows to retrieve one single node from a given XML structure. If the XML contains namespaces, they must be coded into the XPath expression as well, like *//pre:myNode*. <p> See http://www.w3.org/TR/xpath/ for detail about XPath expression syntax.

```
function.xPathGetSingleNode(xmlData,'//book[author=\"MrX\"]')
```

- **parameter**: xml

The XML data that should be processed.

- **parameter**: xpathExpression

The XPath expression to use.

- **return**:

An <code>org.w3c.dom.Node</code> object (or null).

- **since**: 4.8.10
- **since**: use 1.0.4

6.213 xPathGetString()

- **Signature**:

```
public String xPathGetString(String xml, String xpathExpression)
```

Allows to access an XML node which contains a string. If the XML contains namespaces, they must be coded into the XPath expression as well, like *//pre:myNode*. <p> See http://www.w3.org/TR/xpath/ for detail about XPath expression syntax.

```
function.xPathGetString(xmlData,'//book[author=\"MrX\"]/title')
```

- **parameter**: xml

The XML data that should be processed.

- **parameter**: xpathExpression

The XPath expression to use.

- **return**:

A string with the text value of XML node.

- **since**: 4.8.10
- **since**: use 1.0.4

6.214 xmlDecode()

- **Signature**:

```
public String xmlDecode(String input)
```

XML decodes a string.

For example, < is translated to <.

- **parameter**: input

The string to decode.

- **return**:

The decoded string, never null.

- **since**: 4.22.0
- **since**: use 1.0.5

6.215 xmlEncode()

- **Signature**:

```
public String xmlEncode(String input)
```

XML encodes a string.

For example, the less sign < is translated to <.

- **parameter**: input

The string to encode.

- **return**:

The encoded string, never null.

- **since**: 4.22.0
- **since**: use 1.0.5

Chapter 7

googleauth

JEXL functions to be used for the Google Authenticator login.

7.1 createQRcode()

- **Signature**:

```
public String createQRcode()
```

Returns a Base64-encoded PNG image with a QR code. Can be displayed in a web page as inline image.

- **return**:

The base64-encoded PNG image.

- **since**: 4.39.0

7.2 createSecret()

- **Signature**:

```
public String createSecret()
```

Creates a base32-encoded, 16 characters long secure random secret.

- **return**:

The secure secret, base32-encoded.

- **since**: 4.39.0

7.3 getSecret()

- **Signature**:

```
public String getSecret(boolean stripPadding, boolean formatHumanReadable)
```

Convenience method for getting the secret configured in the configuration property googleauth.secret.encoded.

- **parameter**: stripPadding

If "true", any padding characters ("=") will be stripped from the base32 encoding string. This is currently recommended since some implementations of the Google Authenticator app on iOS do not support padding characters.

- **parameter**: formatHumanReadable

If "true", the resulting string is broken up into groups of four characters, separated by blanks ("AAAA BBBB CCCC DDDD"). This representation can be used when the secret should be displayed in a JSP using the "getJexl" tag.

- **return**:

The configured secret.

- **since**: 4.39.0

Chapter 8

idp

SAML Identity Provider Adapter: Functions.

8.1 createAssertion()

- **Signature**:

```
public Assertion createAssertion()
```

Creates an assertion in the session, exactly like the model state "do.saml.idp.create.assertion". Once either the model state or this function have been executed, an assertion exists in the session, which can then be further adjusted (adding attributes etc).

Once the assertion is to be signed (and, depending on the configuration, encrypted), the function "sealAssertion()" can be invoked, which will create the finalized assertion (signed, and possibly encrypted). After that point, the assertion in the session is not mutable anymore, until this function or the model state are executed again.

Alternatively to the "sealAssertion()" function, the SAML model state "do.saml.idp.sendmsg" can be executed, which will also finalize the assertion and then directly send it to the target SP in a SAML response.

Note
The return value of this function is the assertion, but it is not necessary to use it in order to add attributes to it. Just calling any of the "idp.setAssertionAttribute. . . " functions will do that. The object returned by this function just allows for more elaborate access to the OpenSAML object structure, should that be required.

- **return**:

An assertion object of type "org.opensaml.saml2.core.Assertion".

- **since**: 5.10.0.0

8.2 createRandomId()

- **Signature**:

```
public String createRandomId()
```

Create a random ID string, suitable e.g. as a transient NameID in SAML Assertions.

```
${idp.createRandomId()}
```

- **return**:

A random ID.

- **since**: 4.12.0

8.3 getDecryptedData()

- **Signature**:

```
public byte[] getDecryptedData(String alias)
```

Allows to decrypt piggy-backed data from the otherwise unused field "ProviderName" in the current SAML request. The piggy-backed data is stored in there by the SLS service provider, using the corresponding JEXL function "sp_authn_request.addEncryptedData(String, byte[])", which encrypts the data and stores it as a Base64 string in the "ProviderName" field.

Note
This function can be used in model triggers, unlike the similar function "idp_authn_request.getDecryptedData()", which works only after the model state for processing the SAML request has been executed.

- **parameter**: alias

The alias for the "crypto." properties group.

- **return**:

The decrypted data or null, if none was found.

- **since**: 5.5.0.0

8.4 getMetadata()

- **Signature**:

```
public String getMetadata()
```

Get SAML 2.0 IdP Metadata as a multi-line XML string, useful e.g. for displaying in a JSP.

Uses the key pair alias as defined in config properties. If a future key pair alias is defined via config property, the corresponding certificate is additionally included.

```
${idp.getMetadata()}
```

- **return**:

Metadata or empty string if could not get metadata.

- **since**: 4.12.0

8.5 getMetadata()

- **Signature**:

```
public String getMetadata(String keyPairAlias)
```

Get SAML 2.0 IdP Metadata as a multi-line XML string, using the given key pair alias to retrieve the certificate.

If a future key pair alias is defined via config property, the corresponding certificate is additionally included.

```
${idp.getMetadata('idp')}
```

- **parameter**: keyPairAlias

The key pair alias to use for retrieving the IdP certificate.

- **return**:

Metadata or empty string if could not get metadata.

- **since**: 4.37.0

8.6 getSamlMessage()

- **Signature**:

```
public SAMLObject getSamlMessage()
```

Get the SAML message last received by the IdP with any previous HTTP request during the current login session.

Use e.g. for free handling of SAML messages in the login model, by directly accessing the OpenSAML 2.0 API, which directly mirrors the XML structure of the SAML message.

Implementation note and interaction with sp.getSamlMessage(): If idp.getSamlMessage() is called before do.saml.idp.handlemsg, the message is retrieved from a SAML credential created in the login session when receiving the HTTP request;

8.7 getSamlMessageAgeSecs()

- **Signature**:

```
public int getSamlMessageAgeSecs()
```

Determine the age in seconds of the last SAML message received with any previous HTTP request during the current login session. More precisely, this calculates the time difference in seconds between now and the time indicated in the IssueInstant attribute of the SAML message.

Use to recognize requests that came from bookmarked previous requests;

8.8 getSamlMessageBinding()

- **Signature**:

```
public String getSamlMessageBinding()
```

Get the binding with which the last SAML message was received by the IdP with any previous HTTP request during the current login session.

Possible return values include "redirect" and "post".

```
${idp.getSamlMessageBinding()}
```

- **return**:

SAML binding as string if any SAML message has been received, null otherwise

- **since**: 4.39.0

8.9 getSamlMessageIssuer()

- **Signature**:

```
public String getSamlMessageIssuer()
```

Get the issuer of the last SAML message received with any previous HTTP request during the current login session. More precisely, this gets the value of the Issuer attribute in the SAML message, i.e. the EntityID of the issuer.

Use e.g. to handle requests that came from bookmarked previous requests;

8.10 getSamlMessageIssuerAlias()

- **Signature**:

```
public String getSamlMessageIssuerAlias()
```

Get the SP alias of the issuer of the last SAML message received with any previous HTTP request during the current login session. More precisely, this gets the value of the Issuer attribute in the SAML message, i.e. the EntityID of the issuer and maps it to the corresponding SP alias, defaulting to the EntiyID if no SP alias has been defined in the configuration or no SP corresponds to the EntityID.

Use e.g. to handle requests that came from bookmarked previous requests;

8.11 getSamlMessageIssuerSpUrl()

- **Signature**:

```
public String getSamlMessageIssuerSpUrl()
```

Get the SP URL of the issuer of the last SAML message received with any previous HTTP request during the current login session. More precisely, this gets the value of the Issuer attribute in the SAML message, i.e. the EntityID of the issuer and maps it to the corresponding SP URL, defaulting to null if no SP URL has been defined in the configuration or no SP corresponds to the EntityID.

```
${idp.getSamlMessageIssuerSpUrl()}
```

- **return**:

URL or null if could not determine or no URL configured for the SP

- **since**: 4.26.0

8.12 getSloResults()

- **Signature**:

```
public ISloResult[] getSloResults()
```

Get results of SAML Single Logout (SLO) as an array in which each element represents the logout result from a single SP.

Typically used in a SLO result JSP.

The interface ISloResult has the following getters: <ul> boolean isLogoutSuccessful() String getLogutFailedReason() String getEntityId() String getAlias() String getUrl()

```
${idp.getSloResults().0.getLogutFailedReason()}
```

- **return**:

Array of SLO result, never null but may be empty.

- **since**: 4.14.0

8.13 getSpInfo()

- **Signature**:

```
public ISpInfo getSpInfo(String aliasOrEntityId)
```

Get info about the configured SP with given EntityID or alias.

```
${idp.getSpInfo('acme-sp')}
```

```
${idp.getSpInfo('acme-sp')}
${idp.getSpInfo('https://sp.acme.org/sp/')}
```

- **parameter**: aliasOrEntityId

SP alias or EntityID

- **return**:

info or null if no SP with given alias or EntityID or if given string is null

- **since**: 4.39.0

8.14 getSpInfos()

- **Signature**:

```
public ISpInfo[] getSpInfos()
```

Get info about all configured SPs as an array in which each element represents an SP info object.

Typically used in a JSP for displaying a selection of SPs to choose from for login.

The interface ISpInfo has the following getters: <ul> String getEntityId() String getAlias() String getUrl() String getIndex()

```
${idp.getSpInfos().0.getUrl()}
```

- **return**:

Array of SP info.

- **since**: 4.26.0

8.15 getSpUrl()

- **Signature**:

```
public String getSpUrl(String aliasOrEntityId)
```

Get configured SP URL for the SP with given alias or EntityID.

```
${idp.getSpUrl('acme-sp')}
```

```
${idp.getSpUrl('acme-sp')}
${idp.getSpUrl('https://sp.acme.org/sp/')}
```

- **parameter**: aliasOrEntityId

SP alias or EntityID

- **return**:

URL or null if could not determine or no URL configured for the SP

- **since**: 4.26.0

8.16 getSsoAttribute()

- **Signature**:

```
public String getSsoAttribute(String key)
```

DEPRECATED: Use "idp.getSsoAttributeValue(String)" instead.

@deprecated Use "idp.getSsoAttributeValue(String)" instead.

- **parameter**: key

The SSO attribute key.

- **return**:

The SSO attribute value. @throws SLSJexlRuntimeException if no SAML IdP context or no attribute with given key or not a single value

- **since**: 4.13.0

8.17 getSsoAttributeValue()

- **Signature**:

```
public String getSsoAttributeValue(String key)
```

Get SAML SSO attribute value (string) that has been configured for the given key, either from configured values or from previous login from user info cookie.

```
${idp.getSsoAttributeValue('email')}
```

- **parameter**: key

The SSO attribute key.

- **return**:

The SSO attribute value. @throws SLSJexlRuntimeException if no SAML IdP context or no attribute with given key or not a single value

- **since**: 4.24.0

8.18 getSsoAttributeValues()

- **Signature**:

```
public String[] getSsoAttributeValues(String key)
```

Get SAML SSO attribute value (string array) that has been configured for the given key, either from configured values or from previous login from user info cookie.

```
${idp.getSsoAttributeValues('email')}
```

- **parameter**: key

The SSO attribute key.

- **return**:

The SSO attribute value array, may be empty. @throws SLSJexlRuntimeException if no SAML IdP context or no attribute with given key

- **since**: 4.24.0

8.19 hasSamlMessage()

- **Signature**:

```
public boolean hasSamlMessage()
```

Determine if the IdP has received a SAML message with any previous HTTP request during the current login session.

Use e.g. to handle requests to the IdP's login page without a SAML AuthnRequest;

8.20 isAuthenticated()

- **Signature**:

```
public boolean isAuthenticated()
```

Determine if user is already authenticated at the IdP and credentials and attributes have been restored from the user info cookie.

Used typically to skip user authentication in the login model (SSO).

In order to return true, at least a previous login had to be successful, the SP must by configuration allow SSO and the AuthnRequest must not have its forceAuthn set to true.

Note that this function returns always false as long as neither an incoming SAML message has been handled with the do.saml.idp.handlemsg action nor (in case of an IdP-initated login) idp.setLoginSp() has been called. This is because the returned value depends on SP configuration (namely the idp.sp.<no>.sso property) and authentication can also be mandated in the AuthnRequest with the ForceAuthn attribute.

```
${idp.isAuthenticated()}
```

- **return**:

True if authenticated, false if not or no SAML IdP context found.

- **since**: 4.13.0

8.21 isKnownSp()

- **Signature**:

```
public boolean isKnownSp(String aliasOrEntityId)
```

Determines if there is a configured SP with given alias or EntityID.

```
${idp.isKnownSp('acme-sp')}
${idp.isKnownSp('https://sp.acme.org/sp/')}
```

- **parameter**: aliasOrEntityId

SP alias or EntityID

- **return**:

true or false

- **since**: 4.26.0

8.22 isMessageAuthnRequest()

- **Signature**:

```
public boolean isMessageAuthnRequest()
```

Determine if the IdP has received a SAML AuthnRequest message, i.e. a "login" request, with any previous HTTP request during the current login session.

```
${idp.isMessageAuthnRequest()}
```

- **return**:

true if got an AuthnRequest, false otherwise

- **since**: 5.5.0.0

8.23 isMessageLogoutResponse()

- **Signature**:

```
public boolean isMessageLogoutResponse()
```

Determine if the has received a SAML LogoutResponse message with any previous HTTP request during the current login session.

```
${idp.isMessageLogoutResponse()}
```

- **return**:

true if got a LogoutResponse, false otherwise

- **since**: 5.5.0.0

8.24 isReauthentication()

- **Signature**:

```
public boolean isReauthentication()
```

Determine if user has already been authenticated coming from the same SP.

This happes normally if there is an inactivity timeout at the SP, and possibly in this case it may be desired to ask explicitly for credentials even if isAuthenticated() is true.

Note that this function returns always false as long as neither an incoming SAML message has been handled with the do.saml.idp.handlemsg action nor (in case of an IdP-initated login) idp.setLoginSp() has been called. This is because the returned value depends on SP configuration (namely the idp.sp.<no>.sso property) and authentication can also be mandated in the AuthnRequest with the ForceAuthn attribute.

```
${idp.isReauthentication()}
```

- **return**:

True if is reauthentication, false if not or no SAML IdP context found.

- **since**: 4.22.0

8.25 isSloComplete()

- **Signature**:

```
public boolean isSloComplete()
```

Determine if SAML Single Logout (SLO) has been tried for all SPs - note that logout may have failed for some or all SPs, use <code>isSloSuccesful()</code> to determine if logout was successful for all SPs.

Used typically in the SLO model to determine if SLO is already complete.

```
${idp.isSloComplete()}
```

- **return**:

True if logout has been completed for all SPs (maybe with errors), false otherwise.

- **since**: 4.14.0

8.26 isSloSuccessful()

- **Signature**:

```
public boolean isSloSuccessful()
```

Determine if SAML Single Logout (SLO) has been tried for all SPs and was successful for all of them.

Typically used in a SLO result JSP.

```
${idp.isSloSuccessful()}
```

- **return**:

True if logout has been successful for all SPs, false otherwise.

- **since**: 4.14.0

8.27 sealAssertion()

- **Signature**:

```
public String sealAssertion()
```

Seals the assertion by signing it and, if necessary, encrypting it. Once this function has been called, the assertion in the session cannot be changed anymore.

- **return**:

An XML string containing the assertion.

- **since**: 5.10.0.0

8.28 setAssertionAttribute()

- **Signature**:

```
public void setAssertionAttribute(String nameFormat, String friendlyName,  ↩
    String name, String value)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The value type is set to string.

- **parameter**: nameFormat

The name format of the attribute, currently "basic" and "uri" are supported.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: value

The attribute value.

```
${idp.setAssertionAttribute('basic','uid','urn:....','myuser')}
```

- **since**: 4.17.0 (name format "uri" since 4.24.0)

8.29 setAssertionAttribute()

- **Signature**:

```
public void setAssertionAttribute(String nameFormat, String friendlyName,  ↩
    String name, String[] values)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The value type is set to string.

- **parameter**: nameFormat

The name format of the attribute, currently "basic" and "uri" are supported.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: values

The array of attribute values.

```
${idp.setAssertionAttribute('basic','uid','urn:oid:0.....1',myArray)}
```

- **since**: 4.24.0

8.30 setAssertionAttribute()

- **Signature**:

```
public void setAssertionAttribute(String nameFormat, String friendlyName,  ↩
    String name, String value, String valueType)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced.

- **parameter**: nameFormat

The name format of the attribute, currently "basic" and "uri" are supported.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: value

The attribute value.

- **parameter**: valueType

The value type, can be "string", "base64" or "base64.provided".

```
${idp.setAssertionAttribute('basic','uid','urn:....1','myuser','string')}
```

- **since**: 4.24.0

8.31 setAssertionAttribute()

- **Signature**:

```
public void setAssertionAttribute(String nameFormat, String friendlyName,  ↩
    String name, String[] values, String valueType)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced.

- **parameter**: nameFormat

The name format of the attribute, currently "basic" and "uri" are supported.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: values

The array of attribute values.

- **parameter**: valueType

The value type, can be "string", "base64" or "base64.provided".

```
${idp.setAssertionAttribute('basic','uid','urn:....1',myArray,'string')}
```

- **since**: 4.24.0

8.32 setAssertionAttribute()

- **Signature**:

```
public void setAssertionAttribute(AttributeWrapper wrapper)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced.

- **parameter**: wrapper

The attribute wrapper for the attribute.

```
${idp.setAssertionAttribute(wrapper)}
```

- **since**: 4.32.0

8.33 setAssertionAttributeBasic()

- **Signature**:

```
public void setAssertionAttributeBasic(String friendlyName, String name,  ↩
    String value)
```

Set basic profile attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The nameFormat is set to basic, the valueType to string.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: value

The attribute value.

```
${idp.setAssertionAttribute('basic','uid','urn:....1','myuser')}
```

- **since**: 4.24.0

8.34 setAssertionAttributeBasic()

- **Signature**:

```
public void setAssertionAttributeBasic(String friendlyName, String name,  ↩
    String[] values)
```

Set basic profile attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The nameFormat is set to basic, the valueType to string.

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: values

The array of attribute values.

```
${idp.setAssertionAttributeBasic('uid','urn:...1',myArray)}
```

- **since**: 4.24.0

8.35 setAssertionAttributeX500Ldap()

- **Signature**:

```
public void setAssertionAttributeX500Ldap(String friendlyName, String name,  ↩
    String value, String valueType)
```

Set X500/LDAP profile attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The nameFormat is set to X500/LDAP, the Endoding XML attribute of each value is set to "LDAP".

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: value

The attribute value.

- **parameter**: valueType

The value type, can be "string", "base64" or "base64.provided".

```
${idp.setAssertionAttributeX500Ldap('uid','urn:oid:...1','myuser','string')}
```

- **since**: 4.24.0

8.36 setAssertionAttributeX500Ldap()

- **Signature**:

```
public void setAssertionAttributeX500Ldap(String friendlyName, String name,  ↩
    String[] values, String valueType)
```

Set X500/LDAP profile attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced. The nameFormat is set to X500/LDAP, the Endoding XML attribute of each value is set to "LDAP".

- **parameter**: friendlyName

The friendly name of the attribute for human readers (e.g. "uid").

- **parameter**: name

The full name of the attribute (e.g. "urn:oid:0.9.2342.19200300.100.1.1").

- **parameter**: values

The array of attribute values.

- **parameter**: valueType

The value type, can be "string", "base64" or "base64.provided".

```
${idp.setAssertionAttributeX500Ldap('uid','urn:...1',myArray,'base64')}
```

- **since**: 4.24.0

8.37 setAssertionAttributes()

- **Signature**:

```
public void setAssertionAttributes(List<AttributeWrapper> wrappers)
```

Set attribute in list of attributes to be put into the SAML assertion's attribute statement. If the list contains already an attribute with the same name, it is replaced.

- **parameter**: wrappers

The attribute wrappers for the attributes.

```
${idp.setAssertionAttributes(wrappers)}
```

- **since**: 4.32.0

8.38 setLoginSp()

- **Signature**:

```
public void setLoginSp(String aliasOrEntityId)
```

Set the desired SP for IdP-initiated login with given EntityID or alias.

Must be an SP known to the IdP, i.e. configured including SP Metadata (which lists AssertionConsumerService endpoint URLs for login).

Note that for an IdP-initiated login, this function must be called before calling idp.isAuthenticated() or idp.isReauthentication(), until that both functions return always false. This is because the returned value of these functions depends on SP configuration (namely the idp.sp.<no>.sso property) and authentication can also be mandated in the AuthnRequest with the ForceAuthn attribute.

```
${idp.setLoginSp('acme-sp')}
```

```
${idp.setLoginSp('acme-sp')}
${idp.setLoginSp('https://sp.acme.org/sp/')}
```

- **parameter**: aliasOrEntityId

SP alias or EntityID

- **since**: 4.41.0.0

Chapter 9

idp_assertion

SAML Identity Provider Adapter: Script (JEXL/Groovy) wrapper for the SAML assertion created by the IdP. This wrapper is used to access the assertion before it is sent back to the SP. <p> NOTE: Before any of these functions can be used, the model state "do.saml.idp.create.assertion" must have been invoked. Otherwise, no assertion object exists in the session.

9.1 getOpenSamlAssertion()

- **Signature**:

```
public Assertion getOpenSamlAssertion()
```

Returns a reference to the OpenSAML-API object instance which represents the assertion that will be sent back to the SP. At this point, the assertion is not yet signed / encrypted (this will happen automatically before it is actually sent, during the model state "do.saml.idp.sendmsg").

Note
Will return null if the assertion has already been sealed (finalized) using the function "idp.sealAssertion()".

- **return**:

An instance of the class "org.opensaml.saml2.core.Assertion", or null if the assertion has already been sealed.

- **since**: 4.30.0

9.2 setAuthenticationMethod()

- **Signature**:

```
public void setAuthenticationMethod(String methodURN)
```

Sets the authentication method in the AuthnContext of the first AuthnStatement in the Assertion.

- **parameter**: methodURN

The authentication method. @throws SLSJexlRuntimeException If the assertion had already been finalized using "idp.sealAssertion()", because at that point, the authentication method cannot be changed anymore.

- **since**: 4.32.0

Chapter 10

idp_authn_request

SAML Identity Provider Adapter: Script (JEXL/Groovy) wrapper for the SAML AuthnRequest sent by an SP. This wrapper is used to access the SP request before it is sent to the IdP. <p> NOTE: Before any of these functions can be used, the model state "do.saml.idp.handlemsg" must have been invoked. Otherwise, no request object exists in the session.

10.1 containsRequestedAuthenticationMethod()

- **Signature**:

```
public boolean containsRequestedAuthenticationMethod(String methodURN)
```

Checks if the "AuthnRequest" contains a "RequestedAuthnContext" which defined one or multiple required / supported certain authentication types, e.g. "urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI" for certificate login.

- **parameter**: methodURN

The URN of the required / supported authentication method.

- **return**:

true if the given URN string was defined in the "AuthnRequest".

- **since**: 4.30.0.1

10.2 getDecryptedData()

- **Signature**:

```
public byte[] getDecryptedData(String alias)
```

Allows to decrypt piggy-backed data from the otherwise unused field "ProviderName" in the SAML "AuthnRequest". The piggy-backed data is stored in there by the SLS service provider, using the corresponding JEXL function "sp_authn_request.addEncryptedData(String, byte[])", which encrypts the data and stores it as a Base64 string in the "ProviderName" field.

Note
This function does NOT work in a model trigger, because it requires the model state "do.saml.idp.handlemsg" to be executed first. For model triggers, use the similar function "idp.getDecryptedData()" instead.

- **parameter**: alias

The alias for the "crypto." properties group.

- **return**:

The decrypted data, or null, if there was no or invalid data.

10.3 getIdpList()

- **Signature**:

```
public List<IDPEntry> getIdpList()
```

Returns the list of IDPs to be supported for the login.

- **return**:

The IdP list (may be null or empty).

- **since**: 4.30.0

10.4 getIdpListEntry()

- **Signature**:

```
public IDPEntry getIdpListEntry(String providerID)
```

Returns the IDPEntry for the given providerID if the AuthnRequest contains an IDPList and the list contains the given provider.

- **parameter**: providerID

The Entity ID of the IDP to look for.

- **return**:

IDPEntry if there is such an IDP in the list, null otherwise.

- **since**: 4.32.0

10.5 getOpenSamlRequest()

- **Signature**:

```
public AuthnRequest getOpenSamlRequest()
```

Returns a reference to the OpenSAML-API object instance which represents the authentication request that will be sent to the IdP. At this point, the request is not yet signed (this will happen automatically before it is actually sent, during the model state "do.saml.sp.sendmsg").

- **return**:

An instance of the class "org.opensaml.saml2.core.AuthnRequest".

- **since**: 4.30.0

10.6 getProxyCount()

- **Signature**:

```
public int getProxyCount()
```

Returns the value of the ProxyCount element, if there is one.

- **return**:

The ProxyCount value, or -1, if there is no ProxyCount element.

10.7 getRequestedAuthnContextComparison()

- **Signature**:

```
public String getRequestedAuthnContextComparison()
```

Allows to retrieve the value of the "AuthnContextComparison" element in the current "AuthnRequest" (usually a value of either "MINIMUM", "MAXIMUM", "EXACT" or "BETTER").

- **return**:

The comparison type string value (or null, if there was none).

- **since**: 4.30.0.1

10.8 hasIdpList()

- **Signature**:

```
public boolean hasIdpList()
```

Allows to check if the AuthnRequest contains an IDP list.

- **return**:

True if there is an IDP list.

- **since**: 4.30.0.1

10.9 hasProxyCount()

- **Signature**:

```
public boolean hasProxyCount()
```

Allows to check if the AuthnRequest contains a ProxyCount element.

- **return**:

True if there is a ProxyCount element.

10.10 idpListContains()

- **Signature**:

```
public boolean idpListContains(String providerID)
```

Checks if the AuthnRequest contains an IDPList, and if so, if the given IDP is in that list.

- **parameter**: providerID

The Entity ID of the IDP to look for.

- **return**:

True if there is such an IDP in the list.

- **since**: 4.30.0.1

Chapter 11

idp_response

SAML Identity Provider Adapter: Script (JEXL/Groovy) wrapper for SAML response of IdP in context. This wrapper is used to access the IdP response before it is sent back to the SP. <p> NOTE: Before any of these functions can be used, the model state "do.saml.idp.createmsg" must have been invoked. Otherwise, no response object exists in the session.

11.1 getOpenSamlAssertion()

- **Signature**:

```
public Assertion getOpenSamlAssertion()
```

Returns a reference to the OpenSAML-API object instance which represents the assertion that will be sent back to the SP. At this point, the assertion is not yet signed / encrypted (this will happen automatically before it is actually sent, during the model state "do.saml.idp.sendmsg").

- **return**:

An instance of the class "org.opensaml.saml2.core.Assertion".

- **since**: 4.30.0

11.2 getOpenSamlResponse()

- **Signature**:

```
public Response getOpenSamlResponse()
```

Returns a reference to the OpenSAML-API object instance which represents the success response that will be sent back to the SP. At this point, the response is not yet signed (this will happen automatically before it is actually sent, during the model state "do.saml.idp.sendmsg").

- **return**:

An instance of the class "org.opensaml.saml2.core.Response".

- **since**: 4.30.0

Chapter 12

jwt

Provides JSON Web Token (JWT) functions.

12.1 create()

- **Signature**:

```
public JWT create(String alias)
```

Creates a JWT from "jwt.def.<alias>.*" configuration properties. See the chapter "JSON Web Tokens" in the SLS Admin Guide for how to set up the configuration.

- **parameter**: alias

The alias in the config properties.

- **return**:

A com.nimbusds.jwt.JWT object.

- **since**: 5.5.0.0

12.2 getClaims()

- **Signature**:

```
public Map<String,Object> getClaims(JWT jwt)
```

Get claims of the given JWT.

- **parameter**: jwt

JWT object.

- **return**:

Claims or null if could not get them.

- **since**: 5.2.0.0

12.3 getIso8601UtcDateFor()

- **Signature**:

```
public String getIso8601UtcDateFor(Date date)
```

Get the given date+time in ISO 8601 UTC format, as needed for inside JWT claims that are represent a date+time. Example: "2016-11-16T12:34:53Z".

- **return**:

Date+time string in ISO 8601 UTC format.

- **since**: 5.5.0.0

12.4 getIso8601UtcDateForNow()

- **Signature**:

```
public String getIso8601UtcDateForNow()
```

Get the current date+time in ISO 8601 UTC format, as needed for inside JWT claims that are represent a date+time. Example: "2016-11-16T12:34:53Z".

- **return**:

Date+time string in ISO 8601 UTC format.

- **since**: 5.5.0.0

12.5 getSigningAlgorithm()

- **Signature**:

```
public String getSigningAlgorithm(JWT jwt)
```

Get the algorithm (alg) of the key used to sign the token.

- **parameter**: jwt

JWT object.

- **return**:

The algorithm or null if not signed.

- **since**: 5.2.0.0

12.6 getSigningKeyId()

- **Signature**:

```
public String getSigningKeyId(JWT jwt)
```

Get the key ID (kid) of the key used to sign the token.

- **parameter**: jwt

JWT object.

- **return**:

The key ID or null if not signed.

- **since**: 5.2.0.0

12.7 isSigned()

- **Signature**:

```
public boolean isSigned(JWT jwt)
```

Determine if given JWT is signed or not.

- **parameter**: jwt

JWT object obtained from parsing.

- **return**:

True if signed, false if not.

- **since**: 5.2.0.0

12.8 parse()

- **Signature**:

```
public JWT parse(String jwtString)
```

Parse given JWT string to a JWT object. The JWT string is typically several base64-encoded sequences separated by dots. The returned object is a com.nimbusds.jwt.JWT, see https://connect2id.com/products/nimbus-jose-jwt

- **parameter**: jwtString

JWT string to parse.

- **return**:

JWT object or null if parsing failed.

- **since**: 5.2.0.0

12.9 serialize()

- **Signature**:

```
public String serialize(JWT jwt)
```

Serialises the JSON Web Token (JWT) to its compact format consisting of Base64URL-encoded parts delimited by period (.) characters.

- **parameter**: jwt

JWT object.

- **return**:

Serialized string.

- **since**: 5.5.0.0

12.10 toMultiLineString()

- **Signature**:

```
public String toMultiLineString(JWT jwt)
```

Turn given jwt into a multi line string in a format suitable for human readers (for logging etc.).

- **parameter**: jwt

JWT object.

- **return**:

Multi line string, null if jwt was null.

- **since**: 5.5.0.0

12.11 toSingleLineString()

- **Signature**:

```
public String toSingleLineString(JWT jwt)
```

Turn given jwt into a single line string in a format suitable for human readers (for logging etc.).

- **parameter**: jwt

JWT object.

- **return**:

Single line string, null if jwt was null.

- **since**: 5.5.0.0

12.12 validateAudience()

- **Signature**:

```
public boolean validateAudience(JWT jwt, String allowedAudience)
```

Validate if the audience claim ("aud") of the given JWT contains the given allowed audience.

- **parameter**: jwt

JWT object.

- **parameter**: allowedAudience

The allowed audience.

- **return**:

True if the JWT contains the allowed audience in its "aud" claim.

- **since**: 5.2.0.0

12.13 validateAudience()

- **Signature**:

```
public boolean validateAudience(JWT jwt, String[] allowedAudiences)
```

Validate if the audience claim ("aud") of the given JWT contains any of the given allowed audiences.

- **parameter**: jwt

JWT object.

- **parameter**: allowedAudiences

The allowed audiences.

- **return**:

True if the JWT contains any of the allowed audiences in its "aud" claim.

- **since**: 5.2.0.0

12.14 validateSignatureNotExpired()

- **Signature**:

```
public boolean validateSignatureNotExpired(JWT jwt, String keystoreFileName,  ↩
    String keystoreType,
String keystorePass, String keyAlias)
```

Validate signature and that JWT has not expired. Note that the SLS currently only supports "RS256" signatures.

- **parameter**: jwt

JWT object.

- **parameter**: keystoreFileName

The name of the keystore file, absolute or relative to the webapp directory.

- **parameter**: keystoreType

The type of the keystore, if null, defaults to "JKS".

- **parameter**: keystorePass

The passphrase for keystore and key.

- **parameter**: keyAlias

The alias of the key/certificate in the keystore.

- **return**:

True if validated successfully, false if failed for technical reasons, signature is not correct or the JWT is not valid any more.

- **since**: 5.2.0.0

12.15 validateSignatureNotExpired()

- **Signature**:

```
public boolean validateSignatureNotExpired(JWT jwt, String certificateAlias)
```

Validate signature and that JWT has not expired. Note that the SLS currently only supports "RS256" signatures.

- **parameter**: jwt

JWT object.

- **parameter**: certificateAlias

The alias to use to look up the certificate file, from a config property "certificate.file.<alias>".

- **return**:

True if validated successfully, false if failed for technical reasons, signature is not correct or the JWT is not valid any more.

- **since**: 5.2.0.0

Chapter 13

ldap

Functions for performing custom LDAP operations in generic model states.

13.1 escapeDNValue()

- **Signature**:

```
public String escapeDNValue(String dnValue)
```

Performs LDAP escaping on a value of a key in a DN string. NOTE: Do NOT apply this function on a complete DN, because it will then escape ALL the DN characters like equals etc. Only use this to escape parts of DNs that need to be escaped, such as values coming from request parameters etc.

- **parameter**: dnValue

The string value that must be escaped.

- **return**:

The escaped value.

- **since**: 4.1.0

13.2 escapeFilterValue()

- **Signature**:

```
public String escapeFilterValue(String filterValue)
```

Performs LDAP escaping on a value of a key in a search filter string. NOTE: Do NOT apply this function on a complete search filter, because it will then escape ALL the filter characters like brackets etc. Only use this to escape parts of filters that need to be escaped, such as values coming from request parameters etc.

- **parameter**: filterValue

The string value that must be escaped.

- **return**:

The escaped value.

- **since**: 4.1.0

13.3 putLdapBackendSystem()

- **Signature**:

```
public void putLdapBackendSystem(String alias, String url)
```

Allows to enforce the use of a certain single LDAP back-end system in the current session. The function will just create a temporary backend in the current session that will be used for the given alias. The first parameter "alias" defines the group of back-end systems to which the second parameter "url" refers. For custom LDAP actions, the alias must be the same as the alias of the LDAP custom operation property group. For basic operations like authentication, the alias is empty (''). The second parameter must be a single LDAP URL. It is not possible to use multiple URLs (for failover or load-balancing) with this function!

```
${ldap.putLdapBackendSystem('', 'ldap://10.205.2.100:389')}
```

```
${ldap.putLdapBackendSystem('', 'ldap://10.205.2.100:389')}
${ldap.putLdapBackendSystem('userlookup-search', 'ldap://10.205.2.100:389')}
```

- **parameter**: alias The alias from the ldap-adapter.properties file. Use an empty string '' for adding a back-end to the group defined by "ldap.url", or enforcing the use of a certain back-end configured by that property. Use the alias of a custom operation property group to enforce the use of a certain LDAP back-end system for that operation, like "userlookup-search".
- **parameter**: url The url or hostname of the ldap system to use.
- **since**: 4.1.0

13.4 searchWithFilter()

- **Signature**:

```
public boolean searchWithFilter(String searchDN, String filter, String ↩
    userAlias)
```

Performs a search with a certain base DN and a filter. If an object is found, the JEXL variable "ldap.search.dn" will be set with the DN of that found object. Also, a JEXL variable "attribute.ldap.<attribute-name>" is created for each attribute of that object (or at least for each single-value attribute with a string or numeric value).

```
${ldap.searchWithFilter('ou=X,dc=com','(cn='+parameter.id+')','default')}
```

- **parameter**: searchDN

The search DN.

- **parameter**: filter

The filter string.

- **parameter**: userAlias

The alias of the tech user, such as *default*.

- **return**:

True if a result was found, false if not.

- **since**: 4.1.0

Chapter 14

mobileid

Provides general utility functions.

14.1 checkSignatureResponse()

- **Signature**:

```
public void checkSignatureResponse()
```

Verifies the response of a signature request. Status code must be 100 (REQUEST_OK). All other codes will throw an exception.

@throws SLSJexlRuntimeException If no valid MID response was received, or the MID status code was not "100".

- **since**: 5.14.0.0

14.2 checkStatusResponse()

- **Signature**:

```
public void checkStatusResponse()
```

Verifies the response of a status request. Mobile ID status code must be 500 (SIGNATURE). All other status codes will throw an exception, forcing the SLS to jump to the current failed state.

- **since**: 5.14.0.0

14.3 getSecureRandomText()

- **Signature**:

```
public String getSecureRandomText()
```


Creates a 4-digit random numeric text, shown to the user on the phone and the screen. NOTE: The value is created only once per session and then cached in the session. This allows to call this method to retreive the value to be set in the request and in the JSP.

- **return**:

The random 4-digit number string.

- **since**: 5.14.0.0

14.4 hasOutstandingTransaction()

- **Signature**:

```
public boolean hasOutstandingTransaction()
```

Allows to check if the backend is still waiting for the user to complete the transaction.

- **return**:

True if the Mobile ID status response is still "OUTSTANDING_TRANSACTION".

- **since**: 5.14.0.0

14.5 initMobileId()

- **Signature**:

```
public void initMobileId()
```

Initialize MID properties for signature request. This requires the following configuration properties to be set: <ul> MANDATORY: <code>mobileid.ap-id</code> OPTIONAL: <code>mobileid.ap-pwd</code> MANDATORY: <code>mobileid.mobilenumber</code> MANDATORY: <code>mobileid.msg-prefix</code> MANDATORY: <code>mobileid.msg-data</code> See "SLS Administration Guide", chapter "Mobile ID" for details about these configuration properties.

- **since**: 5.14.0.0

14.6 verifySignature()

- **Signature**:

```
public void verifySignature()
```

Verifies the signature. This requires the correct Swisscom CA certificate to be installed in the truststore.

- **since**: 5.14.0.0

Chapter 15

oidc_op

OIDC OP Adapter: Functions for the OpenID Connect (OIDC) OpenID Provider (OP) Adapter.

15.1 Object> createConfigAsMap()

- **Signature**:

```
public Map<String, Object> createConfigAsMap()
```

Creates the OIDC OP metadata configuration object. This object is also stored in a session variable "config_key".

- **return**:

A single-line JSON structure with the OIDC OP metadata.

- **since**: 5.17.0.0

15.2 Object> createJwksAsMap()

- **Signature**:

```
public Map<String, Object> createJwksAsMap()
```

Get map suitable for a JWKS endpoint, containing the public key for the currently configured key pair.

- **return**:

Map containing the JWKS information (or null if it could not be created). Note: The value returned is also stored in the session, under the key "jwk_key". If the method "getJwks()" is called afterwards, it will use that existing instance of the map. Therefore, this allows to call this method here first, then edit/change the map manually if required, and then generate the finalized JSON by calling "getJwks()".

- **since**: 5.17.0.0

15.3 getConfig()

- **Signature**:

```
public String getConfig()
```

Creates the OIDC OP metadata configuration JSON, used in the JSP for the OIDC OP discovery endpoint. It contains all endpoint URLs, supported ciphers, claims etc.

Note
If the method "createConfigAsMap()" is called before this one, this method here will generate the JSON based on the data already generated by the other method. This allows to invoke "createConfigAsMap()" first (e.g. in a model), then edit/change the map externally (i.e. using Groovy), and then have the final JSON be generated accordingly when invoking this method.

- **return**:

A single-line JSON structure with the OIDC OP metadata.

- **since**: 5.16.0.2

15.4 getJwks()

- **Signature**:

```
public String getJwks()
```

Get string suitable for a JWKS endpoint, containing the public key for the currently configured key pair.

Note
If the method "createJwkAsMap()" is called before this one, this method here will generate the JSON based on the data already generated by the other method. This allows to invoke "createJwkAsMap()" first (e.g. in a model), then edit/change the map externally (i.e. using Groovy), and then have the final JSON be generated accordingly when invoking this method.

- **return**:

JWKS JSON string or empty string if could not get.

- **since**: 5.16.0.0

15.5 getOidcRequestWrapper()

- **Signature**:

```
public OidcOpRequestWrapper getOidcRequestWrapper()
```

Gets a wrapper for accessing the OIDC request last received by the OP with any previous HTTP request during the current login session.

Use e.g. for free handling of OIDC requests in the login model. Note that this method always returns a wrapper instance, even if no OIDC request has been received, yet.

```
${oidc_op.getOidcRequestWrapper()}
```

- **return**:

oidc_op_request instance (a wrapper around some internal representation of an OIDC request), even if no OIDC request has been received (i.e. never returns null)

- **since**: 5.2.0.0

15.6 getRpInfo()

- **Signature**:

```
public IRpInfo getRpInfo(String clientId)
```

Get info about the configured RP with given client_id. (See oidc_op.getRpInfos() for the getters of the returned IRpInfo.)

```
${oidc_op.getRpInfo('acme')}
```

- **parameter**: clientId

client_id

- **return**:

info or null if no RP with given client_id or if given string is null

- **since**: 5.16.0.0

15.7 getRpInfos()

- **Signature**:

```
public IRpInfo[] getRpInfos()
```

Get info about all configured RPs as an array in which each element represents an RP info object.

The interface IRpInfo has the following getters: <ul> String getClientId() String getClientSecret() List<String> getRedirectUriList() boolean isPublicClient() boolan isPixyRequire() boolan isPixyAllowPlain() int getIndex()

```
${oidc_op.getRpInfos().0.getClientId()}
```

- **return**:

Array of RP info.

- **since**: 5.16.0.0

15.8 hasOidcRequest()

- **Signature**:

```
public boolean hasOidcRequest()
```

Determines if the OP has received an OIDC request with any previous HTTP request during the current login session.

```
${oidc_op.hasOidcRequest()}
```

- **return**:

true if got an OIDC request, false otherwise

- **since**: 5.2.0.0

Chapter 16

oidc_op_authorization_code

OIDC OP Adapter: Script (JEXL/Groovy) wrapper for accessing the authorization code in order to preserve information during the "Authorization Code Flow", between the initial Authentication Request, which obtains an authorization code, which is then used to get the token in the following Token Request. Provides access to userid and client_id, as well as to custom attributes that can be freely set in the login model.

16.1 getAttribute()

- **Signature**:

```
public Object getAttribute(String name)
```

Gets the attribute with the given name.

- **parameter**: name

attribute name

- **return**:

attribute value or null if no such attribute

- **since**: 5.14.0.0

16.2 getAttributeAsStringList()

- **Signature**:

```
public List<String> getAttributeAsStringList(String name)
```

Convenience function that gets the attribute converted to a list of strings.

If the attribute is a string or something else than a list/array of items, just the attribute converted to a list containing a single string value is returned; if the attribute is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the attribute does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored an attribute as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the attribute directly, except for different behaviour if the attribute was not set at all (returns empty list instead of null).

- **parameter**: name

attribute name

- **return**:

attribute value converted to a list of strings or empty list if no such attribute

- **since**: 5.14.0.0

16.3 getAttributeNames()

- **Signature**:

```
public Set<String> getAttributeNames()
```

Returns a set of the names of all attributes.

- **return**:

set of attributes, never null

- **since**: 5.14.0.0

16.4 getAuthenticationDate()

- **Signature**:

```
public Date getAuthenticationDate()
```

Gets date and time for when authentication was done, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

authentication date and time

- **since**: 5.14.0.0

16.5 getClientId()

- **Signature**:

```
public String getClientId()
```

Gets the client_id of the client (RP) for which authentication was done, set during oidc.op.handlemsg of the Authentication Request, i.e. available both both when processing Authentication Request and Token Request.

- **return**:

client_id

- **since**: 5.14.0.0

16.6 getExpirationDate()

- **Signature**:

```
public Date getExpirationDate()
```

Gets date and time for when the authorization code expires, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

expiration date and time

- **since**: 5.14.0.0

16.7 getNonce()

- **Signature**:

```
public String getNonce()
```

Gets the optional nonce for which authentication was done, set during oidc.op.handlemsg of the Authentication Request, i.e. available both both when processing Authentication Request and Token Request.

- **return**:

nonce

- **since**: 5.14.0.0

16.8 getUserid()

- **Signature**:

```
public String getUserid()
```

Gets the userid for which authentication was done, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

userid

- **since**: 5.14.0.0

16.9 removeAttribute()

- **Signature**:

```
public Object removeAttribute(String name)
```

Gets and removes the attribute with the given name.

- **parameter**: name

attribute name

- **return**:

previous attribute value or null if there was no such attribute

- **since**: 5.14.0.0

16.10 setAttribute()

- **Signature**:

```
public void setAttribute(String name, Object value)
```

Sets the attribute with given name to the given value.

If there was already an attribute with the same name, it is overwritten.

Note that if you set a list or array as value, you will get a list when you read it out after it has been parsed from the code parameter.

- **parameter**: name

attribute name

- **parameter**: value

attribute value

- **since**: 5.14.0.0

Chapter 17

oidc_op_id_token

OIDC OP Adapter: Script (JEXL/Groovy) wrapper for accessing the id_token before it is signed and sent to the RP, also wraps the access_token at the same time: The content of the access_token is kept identical to the one of the id_token, except that the access_token is separately enciphered and signed, see the SLS Administration Guide for more details. Note that in case of plain OAuth, these methods effectively only make changes to the access_token, which remains opaque to the client, is only processed by the SLS.

17.1 getClaim()

- **Signature**:

```
public Object getClaim(String name)
```

Gets the claim with the given name.

- **parameter**: name

claim name

- **return**:

claim value or null if no such claim

- **since**: 5.2.0.0

17.2 getClaimAsStringList()

- **Signature**:

```
public List<String> getClaimAsStringList(String name)
```

Convenience function that gets the claim converted to a list of strings.

If the claim is a string or something else than a list/array of items, just the claim converted to a list containing a single string value is returned; if the claim is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the claim does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored a claim as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the claim directly, except for different behaviour if the claim was not set at all (returns empty list instead of null).

- **parameter**: name

claim name

- **return**:

claim value converted to a list of strings or empty list if no such claim

- **since**: 5.14.0.0

17.3 getClaimNames()

- **Signature**:

```
public Set<String> getClaimNames()
```

Returns a set of the names of all claims.

- **return**:

set of claims, never null

- **since**: 5.2.0.0

17.4 removeClaim()

- **Signature**:

```
public Object removeClaim(String name)
```

Gets and removes the claim with the given name.

- **parameter**: name

claim name

- **return**:

previous claim value or null if there was no such claim

- **since**: 5.2.0.0

17.5 setClaim()

- **Signature**:

```
public void setClaim(String name, Object value)
```

Sets the claim with given name to the given value.

If there was already a claim with the same name, it is overwritten.

Note that if you set a list or array as value, you will get a list when you read it out from a parsed JWT.

- **parameter**: name

claim name

- **parameter**: value

claim value

- **since**: 5.2.0.0

Chapter 18

oidc_op_refresh_token

OIDC OP Adapter: Script (JEXL/Groovy) wrapper for accessing the refresh_token before it is signed and sent to the RP.

18.1 getClaim()

- **Signature**:

```
public Object getClaim(String name)
```

Gets the claim with the given name.

- **parameter**: name

claim name

- **return**:

claim value or null if no such claim

- **since**: 5.2.0.0

18.2 getClaimAsStringList()

- **Signature**:

```
public List<String> getClaimAsStringList(String name)
```

Convenience function that gets the claim converted to a list of strings.

If the claim is a string or something else than a list/array of items, just the claim converted to a list containing a single string value is returned; if the claim is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the claim does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored a claim as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the claim directly, except for different behaviour if the claim was not set at all (returns empty list instead of null).

- **parameter**: name

claim name

- **return**:

claim value converted to a list of strings or empty list if no such claim

- **since**: 5.14.0.0

18.3 getClaimNames()

- **Signature**:

```
public Set<String> getClaimNames()
```

Returns a set of the names of all claims.

- **return**:

set of claims, never null

- **since**: 5.2.0.0

18.4 removeClaim()

- **Signature**:

```
public Object removeClaim(String name)
```

gets and removes the claim with the given name.

- **parameter**: name

claim name

- **return**:

previous claim value or null if there was no such claim

- **since**: 5.2.0.0

18.5 setClaim()

- **Signature**:

```
public void setClaim(String name, Object value)
```

Sets the claim with given name to the given value.

If there was already a claim with the same name, it is overwritten.

Note that if you set a list or array as value, you will get a list when you read it out from a parsed JWT.

- **parameter**: name

claim name

- **parameter**: value

claim value

- **since**: 5.2.0.0

Chapter 19

oidc_op_request

OIDC OP Adapter: Script (JEXL/Groovy) wrapper for accessing the last OIDC request sent by an RP. This wrapper is used to access the RP request before it is validated by the OP. NOTE: If used before do.oidc.op.handlemsg, attempts to get the request from the original HTTP request stored in the SLS session; if used afterwards, the request is retrieved from the OIDC OP context.

19.1 getBasicAuthPassword()

- **Signature**:

```
public String getBasicAuthPassword()
```

Gets the password from a basic auth header if present.

- **return**:

password or null.

- **since**: 5.16.0.0

19.2 getBasicAuthUsername()

- **Signature**:

```
public String getBasicAuthUsername()
```

Gets the username from a basic auth header if present.

- **return**:

username or null.

- **since**: 5.16.0.0

19.3 getRequestHeader()

- **Signature**:

```
public String getRequestHeader(String name)
```

Returns the value of the indicated request header.

- **parameter**: name

header name

- **return**:

header value or null if there is no such header or no request

- **since**: 5.16.0.0

19.4 getRequestParameter()

- **Signature**:

```
public String getRequestParameter(String name)
```

Returns the value of the indicated OIDC parameter.

- **parameter**: name

parameter name

- **return**:

Parameter value or null if there is no such parameter or no request

- **since**: 5.2.0.0

19.5 getType()

- **Signature**:

```
public String getType()
```

Returns the type of the OIDC request, e.g. "authenticate" or "token".

- **return**:

Type or null if no request

- **since**: 5.2.0.0

19.6 hasRequest()

- **Signature**:

```
public boolean hasRequest()
```

Determine if the OP has received an OIDC request with any previous HTTP request during the current login session.

- **return**:

true if got an OIDC request, false otherwise

- **since**: 5.2.0.0

19.7 isOAuthAuthorizationRequest()

- **Signature**:

```
public boolean isOAuthAuthorizationRequest()
```

Returns true if the OIDC request is of type "auth" and has no scope parameter that contains the scope "openid", i.e. if this is a plain OAuth 2.0 Authorization request.

- **return**:

True if this is a plain OAuth Authorization request, false otherwise

- **since**: 5.9.0.0

19.8 isOidcAuthenticationRequest()

- **Signature**:

```
public boolean isOidcAuthenticationRequest()
```

Returns true if the OIDC request is of type "auth" and has a scope parameter that contains the scope "openid", i.e. if this is an OIDC Authentication request.

- **return**:

True if this is an OIDC Authentication request, false otherwise

- **since**: 5.9.0.0

Chapter 20

oidc_op_tokencode

Deprecated, use oidc_op_authorization_code instead. OIDC OP Adapter: Script (JEXL/Groovy) wrapper for accessing the authorization code (misnamed "token code" here) in order to preserve information during the "Authorization Code Flow", between the initial Authentication Request, which obtains an authorization code, which is then used to get the token in the following Token Request. Provides access to userid and client_id, as well as to custom attributes that can be freely set in the login model.

20.1 getAttribute()

- **Signature**:

```
public Object getAttribute(String name)
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets the attribute with the given name.

- **parameter**: name

attribute name

- **return**:

attribute value or null if no such attribute

- **since**: 5.2.0.0

20.2 getAttributeAsStringList()

- **Signature**:

```
public List<String> getAttributeAsStringList(String name)
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Convenience function that gets the attribute converted to a list of strings.

If the attribute is a string or something else than a list/array of items, just the attribute converted to a list containing a single string value is returned; if the attribute is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the attribute does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored an attribute as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the attribute directly, except for different behaviour if the attribute was not set at all (returns empty list instead of null).

- **parameter**: name

attribute name

- **return**:

attribute value converted to a list of strings or empty list if no such attribute

- **since**: 5.14.0.0

20.3 getAttributeNames()

- **Signature**:

```
public Set<String> getAttributeNames()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Returns a set of the names of all attributes.

- **return**:

set of attributes, never null

- **since**: 5.2.0.0

20.4 getAuthenticationDate()

- **Signature**:

```
public Date getAuthenticationDate()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets date and time for when authentication was done, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

authentication date and time

- **since**: 5.2.0.0

20.5 getClientId()

- **Signature**:

```
public String getClientId()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets the client_id of the client (RP) for which authentication was done, set during oidc.op.handlemsg of the Authentication Request, i.e. available both both when processing Authentication Request and Token Request.

- **return**:

client_id

- **since**: 5.2.0.0

20.6 getExpirationDate()

- **Signature**:

```
public Date getExpirationDate()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets date and time for when the authorization code expires, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

expiration date and time

- **since**: 5.2.0.0

20.7 getNonce()

- **Signature**:

```
public String getNonce()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets the optional nonce for which authentication was done, set during oidc.op.handlemsg of the Authentication Request, i.e. available both both when processing Authentication Request and Token Request.

- **return**:

nonce

- **since**: 5.2.0.0

20.8 getUserid()

- **Signature**:

```
public String getUserid()
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets the userid for which authentication was done, set during oidc.op.sendmsg of the Authentication Request handling, i.e. effectively only available during Token Request.

- **return**:

userid

- **since**: 5.2.0.0

20.9 removeAttribute()

- **Signature**:

```
public Object removeAttribute(String name)
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Gets and removes the attribute with the given name.

- **parameter**: name

attribute name

- **return**:

previous attribute value or null if there was no such attribute

- **since**: 5.2.0.0

20.10 setAttribute()

- **Signature**:

```
public void setAttribute(String name, Object value)
```

Deprecated, use corresponding oidc_op_authorization_code function instead.

Sets the attribute with given name to the given value.

If there was already an attribute with the same name, it is overwritten.

- **parameter**: name

attribute name

- **parameter**: value

attribute value

- **since**: 5.2.0.0

Chapter 21

oidc_op_userinfo

OIDC OP Adapter: JEXL wrapper for accessing the userinfo before it is sent to the RP.

21.1 getClaim()

- **Signature**:

```
public Object getClaim(String name)
```

Gets the claim with the given name.

- **parameter**: name

claim name

- **return**:

claim value or null if no such claim

- **since**: 5.2.0.0

21.2 getClaimAsStringList()

- **Signature**:

```
public List<String> getClaimAsStringList(String name)
```

Convenience function that gets the claim converted to a list of strings.

If the claim is a string or something else than a list/array of items, just the claim converted to a list containing a single string value is returned; if the claim is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the claim does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored a claim as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the claim directly, except for different behaviour if the claim was not set at all (returns empty list instead of null).

- **parameter**: name

claim name

- **return**:

claim value converted to a list of strings or empty list if no such claim

- **since**: 5.14.0.0

21.3 getClaimNames()

- **Signature**:

```
public Set<String> getClaimNames()
```

Returns a set of the names of all claims.

- **return**:

set of claims, never null

- **since**: 5.2.0.0

21.4 removeClaim()

- **Signature**:

```
public Object removeClaim(String name)
```

Gets and removes the claim with the given name.

- **parameter**: name

claim name

- **return**:

previous claim value or null if there was no such claim

- **since**: 5.2.0.0

21.5 setClaim()

- **Signature**:

```
public void setClaim(String name, Object value)
```

Sets the claim with given name to the given value.

If there was already a claim with the same name, it is overwritten.

Note that if you set a list or array as value, you will get a list when you read it out from a parsed JWT.

- **parameter**: name

claim name

- **parameter**: value

claim value

- **since**: 5.2.0.0

Chapter 22

oidc_rp

OIDC RP Adapter: Functions for the OpenID Connect (OIDC) Relying Party (RP) Adapter.

22.1 getAuthResponseWrapper()

- **Signature**:

```
public OidcRpAuthResponseWrapper getAuthResponseWrapper()
```

Gets a wrapper for accessing the authentication response received in response to a sent authentication request in this login session.

Use e.g. for free handling of authentication response in the login model. Note that this method always returns a wrapper instance, even if no authentication response has been received, yet.

```
${oidc_rp.getAuthResponseWrapper()}
```

- **return**:

oidc_rp_auth_response instance (a wrapper around some internal representation of an authentication response), even if no authentication request has been received (i.e. never returns null)

- **since**: 5.16.0.0

22.2 getClientInfo()

- **Signature**:

```
public IClientInfo getClientInfo(String alias)
```

Get info about configured client for the given alias as a client info object. (See oidc_rp.getClientInfos() for the getters of the returned IClientInfo.)

```
${oidc_rp.getClientInfo('myalias')}
```

- **parameter**: alias

alias

- **return**:

Client info for given alias or null if none.

- **since**: 5.16.0.0

22.3 getClientInfoForIssuerAndClientId()

- **Signature**:

```
public IClientInfo getClientInfoForIssuerAndClientId(String issuer, String ↩
    clientId)
```

Get info about configured client for the given issuer and client_id as a client info object. (See oidc_rp.getClientInfos() for the getters of the returned IClientInfo.)

```
${oidc_rp.getClientInfoForIssuerAndClientId('https://acme.org/', 'myalias')}
```

- **parameter**: issuer

issuer

- **parameter**: clientId

client_id

- **return**:

Client info for given alias or null if none.

- **since**: 5.16.0.0

22.4 getClientInfos()

- **Signature**:

```
public List<IClientInfo> getClientInfos()
```

Get info about all configured clients as a list in which each element represents a client info object.

The interface IClientInfo has the following getters: <ul> String getAlias() String getDescription() boolean isForCurrentVhost() String getIssuer() String getClientId() String getClientSecret() String getRedirectUri() List<String> getScopes() boolean isPublicClient() boolean isPixyUse() int getIndex()

```
${oidc_rp.getClientInfos().0.getAlias()}
```

- **return**:

Unmodifiable list of client info, may be empty but never null.

- **since**: 5.16.0.0

22.5 getClientInfosForIssuer()

- **Signature**:

```
public List<IClientInfo> getClientInfosForIssuer(String issuer)
```

Get info about all clients configured by the OP identified by issuer as a list in which each element represents a client info object. (See oidc_rp.getClientInfos() for the getters of the returned (list of) IClientInfo.)

```
${oidc_rp.getClientInfosForIssuer('https://acme.org/').0.getAlias()}
```

- **return**:

Unmodifiable list of client info, may be empty but never null.

- **since**: 5.16.0.0

22.6 getOpInfo()

- **Signature**:

```
public IOpInfo getOpInfo(String issuer)
```

Get info about configured OP for the given issuer as an OP info object. (See oidc_rp.getOpInfos() for the getters of the returned IOpInfo.)

```
${oidc_rp.getOpInfo('https://acme.org/')}
```

- **parameter**: issuer

issuer

- **return**:

OP info for given issuer or null if none.

- **since**: 5.16.0.0

22.7 getOpInfos()

- **Signature**:

```
public List<IOpInfo> getOpInfos()
```

Get info about all configured OPs as a list in which each element represents an OP info object.

The interface IOpInfo has the following getters: <ul> String getIssuer() String getAuthUri() String getTokenUri() String getJwksSourceType() String getJwksSource() int getIndex()

```
${oidc_rp.getOpInfos().0.getIssuer()}
```

- **return**:

Unmodifiable list of OP info, may be empty but never null.

- **since**: 5.16.0.0

22.8 getTokenResponseWrapper()

- **Signature**:

```
public OidcRpTokenResponseWrapper getTokenResponseWrapper()
```

Gets a wrapper for accessing the token response last received by the RP during the current login session (set in the OIDC RP context if the sendmsg http callout got a response from the OP).

Use e.g. for free handling of token response in the login model. Note that this method always returns a wrapper instance, even if no token response has been received, yet.

```
${oidc_rp.getTokenResponseWrapper()}
```

- **return**:

oidc_rp_token_response instance (a wrapper around some internal representation of a token response), even if no authentication request has been received (i.e. never returns null)

- **since**: 5.16.0.0

22.9 hasAuthResponse()

- **Signature**:

```
public boolean hasAuthResponse()
```

Determines if the RP has received an authentication response in response to a sent authentication request in this login session.

```
${oidc_rp.hasAuthResponse()}
```

- **return**:

true if got an authentication response, false otherwise

- **since**: 5.16.0.0

22.10 hasTokenResponse()

- **Signature**:

```
public boolean hasTokenResponse()
```

Determines if the RP has received a token response during the current login session (set in the OIDC RP context if the sendmsg http callout got a response from the OP).

```
${oidc_rp.hasTokenResponse()}
```

- **return**:

true if got a token response, false otherwise

- **since**: 5.16.0.0

Chapter 23

oidc_rp_id_token

OIDC RP Adapter: Script (JEXL/Groovy) wrapper for accessing the validated id_token.

23.1 getClaim()

- **Signature**:

```
public Object getClaim(String name)
```

Gets the claim with the given name.

- **parameter**: name

claim name

- **return**:

claim value or null if no such claim

- **since**: 5.16.0.0

23.2 getClaimAsStringList()

- **Signature**:

```
public List<String> getClaimAsStringList(String name)
```

Convenience function that gets the claim converted to a list of strings.

If the claim is a string or something else than a list/array of items, just the claim converted to a list containing a single string value is returned; if the claim is a list/array of items, each item is converted to a string (toString()) and returned as a list of strings; if the claim does not exist (or contains an empty list/array), an empty list is returned.

Note that if you previously stored a claim as a list or array of strings, you will automatically get it back as a list (not an array) of strings after parsing, i.e. in that case calling this method makes no difference from getting the claim directly, except for different behaviour if the claim was not set at all (returns empty list instead of null).

- **parameter**: name

claim name

- **return**:

claim value converted to a list of strings or empty list if no such claim

- **since**: 5.16.0.0

23.3 getClaimNames()

- **Signature**:

```
public Set<String> getClaimNames()
```

Returns a set of the names of all claims.

- **return**:

set of claims, never null

- **since**: 5.16.0.0

23.4 getJwt()

- **Signature**:

```
public JWT getJwt()
```

Returns the JWT object of the id_token; use jwt.* script functions to extract items or to get the id_token in its serialized form, or directly use the methods of com.nimbusds.jwt.JWT, see https://connect2id.com/products/nimbus-jose-jwt.

```
#{idToken = jwt.serialize(oidc_rp_id_token.getJwt())}
```

- **return**:

JWT object

- **since**: 5.16.0.0

Chapter 24

passwordpolicy

Wrapper class for the SLS password policy for Groovy/JEXL.

24.1 getInclusionPattern()

- **Signature**:

```
public String getInclusionPattern()
```

Returns the character inclusion regex pattern of the policy.

- **return**:

The inclusion regex pattern string.

- **since**: 4.27.0.3

24.2 getLowerCaseCount()

- **Signature**:

```
public int getLowerCaseCount()
```

Returns an integer value with a code which tells if and how often lowercase characters must appear in a password. -1 = Must not appear (not allowed) 0 = Allowed 1 = Must contain lowercase characters There is also a function "getLowerCaseOccurrence()", which returns string constant values instead of integers. But for reasons of backward compatibility, this funtion with numeric values is also still available.

- **return**:

A code ranging from -1 to 1.

- **since**: 4.27.0.3

24.3 getLowerCaseOccurrence()

- **Signature**:

```
public String getLowerCaseOccurrence()
```

Returns a string value with a code which tells if and how often lowercase characters must appear in a password. NONE = Must not appear (not allowed) ALLOW = Allowed MUST = Must contain lowercase characters

- **return**:

The string value, either "NONE", "ALLOW" or "NONE".

- **since**: 4.27.0.3

24.4 getMaximumLength()

- **Signature**:

```
public int getMaximumLength()
```

Returns the maximum length for a password.

- **return**:

The maximum password length.

- **since**: 4.27.0.3

24.5 getMinimumLength()

- **Signature**:

```
public int getMinimumLength()
```

Returns the minimum length for a password.

- **return**:

The minimum password length.

- **since**: 4.27.0.3

24.6 getNumericCount()

- **Signature**:

```
public int getNumericCount()
```

Returns an integer value with a code which tells if and how often numeric characters must appear in a password. -1 = Must not appear (not allowed) 0 = Allowed 1 = Must contain numeric characters There is also a function "getNumericOccurrence()", which returns string constant values instead of integers. But for reasons of backward compatibility, this funtion with numeric values is also still available.

- **return**:

A code ranging from -1 to 1.

- **since**: 4.27.0.3

24.7 getUpperCaseCount()

- **Signature**:

```
public int getUpperCaseCount()
```

Returns an integer value with a code which tells if and how often uppercase characters must appear in a password. -1 = Must not appear (not allowed) 0 = Allowed 1 = Must contain uppercase characters There is also a function "getUpperCaseOccurrence()", which returns string constant values instead of integers. But for reasons of backward compatibility, this funtion with numeric values is also still available.

- **return**:

A code ranging from -1 to 1.

- **since**: 4.27.0.3

24.8 getUpperCaseOccurrence()

- **Signature**:

```
public String getUpperCaseOccurrence()
```

Returns a string value with a code which tells if and how often uppercase characters must appear in a password. NONE = Must not appear (not allowed) ALLOW = Allowed MUST = Must contain uppercase characters

- **return**:

The string value, either "NONE", "ALLOW" or "NONE".

- **since**: 4.27.0.3

24.9 setInclusionPattern()

- **Signature**:

```
public void setInclusionPattern(String pattern)
```

Allows to define the setting for the regex pattern for allowed characters.

- **parameter**: pattern

The inclusion regex pattern.

- **since**: 4.27.0.3

24.10 setLowerCaseOccurrence()

- **Signature**:

```
public void setLowerCaseOccurrence(String value)
```

Allows to define the setting for lowercase characters.

- **parameter**: value

Must bei either "ALLOW", "MUST" or "NONE".

- **since**: 4.27.0.3

24.11 setMaximumLength()

- **Signature**:

```
public void setMaximumLength(int maximumLength)
```

Allows to define the setting for maximum password length.

- **parameter**: maximumLength

The maximum length for a password.

- **since**: 4.27.0.3

24.12 setMinimumLength()

- **Signature**:

```
public void setMinimumLength(int minimumLength)
```

Allows to define the setting for minimum password length.

- **parameter**: minimumLength

The minimum length for a password.

- **since**: 4.27.0.3

24.13 setNumericOccurrence()

- **Signature**:

```
public void setNumericOccurrence(String value)
```

Allows to define the setting for numeric characters.

- **parameter**: value

Must bei either "ALLOW", "MUST" or "NONE".

- **since**: 4.27.0.3

24.14 setUpperCaseOccurrence()

- **Signature**:

```
public void setUpperCaseOccurrence(String value)
```

Allows to define the setting for uppercase characters.

- **parameter**: value

Must bei either "ALLOW", "MUST" or "NONE".

- **since**: 4.27.0.3

Chapter 25

pki

PKI Adapter: Functions.

25.1 setTrustGroup()

- **Signature**:

```
public void setTrustGroup(String alias)
```

Sets the desired trust group for the PKI adapter in the SLS session.

- **parameter**: alias

trust group alias (pass null to unset)

- **since**: 4.41.0.0 @throws SLSJexlRuntimeException If the trust group alias is not configured.

Chapter 26

response

Provides methods for sending custom response headers, redirecting etc.

26.1 addHeader()

- **Signature**:

```
public void addHeader(String name, String value)
```

Adds a custom response header value. This function allows to set a response header with the same name, but different values multiple times. <p> If this header is meant to be received by the browser client, it must be enabled in the SRM login service location configuration as well;

26.2 clearAuthorizations()

- **Signature**:

```
public void clearAuthorizations()
```

Clear all session authorizations of usage type "ac-app-az".

- **since**: 4.3.8

26.3 createCookie()

- **Signature**:

```
public void createCookie(String name, String value, String path, int maxAge)
```

Creates a custom cookie and adds it to the HttpResponse.

- **parameter**: name

The name of the cookie.

- **parameter**: value

The value string of the cookie.

- **parameter**: path

The URL path for which the cookie should be set.

- **parameter**: maxAge

The maximum age of the cookie, specified in seconds. -1 indicates the cookie will only persist until browser shutdown. 0 will delete the cookie immediately.

- **since**: 4.0.18

26.4 createCookie()

- **Signature**:

```
public void createCookie(String name, String value, String path, String domain ↩
    ,
int maxAge)
```

Creates a custom cookie and adds it to the HttpResponse.

- **parameter**: name

The name of the cookie.

- **parameter**: value

The value string of the cookie.

- **parameter**: path

The URL path for which the cookie should be set.

- **parameter**: domain

The domain, can be null.

- **parameter**: maxAge

The maximum age of the cookie, specified in seconds. -1 indicates the cookie will only persist until browser shutdown. 0 will delete the cookie immediately.

- **since**: 4.5.5

26.5 createCookie()

- **Signature**:

```
public void createCookie(String name, String value, String path, String domain ↩
    ,
int maxAge, int version)
```

DEPRECATED: Use "createCookie(String name, String value, String path, String domain, int maxAge)" instead.

26.6 propagateBasicAuthHeaderToApp()

- **Signature**:

```
public void propagateBasicAuthHeaderToApp()
```

Propagates a "Basic Authentication" header to the application. In other words, after the login, each request sent to the application will contain an "Authorization" header with the basic authentication information. The header will be propagated to all URL locations that belong to the current "authorized path". NOTE 1: This header will only be actually created after a successful login (during the "do.success" state of the login model). NOTE 2: This function can only be used in a login model (or with an authentication adapter) where the credentials of type *USERNAME* and *PASSWORD* are made available to the SLS. In an NTLM login, for example, the SLS never receives the clients password, so this function will not work there, and will fail with a runtime exception. NOTE 3: If the *username* and *password* credentials are not available in the login session and verified, this function will throw an exception. NOTE 4: If neither the *requested authorized path* nor the *requested page path* are available, this function will throw an exception.

```
${response.propagateBasicAuthHeaderToApp()}
```

@throws SLSJexlRuntimeException If either the *USERNAME* or the *PASSWORD* credential was missing, or the path for the propagated header could not be determined.

- **since**: 4.12.1

26.7 propagateBasicAuthHeaderToApp()

- **Signature**:

```
public void propagateBasicAuthHeaderToApp(String path)
```

Propagates a "Basic Authentication" header to the application. In other words, after the login, each request sent to the application will contain an "Authorization" header with the basic authentication information. The header will be propagated to all URL locations that belong to the current "authorized path". NOTE 1: This header will only be actually created after a successful login (during the "do.success" state of the login model). NOTE 2: This function can only be used in a login model (or with an authentication adapter) where the credentials of type *USERNAME* and *PASSWORD* are made available to the SLS. In an NTLM login, for example, the SLS never receives the clients password, so this function will not work there, and will fail with a runtime exception. NOTE 3: If the *username* and *password* credentials are not available in the login session and verified, this function will throw an exception. NOTE 4: If the given path is *null*, this function will throw an exception.

```
${response.propagateBasicAuthHeaderToApp('/demoapp')}
```

- **parameter**: path

The URI path to which to propagate the header. Must be equal to or a sub-path of the current "authorized path". @throws SLSJexlRuntimeException If either the *USERNAME* or the *PASSWORD* credential was missing, or the path for the propagated header could not be determined.

- **since**: 4.12.1

26.8 propagateBasicAuthHeaderToApp()

- **Signature**:

```
public void propagateBasicAuthHeaderToApp(String username, String password)
```

Propagates a "Basic Authentication" header to the application. In other words, after the login, each request sent to the application will contain an "Authorization" header with the basic authentication information. The header will be propagated to all URL locations that belong to the current "authorized path". NOTE 1: This header will only be actually created after a successful login (during the "do.success" state of the login model). NOTE 2: This function can only be used in a login model (or with an authentication adapter) where the credentials of type *USERNAME* and *PASSWORD* are made available to the SLS. In an NTLM login, for example, the SLS never receives the clients password, so this function will not work there, and will fail with a runtime exception.

```
${response.propagateBasicAuthHeaderToApp(parameter.id, parameter.pwd)}
```

- **parameter**: username

The username to use in the basic auth header.

- **parameter**: password

The password to use in the basic auth header. @throws SLSJexlRuntimeException If either the *USERNAME* or the *PASSWORD* credential was missing.

- **since**: 4.12.1

26.9 propagateBasicAuthHeaderToApp()

- **Signature**:

```
public void propagateBasicAuthHeaderToApp(String username, String password,  ↩
    String path)
```

Propagates a "Basic Authentication" header to the application. In other words, after the login, each request sent to the application will contain an "Authorization" header with the basic authentication information. Note: This header will only be actually created after a successful login (during the "do.success" state of the login model).

```
${response.propagateBasicAuthHeaderToApp(parameter.id, parameter.pwd, '/')}
```

- **parameter**: username

The username to use in the basic auth header.

- **parameter**: password

The password to use in the basic auth header.

- **parameter**: path

The URI path to which to propagate the header. Must be equal to or a sub-path of the current "authorized path". @throws SLSJexlRuntimeException If either the *USERNAME* or the *PASSWORD* credential was missing.

- **since**: 4.12.1

26.10 propagateHeaderToApp()

- **Signature**:

```
public void propagateHeaderToApp(String name, String value)
```

Instructs the HSP to propagate a custom HTTP header to the application with each request (once the login process has completed). The header will be propagated to every URI below the requested authorized path as defined in the SRM location configuration. <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed). <p> Encoding will be "url" if the config property "session-attribute.encodings" is true, otherwise "string".

- **parameter**: name

The name of the custom HTTP header.

- **parameter**: value

The value of the header.

- **since**: 4.0.18

26.11 propagateHeaderToApp()

- **Signature**:

```
public void propagateHeaderToApp(String name, String value, String path)
```

Instructs the HSP to propagate a custom HTTP header to the application with each request (once the login process has completed). <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed). <p> Encoding will be "url" if the config property "session-attribute.encodings" is true, otherwise "string".

- **parameter**: name

The name of the custom HTTP header.

- **parameter**: value

The value of the header.

- **parameter**: path

The URL path to which to propagate the header (usually the base path of the application).

- **since**: 4.0.18

26.12 propagateHeaderToApp()

- **Signature**:

```
public void propagateHeaderToApp(String name, String value, boolean ↩
    isValueEncoded, String encoding)
```

Instructs the HSP to propagate a custom HTTP header to the application with each request (once the login process has completed). The header will be propagated to every URI below the requested authorized path as defined in the SRM location configuration. <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed). <p> (Note that the mechanism with value provided as encoded is meant to be used with encoding "url" (or "base64"), but not with the legacy encoding "string".)

- **parameter**: name

The name of the custom HTTP header.

- **parameter**: value

The value of the header. * **parameter**: isValueEncoded Whether the given value is already encoded or not; if not already encoded, it will be encoded according to the indicated encoding. * **parameter**: encoding The encoding ("url", "base64", "string").

- **since**: 5.13.0.0

26.13 propagateHeaderToApp()

- **Signature**:

```
public void propagateHeaderToApp(String name, String value, String path, ↩
    boolean isValueEncoded, String encoding)
```

Instructs the HSP to propagate a custom HTTP header to the application with each request (once the login process has completed). <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed). <p> (Note that the mechanism with value provided as encoded is meant to be used with encoding "url" (or "base64"), but not with the legacy encoding "string".)

- **parameter**: name

The name of the custom HTTP header.

- **parameter**: value

The value of the header.

- **parameter**: path

The URL path to which to propagate the header (usually the base path of the application). * **parameter**: isValueEncoded Whether the given value is already encoded or not; if not already encoded, it will be encoded according to the indicated encoding. * **parameter**: encoding The encoding ("url", "base64", "string").

- **since**: 5.13.0.0

26.14 removeAuthorization()

- **Signature**:

```
public boolean removeAuthorization(String authId, String path)
```

Clear the session authorization of usage type "ac-app-az" which identifier equals the parameter given

- **parameter**: authId the authorization identifier
- **parameter**: path

The path of the authorization.

- **return**:

true if the authorization was found and successfully removed, false otherwise.

- **since**: 4.3.8

26.15 setAaiVariable()

- **Signature**:

```
public void setAaiVariable(String variable, String value, String path)
```

Sets the value of an AAI variable used in an SRM configuration file (for a transparent form login). <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed).

- **parameter**: variable

The name of the variable (must correspond to the name used in the SRM AAI script).

- **parameter**: value

The value for the variable (usually the username or password).

- **parameter**: path

The URL path for which to set the variable (usually the base path of the application for which a transparent form login must be performed).

- **since**: 4.0.18

26.16 setAuthorization()

- **Signature**:

```
public void setAuthorization(String authorization)
```

Sets an authorization value which must correspond to a value defined in a "AC_RequireAz" directive in the SRM location configuration. The authorization will be set for every URI below the requested authorized path as defined in the SRM location configuration.

- **parameter**: authorization

The authorization string.

- **since**: 4.0.18

26.17 setAuthorization()

- **Signature**:

```
public void setAuthorization(String authorization, String path)
```

Sets an authorization value which must correspond to a value defined in a "AC_RequireAz" directive in the SRM location configuration.

- **parameter**: authorization

The authorization string.

- **parameter**: path

The path for which to set the authorization.

- **since**: 4.0.18

26.18 setAuthorizations()

- **Signature**:

```
public void setAuthorizations(String content, String delimiter)
```

Sets a list of authorizations, which are read from the content string, separated by the given delimiter.

- **parameter**: content

A string containing all authorization values.

- **parameter**: delimiter

The separator for the authorization values in the content string.

- **since**: 4.1.1

26.19 setAuthorizations()

- **Signature**:

```
public void setAuthorizations(String content, String delimiter,
String path)
```

Sets a list of authorizations, which are read from the content string, separated by the given delimiter. The authorizations are set for a specific URI path.

- **parameter**: content

A string containing all authorization values.

- **parameter**: delimiter

The separator for the authorization values in the content string.

- **parameter**: path

The path for which to set the authorizations.

- **since**: 4.0.18

26.20 setBasicAuthRequiredHeaders()

- **Signature**:

```
public void setBasicAuthRequiredHeaders()
```

Sets HTTP headers which request basic authentication from the client.

- **since**: 4.3.12

26.21 setHeader()

- **Signature**:

```
public void setHeader(String name, String value)
```

Sets a custom response header. If the same header has already been set in the response, its value will be overwritten (use <code>addHeader()</code>, if you need to set a header with multiple values). <p> If the same header has already been set with multiple values in the response, its first value will be overwritten. <p> If this header is meant to be received by the browser client, it must be enabled in the SRM login service location configuration as well;

26.22 setHttpStatusCode()

- **Signature**:

```
public void setHttpStatusCode(int value)
```

Sets the HTTP status code.

Note that by default the web.xml of the SLS webapp maps the error codes 403, 404 and 500 to the SystemError.jsp, i.e. a response with those status codes will not go to the client but will be forwarded to that JSP. (If you need to send those status codes, you can either adapt the web.xml or else explicitly return an error in the SystemError.jsp with a piece of Java code, e.g. <% response.sendError(500); %>.)

Note that in the case of JSPs, the call may have to be executed near the top of the JSP because once part of the body has been sent out over the line, so have the response headers and the status code.

- **parameter**: value

The numeric HTTP return code.

- **since**: 4.5.5

26.23 setSesSessionAttribute()

- **Signature**:

```
public void setSesSessionAttribute(String name, String usage, String path,  ↩
    String value)
```

Generic method for creating custom SES session attributes. This method is meant to provide absolute freedom in creating any type of SES session attribute, just in case the other existing methods won't support a specific feature or not work as expected. <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed).

- **parameter**: name

The directive name (such as *HGW_Host*).

- **parameter**: usage

The usage type (such as *gw-param*).

- **parameter**: path

The URI path for which to create the session attribute.

- **parameter**: value

The value for the session attribute.

- **since**: 4.0.18

26.24 setSesSessionAttribute()

- **Signature**:

```
public void setSesSessionAttribute(String name, String usage, String path,
String value, String encoding)
```

Generic method for creating custom SES session attributes. This method is meant to provide absolute freedom in creating any type of SES session attribute, just in case the other existing methods won't support a specific feature or not work as expected. <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed).

- **parameter**: name

The directive name (such as *HGW_Host*).

- **parameter**: usage

The usage type (such as *gw-param*).

- **parameter**: path

The URI path for which to create the session attribute.

- **parameter**: value

The value for the session attribute.

- **parameter**: encoding

The encoding; allowed values are *string* for plain, not-encoded text, *url* for URL-encoded text (default) and *base64* for binary data. NOTE: The global configuration property "session-attribute.encodings" must be set to "true", or encodings remain disabled (to ensure backwards compatibility with old HSP versions).

- **since**: 4.5.5

26.25 setSesSessionAttribute()

- **Signature**:

```
public void setSesSessionAttribute(String name, String usage, String path,
String value, String encoding, String timeout)
```

Generic method for creating custom SES session attributes. This method is meant to provide absolute freedom in creating any type of SES session attribute, just in case the other existing methods won't support a specific feature or not work as expected. <p> NOTE: Header propagation is only possible in case of a successfully completed model (the state "do.success" must be processed).

```
response.setSesSessionAttribute('admin','ac-app-az','/','allow','string','90')
```

- **parameter**: name

The directive name (such as *HGW_Host*).

- **parameter**: usage

The usage type (such as *gw-param*).

- **parameter**: path

The URI path for which to create the session attribute.

- **parameter**: value

The value for the session attribute.

- **parameter**: encoding

The encoding (allowed values are *string* for plain, not-encoded text, *url* for URL-encoded text (default) and *base64* for binary data.

- **parameter**: timeout

Can be used only for session attributes of usage type *ac-app-az*. Defines an idle timeout in seconds. If the timeout is reached, the attribute is removed from the session.

- **since**: 4.7.4

26.26 setWafSessionTimeout()

- **Signature**:

```
public void setWafSessionTimeout(String type, int seconds)
```

Sets or overrides a timeout setting of the WAF (HSP) at the successful end of the login (not at the moment this function is used). This variant of this function sets the timeout for all access areas (default). NOTE: Requires HSP (WAF) version 4.18.0.0 or newer!

- **parameter**: type

Must be one of the 3 values "final" (for "CredentialFinalTimeout"), "validity" (for "CredentialValidityPeriod") or "updateTrigger" (for "CredentialUpdateTrigger").

- **parameter**: seconds

The number of seconds for the timeout. NOTE: The HSP configuration defines the maximum and minimum values. If the value used here exceeds the maximum, or is lower than the minimum, the HSP will automatically adjust the value accordingly.

```
${response.setWafSessionTimeout('updateTrigger', 1800)}
```

- **since**: 5.9.0.0

26.27 setWafSessionTimeout()

- **Signature**:

```
public void setWafSessionTimeout(String type, String accessArea, int seconds)
```

Sets or overrides a timeout setting of the WAF (HSP) at the successful end of the login (not at the moment this function is used). NOTE: Requires HSP (WAF) version 4.18.0.0 or newer!

- **parameter**: type

Must be one of the 3 values "final" (for "CredentialFinalTimeout"), "validity" (for "CredentialValidityPeriod") or "updateTrigger" (for "CredentialUpdateTrigger").

- **parameter**: accessArea

If set to "Member" or "Customer", the timeout in question, selected by the "type" parameter, will be set only for the corresponding access area. If the parameter is set to any other value or null, the timeout will be set for all access areas (default).

- **parameter**: seconds

The number of seconds for the timeout. NOTE: The HSP configuration defines the maximum and minimum values. If the value used here exceeds the maximum, or is lower than the minimum, the HSP will automatically adjust the value accordingly.

```
${response.setWafSessionTimeout('updateTrigger', 'member', 1800)}
```

- **since**: 5.9.0.0

Chapter 27

runscript

Provides functions for executing external scripts or binaries.

27.1 groovy()

- **Signature**:

```
public String groovy(String filename)
```

Executes an external Groovy script. The filename can be either an absolute path of a text file containing a Groovy script, or a path relative to the SLS web application directory, such as <code>"WEB-INF/scripts/myscript.groovy"</code>.

- **parameter**: filename

The relative or absolute path of the Groovy script text file.

- **return**:

The evaluated Groovy script.

- **since**: 4.22.0

27.2 groovy()

- **Signature**:

```
public String groovy(String filename, boolean doEncode)
```

Executes an external Groovy script. The filename can be either an absolute path of a text file containing a Groovy script, or a path relative to the SLS web application directory, such as <code>"WEB-INF/scripts/myscript.groovy"</code>.

- **parameter**: filename

The relative or absolute path of the Groovy script text file.

- **parameter**: doEncode

True if the result should be URL encoded, false if not.

- **return**:

The evaluated Groovy script.

- **since**: 4.22.0

27.3 jexl()

- **Signature**:

```
public String jexl(String filename)
```

Executes an external JEXL script. The filename can be either an absolute path of a text file containing a JEXL script, or a path relative to the SLS web application directory, such as <code>"WEB-INF/scripts/myscript.jexl"</code>.

- **parameter**: filename

The relative or absolute path of the JEXL script text file.

- **return**:

The evaluated JEXL script.

- **since**: 4.1.0

27.4 jexl()

- **Signature**:

```
public String jexl(String filename, boolean doEncode)
```

Executes an external JEXL script. The filename can be either an absolute path of a text file containing a JEXL script, or a path relative to the SLS web application directory, such as <code>"WEB-INF/scripts/myscript.jexl"</code>.

- **parameter**: filename

The relative or absolute path of the JEXL script text file.

- **parameter**: doEncode

True if the result should be URL encoded, false if not.

- **return**:

The evaluated JEXL script.

- **since**: 4.1.0

Chapter 28

saml

General/core SAML functions.

28.1 createOpenSamlObject()

- **Signature**:

```
public Object createOpenSamlObject(Class clazz)
```

DEPRECATED: Use "saml.createOpenSamlObject(String)" instead.

@deprecated Use "saml.createOpenSamlObject(String)" instead. Create OpenSAML object for given class.

- **parameter**: clazz

The class.

- **return**:

OpenSAML object

```
${saml.createOpenSamlObject(Assertion.class)}
```

- **since**: 4.32.0

28.2 createOpenSamlObject()

- **Signature**:

```
public Object createOpenSamlObject(String className)
```

Create OpenSAML object for given class name, where the class name can be a simple class name like "Assertion" (preferred) for a class in the "core" package, or a fully qualified class name with package in the form "x.y.z.SomeSamlObject" (use only if cannot use the simple name).

Whenever possible use a simple class name like "Assertion" for optimal forward compatibility.

- **parameter**: className

The class name.

- **return**:

OpenSAML object

```
${saml.createOpenSamlObject("Assertion")}
```

- **since**: 5.11.0

28.3 toMultiLineString()

- **Signature**:

```
public String toMultiLineString(XMLObject samlObject)
```

Gets a multi-line XML string representation of the given SAML object.

- **parameter**: samlObject

OpenSAML object (org.opensaml.xml.XMLObject)

- **return**:

String representation if ok, exception string if an exception occurred, null if the given object was null

```
${saml.toMultiLineString(sp_assertion.getOpenSamlAssertion())}
```

- **since**: 5.11.0

28.4 toSingleLineString()

- **Signature**:

```
public String toSingleLineString(XMLObject samlObject)
```

Gets a single-line XML string representation of the given SAML object.

- **parameter**: samlObject

OpenSAML object (org.opensaml.xml.XMLObject)

- **return**:

String representation if ok, exception string if an exception occurred, null if the given object was null

```
${saml.toSingleLineString(sp_assertion.getOpenSamlAssertion())}
```

- **since**: 5.11.0

Chapter 29

session

Provides methods related to the session / user.

29.1 activateDebugLog()

- **Signature**:

```
public void activateDebugLog(boolean activate)
```

Activate or deactivate DEBUG logging for this login session.

- **parameter**: activate

True to activate debug logging, false to deactivate.

- **since**: 5.15.0.0

29.2 activateTraceLog()

- **Signature**:

```
public void activateTraceLog(boolean activate)
```

Activate or deactivate TRACE logging for this login session.

- **parameter**: activate

True to activate trace logging, false to deactivate.

- **since**: 5.15.0.0

29.3 cancelChallengeResponse()

- **Signature**:

```
public void cancelChallengeResponse()
```

Cancels a challenge/response that had been started with a *get.cred.challenge* action: Removes (if present) the expected challenge/response from the login session and the *challenge* JEXL/Groovy variable, as well as the challenge credential.

If you start a challenge/response that does not result in a verified challenge credential at the *do.success* action, login fails by default for security reasons (since SLS 5.10.0.0).

(In order to facilitate migrations, it is possible to deactivate this check by setting *sls.challenge.verification.mandatory=false* and then to look for warnings in the case where login would fail by default.)

- **since**: 5.10.0.0

29.4 clearAuthorizations()

- **Signature**:

```
public void clearAuthorizations()
```

Clears (removes) all authorizations that the user currently has.

- **since**: 4.0.17

29.5 clearCredentials()

- **Signature**:

```
public void clearCredentials()
```

Removes all credentials from the current session.

- **since**: 4.1.28

29.6 getAuthorizationValues()

- **Signature**:

```
public String[] getAuthorizationValues()
```

Get the authorisations of the current user, as an array of strings that can be used in a *foreach* loop.

- **return**: An array of strings containing all authorisations. The array may be empty (have zero entries) if there are no authorizations.
- **since**: 4.0.17

29.7 getAuthorizations()

- **Signature**:

```
public String getAuthorizations(String separator)
```

Get the authorisation-strings of the current user, separated by the delimiter.

- **parameter**: separator The string separating the different authorisations.
- **return**: A String containing all authorisations.
- **since**: 4.0.13

29.8 getAuthorizations()

- **Signature**:

```
public String[] getAuthorizations()
```

Get the authorisations of the current user (exactly the same as "getAuthorizationValues()").

- **return**: An array of strings containing all authorisations. The array may be empty (have zero entries) if there are no authorizations.
- **since**: 4.11.3

29.9 getCountry()

- **Signature**:

```
public String getCountry()
```

Returns the country code of the current locale of this session. This will either be the empty string or an uppercase ISO 3166 2-letter code.

- **return**:

The country code, such as "US", "CH" etc.

- **since**: 4.41.0.0

29.10 getCred()

- **Signature**:

```
public String getCred(String type)
```

To retrieve a credential.

- **parameter**: type

The semantic type of the credential.

- **return**:

The value of the credential or an empty string if it doesn't exist.

- **since**: 4.0.13

29.11 getFullModelState()

- **Signature**:

```
public String getFullModelState()
```

Returns the current model state, including the extension, if there is one (e.g. "do.generic-check" instead of only "do.generic").

- **return**:

The current state of the SLS model (corresponds to the state names as defined in the "sls.properties" file).

- **since**: 4.24.0

29.12 getLanguage()

- **Signature**:

```
public String getLanguage()
```

Returns the language code for this session, which will either be the empty string or a lowercase ISO 639 code.

- **return**:

The language code, such as "en", "de", "fr"...

- **since**: 4.41.0.0

29.13 getLanguageCode()

- **Signature**:

```
public String getLanguageCode()
```

Returns the language code for this session, which will either be the empty string or a lowercase ISO 639 code.

@deprecated use getLanguage() instead.

- **return**:

The language code, such as "en", "de", "fr"...

- **since**: 4.1.23

29.14 getLocale()

- **Signature**:

```
public Locale getLocale()
```

Returns the Java Locale object of the current SLS session. See Sun's javadoc for details about what methods are available:

@see

- **return**:

The Java Locale object.

- **since**: 4.1.23

29.15 getLocaleCode()

- **Signature**:

```
public String getLocaleCode()
```

Returns the country code of the current locale of this session. This will either be the empty string or an uppercase ISO 3166 2-letter code.

@deprecated use getCountry() instead.

- **return**:

The country code, such as "US", "CH" etc.

- **since**: 4.1.23

29.16 getMandator()

- **Signature**:

```
public String getMandator()
```

DEPRECATED: Use "session.getTenant()" instead.

- **return**:

the selected tenant (short ref) @deprecated Use "session.getTenant()" instead.

- **since**: 4.5.4

29.17 getMandatorSpecific()

- **Signature**:

```
public String getMandatorSpecific(String mappingstr)
```

DEPRECATED: Use "session.getTenantSpecific(String)" instead.

- **parameter**: mappingstr The string containing the mapping from a tenant to the value of the config property needed for this tenant.
- **return**:

The value of the mappingstr belonging to the currently selected tenant. @deprecated Use "Session.getTenantSpecific(String)" instead.

- **since**: 4.5.4

29.18 getMissingAuthorizationValues()

- **Signature**:

```
public String[] getMissingAuthorizationValues()
```

Get the list of missing authorisations (or sometimes called *roles*) that the user requires (as signaled by the HSP reverse proxy).

- **return**:

An array of strings containing all missing, required authorizations. The array may be empty (have zero entries) if there are no missing authorizations.

- **since**: 4.1.20

29.19 getMissingAuthorizations()

- **Signature**:

```
public String[] getMissingAuthorizations()
```

Get the list of missing authorisations (exactly the same like "getMissingAuthorizationValues()").

- **return**:

An array of strings containing all missing, required authorizations. The array may be empty (have zero entries) if there are no missing authorizations.

- **since**: 4.11.3

29.20 getModelInfo()

- **Signature**:

```
public String getModelInfo()
```

Returns the current name and state of the model.

- **return**:

The current model name and state. i.e: "login/get.cred"

- **since**: 4.0.19

29.21 getModelName()

- **Signature**:

```
public String getModelName()
```

Returns the current model name.

- **return**:

The name of the current SLS model (corresponds to the name as defined in the "sls.properties" file, i.e. "login" for model.login)

- **since**: 4.0.19

29.22 getModelState()

- **Signature**:

```
public String getModelState()
```

Returns the current model state.

- **return**:

The current state of the SLS model (corresponds to the state names as defined in the "sls.properties" file).

- **since**: 4.0.13

29.23 getModelStateParameter()

- **Signature**:

```
public String getModelStateParameter(String name)
```

Returns the value of a parameter of the current model state, if that parameter exists.

- **parameter**: name

The name of the parameter.

- **return**:

The value or null, if the parameter does not exist.

- **since**: 4.3.9

29.24 getModelStateProperty()

- **Signature**:

```
public String getModelStateProperty()
```

Returns the value of the property of the current model state, if the property exists.

- **return**:

The value or null, if the property does not exist.

- **since**: 4.3.9

29.25 getNumberOfTokens()

- **Signature**:

```
public int getNumberOfTokens()
```

Returns the number of tokens available to the user. For example, a user might have both a SecurID and a mobile phone that can be used for certain challenge / response processes.

- **return**:

The number of tokens. 0 is possible.

- **since**: 4.0.13

29.26 getObject()

- **Signature**:

```
public Object getObject(String key)
```

Returns any kind of object from the SLS session.

- **parameter**: key

The key under which the value is stored.

- **return**:

The value object, or null, if no value could be found for the given key.

- **since**: 4.3.0

29.27 getRedirectUrl()

- **Signature**:

```
public String getRedirectUrl(String url)
```

Returns a URL to redirect to once the authentication process is completed. The following values are used to build this URL, in the following order: <ol> "target" request parameter (always overrides!) the "sourceUrl" argument (if not null) "RequestedPage", if available "default.redirect" URL from configuration This method also makes sure that the URL is being formatted and URL-encoded correctly, and has the correct host/port information etc.

- **parameter**: url

The incoming URL, which may be a value from the configuration, or from a request parameter or header, such as the "RequestedPage" or "target". If the parameter is null, the configured default redirect URL will be used.

- **return**:

The proper, safe redirect URL.

- **since**: 4.0.17

29.28 getSelectedTokenType()

- **Signature**:

```
public String getSelectedTokenType()
```

Returns the type of the currently selected token. This is the string whole value depends on the implementation of the token type. Supported values are: <ul> optical mobile one_time_password

- **return**:

The token value type (empty, if no selected token was available).

- **since**: 4.0.13

29.29 getSelectionList()

- **Signature**:

```
public List<IHtmlSelectableWithMap> getSelectionList(String sessionKey)
```

Returns a list of the selectable entries that can be rendered as an HTML selection list by the SLS JSP "select" tag. Each entry in the list is an instance of "java.util.Map<String, Object>". This map typically holds two keys at least: <ul> "id" - the index number of the selectable entry. "alias" - the text for the entry. When used with the selection-list mechanism of the Webservice adapter, the following keys are also available: <ul> node - The XML node object (org.w3c.dom.Node); this is the raw data, but relatively cumbersome to use - usually it is much easier to use the gpath field instead. xml - The node object in text form, complete with XML declaration and Namespaces. Theoretically, this field can be null if creating the text from the node object failed, but in practice this should normally not be the case. Note that the text can span multiple lines, depending on the contents of the node object. gpath - The node object in form of a Groovy GPathResult object, which allows to handle the node very easily in Groovy expressions and scripts. Theoretically this field can be null, if creating the object failed, but in practice this should normally not be the case. See "SLS Administration Guide", chapter "Selection with <sls:selection> and do.select action" for more information.

- **parameter**: sessionKey

The session key for the list. For example, if the list is configured with the parameter "param.listkey=session.tokenlist", the session key is "tokenlist".

- **return**:

A list of selectable objects (or null, if there is none).

- **since**: 4.27.0.3

29.30 getSessionAttributeValue()

- **Signature**:

```
public String getSessionAttributeValue(String usage, String name, String path)
```

If a session attribute with the given parameters exist, this function returns its value. If no such attribute exists, null will be returned.

- **parameter**: usage

The session attribute usage ("aai-param", "gw-param", "ac-app-az" or "prop-header").

- **parameter**: name

The name of the header / parameter / AAI-variable etc.

- **parameter**: path

The path for which the attribute is valid.

- **return**:

The attribute value, or null.

- **since**: 4.7.9

29.31 getSlowdownTime()

- **Signature**:

```
public String getSlowdownTime()
```

Returns a string containing the amount of time a user must wait during a slow-down phase (which happens after several subsequent failed login attempts). <p> The time units for minutes, a single minute and seconds must be defined in the language resources files with the following keys: <ul> text.slowdown.time.minutes: Minutes text.slowdown.time.minute: A single minutes text.slowdown.time.seconds: Seconds text.slowdown.time.and: and

- **return**:

The localized waiting time text, such as "30 seconds".

- **since**: 4.0.19

29.32 getStringCred()

- **Signature**:

```
public String getStringCred(String name)
```

To retrieve a credential.

- **parameter**: name

The name of the string credential.

- **return**:

The value of the string credential or an empty string if it doesn't exist.

- **since**: 4.24.0

29.33 getTenant()

- **Signature**:

```
public String getTenant()
```

Get the currently selected tenant.

If the feature is disabled this method returns {@code null}. If the tenant is not (yet) selected this method returns the empty String.   Note: There exists also a Jexl-Variable <code>current.tenant</code>.

- **return**:

the selected tenant (short ref)

- **since**: 4.6.0

29.34 getTenantSpecific()

- **Signature**:

```
public String getTenantSpecific(String mappingstr)
```

This Jexl method allows tenant specific property setting. The mapping from tenant to the property value of this tenant, must be specified by the <code>mappingstr</code> in a well-defined syntax specified below. The idea is, that jexl evaluates the value of a config property dependent of the selected tenant.

A mapping could be for example (*mand1* to *propVal1*, *mand2* to *propVal2*), meaning: If mand1 is selected the property value evaluates to propVal1, or if mand2 is selected it evaluates to mand2.

Of course this jexl method is only useful if the multi-tenant feature is used. If it is called to early, before the tenant selection, no value can be returned.

This method is only designed for usage with the following config properties:

<pre> fake.token.list redirect.default requested.page.whitelist requested.page.blacklist redirect.app.logout </pre> <p> With other properties this method getTenantSpecific() does generally not work. <p> Syntax for the mappingstr:
 -It is a comma separated list, starting with a tenant, followed by the belonging value. This is repeated for each tenant. "<mand1>,<val1>,<mand2>,<val2>,. . . " <p> -Do not use any whitespaces around the commas. They will be interpreted as normal characters, and included into the result. <p> -If a comma is required in the resulting value, they can be escaped by a backslash "\," for example:

<pre> fake.token.list=session.getTenantSpecific(*mand1,securID\,mTan,mand2,mTan*) </pre>

- **parameter**: mappingstr The string containing the mapping from a tenant to the value of the config property needed for this tenant.
- **return**:

The value of the mappingstr belonging to the currently selected tenant.

- **since**: 4.6.0

29.35 getUserInfoValue()

- **Signature**:

```
public String getUserInfoValue(String key)
```

Get the value stored with the given key in the UserInfo properties that will be stored in the User-Info cookie, or {@code null} if there is no value stored under that key.

- **parameter**: key The key name.
- **return**:

The value of the key if found, {@code null} otherwise.

- **since**: 4.7.4

29.36 getUserInfoValue()

- **Signature**:

```
public String getUserInfoValue(String key, String defaultValue)
```

Get the value stored with the given key in the UserInfo properties that will be stored in the User-Info cookie, or the given default value if there is no value stored under that key.

- **parameter**: key The key name.
- **parameter**: defaultValue The default value to return if the key is not found.
- **return**:

The value of the key if found, default value otherwise.

- **since**: 4.17.0

29.37 getValue()

- **Signature**:

```
public String getValue(String key)
```

Returns a string value from the SLS session.

- **parameter**: key

The key under which the value is stored.

- **return**:

The string value, or an empty string, if no value could be found for the given key.

- **since**: 4.0.17

29.38 getVerifiedCred()

- **Signature**:

```
public String getVerifiedCred(String type)
```

To retrieve a verified credential. Verified credentials must have been checked as verified by an adapter.

- **parameter**: type The type of the credential (password, username...).
- **return**:

The value of the credential or null if it doesn't exist.

- **since**: 4.0.13

29.39 getVerifiedStringCred()

- **Signature**:

```
public String getVerifiedStringCred(String name)
```

To retrieve a verified string credential. Verified credentials must have been checked as verified by an adapter.

- **parameter**: name The name of the string credential.
- **return**:

The value of the credential or null if it doesn't exist.

- **since**: 4.24.0

29.40 hasAuthorization()

- **Signature**:

```
public boolean hasAuthorization(String authorization)
```

Determine if the current user has the given authorization (as defined by the "AC_RequireAz" directive in the location configuration of the HSP/SRM).

- **parameter**: authorization The authorization string to check for.
- **return**: True if the user has the given authorization, false otherwise.
- **since**: 4.13.1

29.41 hasCred()

- **Signature**:

```
public boolean hasCred(String type)
```

Checks if a certain credential is available.

- **parameter**: type

The semantic type of the credential.

- **return**:

True if the credential is available, false otherwise.

- **since**: 4.0.17

29.42 hasSessionAttribute()

- **Signature**:

```
public boolean hasSessionAttribute(String usage, String name, String path)
```

Checks if a certain SES session attribute exists for the current session, no matter what its value is.

- **parameter**: usage

The session attribute usage ("aai-param", "gw-param", "ac-app-az" or "prop-header").

- **parameter**: name

The name of the header / parameter / AAI-variable etc.

- **parameter**: path

The path for which the attribute is valid.

- **return**:

True if such an attribute exists in the current session.

- **since**: 4.6.0

29.43 hasSessionAttribute()

- **Signature**:

```
public boolean hasSessionAttribute(String usage, String name, String path,  ↩
    String value)
```

Checks if a certain SES session attribute exists for the current session.

- **parameter**: usage

The session attribute usage ("aai-param", "gw-param", "ac-app-az" or "prop-header").

- **parameter**: name

The name of the header / parameter / AAI-variable etc.

- **parameter**: path

The path for which the attribute is valid.

- **parameter**: value

The attribute value.

- **return**:

True if such an attribute exists in the current session.

- **since**: 4.6.0

29.44 hasStringCred()

- **Signature**:

```
public boolean hasStringCred(String name)
```

Checks if a certain string credential is available.

- **parameter**: name

The name of the string credential.

- **return**:

True if the string credential is available, false otherwise.

- **since**: 4.24.0

29.45 hasVerifiedCred()

- **Signature**:

```
public boolean hasVerifiedCred(String type)
```

Checks if a credential is verified (exactly the same as "isVerifiedCred(String)").

- **parameter**: type The type of the credential (password, username...).
- **return**:

"true" if it is verified "false" if not

- **since**: 4.11.3

29.46 hasVerifiedStringCred()

- **Signature**:

```
public boolean hasVerifiedStringCred(String name)
```

Checks if a string credential is verified (exactly the same as "isVerifiedStringCred(String)").

- **parameter**: name The name of the string credential.
- **return**:

"true" if it is verified "false" if not

- **since**: 4.24.0

29.47 invalidate()

- **Signature**:

```
public void invalidate()
```

Invalidates the SLS session.

- **since**: 4.0.17

29.48 isAuthenticated()

- **Signature**:

```
public boolean isAuthenticated()
```

Checks if the user has an authenticated HSP (SES) session. The function checks that (a) a verified username credential exists, and (b) that the "CredState" field of the HSP header "SessionInfo" has the value "valid".

- **return**:

True if the user has an authenticated session.

29.49 isVerifiedCred()

- **Signature**:

```
public boolean isVerifiedCred(String type)
```

Checks if a credential is verified. Verified credentials must have been checked as verified by an adapter.

- **parameter**: type The type of the credential (password, username...).
- **return**:

"true" if it is verified "false" if not

- **since**: 4.0.13

29.50 isVerifiedStringCred()

- **Signature**:

```
public boolean isVerifiedStringCred(String name)
```

Checks if a string credential is verified. Verified credentials must have been checked as verified by an adapter.

- **parameter**: name The name of the string credential.
- **return**:

"true" if it is verified "false" if not

- **since**: 4.24.0

29.51 markCredAsVerified()

- **Signature**:

```
public void markCredAsVerified(String type)
```

Marks a given credential as *verified*, so that the *do.success* model state will accept it as valid.

- **parameter**: type The type of the credential (password, username...).
- **since**: 4.9.1

29.52 markStringCredAsVerified()

- **Signature**:

```
public void markStringCredAsVerified(String name)
```

Marks a given string credential as *verified*, so that the *do.success* model state will accept it as valid.

- **parameter**: name The name of the string credential.
- **since**: 4.24.0

29.53 mustChangePassword()

- **Signature**:

```
public boolean mustChangePassword()
```

Allows to check if a previous operation indicated that the user has to change the password. Internally, this is signaled by a pseudo-credential of type "REQUIRENEWPWD" in the session. This function really just checks if that credential exists in the session.

- **return**:

True if the user must change the password.

- **since**: 4.25.0

29.54 mustInitSelectedToken()

- **Signature**:

```
public boolean mustInitSelectedToken()
```

Find out if a token must be initialized before it can be used normally. Some tokens need an initialization, that requires a special model flow.

- **return**:

true if there is a selected token with state initialize, false otherwise.

- **since**: 4.0.17

29.55 processNextGETasPOST()

- **Signature**:

```
public void processNextGETasPOST()
```

Usually, when the SLS receives a GET request, it doesn't really do anything, it stays in the current model state. A typical use case is that after some "do.something" action, a model state like "get.cred" is reached, which results in displaying a login page. If the browser sends a GET request now, the SLS won't do anything, remaining the same model state, and therefore just displaying the same page again. <p> However, there are some login processes (NTLM for example) which use multiple GET requests. While specific authentication adapters like the NTLM adapter make sure that those GET requests are handled correctly, this function allows to implement custom login or challenge / response processes that are based on GET requests only. <p> So, if this function is called within a model state action, the following request from the client will be treated like a POST request, even if it's a GET request.

- **since**: 4.7.3

29.56 removeCred()

- **Signature**:

```
public String removeCred(String type)
```

Removes a credential by type.

- **parameter**: type The semantic type of the credential.
- **return**:

The value of the removed credential, if it was a string credential;

29.57 removeStringCred()

- **Signature**:

```
public String removeStringCred(String name)
```

Removes a custom string credential by name. These are credentials of semantic type "STRING".

- **parameter**: name The name of the credential.
- **return**:

The value of the removed credential;

29.58 removeUserInfoValue()

- **Signature**:

```
public void removeUserInfoValue(String key)
```

Removes a key from the UserInfo cookie.

- **parameter**: key

The entry to remove.

- **since**: 4.8.6

29.59 removeValue()

- **Signature**:

```
public void removeValue(String key)
```

Removes a string value from the SLS session.

- **parameter**: key

The key under which the value is stored.

- **since**: 4.0.17

29.60 requiresAuthorization()

- **Signature**:

```
public boolean requiresAuthorization()
```

Allows to check if the user requires at least one authorization (as defined by the "AC_RequireAz" directive in the location configuration of the HSP/SRM) that was not yet granted. <p> This function could be used with the JEXL enabled JSP tags to display a message on the login page, notifying the user that a login is required because of a missing authorization.

- **return**:

True if the user lacks a required authorization.

- **since**: 4.1.20

29.61 requiresAuthorization()

- **Signature**:

```
public boolean requiresAuthorization(String authorization)
```

Allows to check if the user requires a specific authorization (as defined by the "AC_RequireAz" directive in the location configuration of the HSP/SRM) that was not yet granted. <p> This function could be used with the JEXL enabled JSP tags to display a message on the login page, notifying the user that a login is required because of a missing authorization.

- **parameter**: authorization The authorization string to check for.
- **return**:

{@code true} if the user lacks the given, required authorization, {@code false} otherwise.

- **since**: 4.1.20

29.62 resetAuditLogList()

- **Signature**:

```
public boolean resetAuditLogList()
```

Reset the list of already logged audit messages of this session. This is useful when a login restarts from scratch. I.e. when a multi-step login with enabled fakemode fails.

- **return**:

true if there where audit log messages in the list, false otherwise. @see SlsLogHelper#logOnlyOnce(IMessage)

- **since**: 4.3.9

29.63 resetLoginSlowdown()

- **Signature**:

```
public void resetLoginSlowdown()
```

Resets the login slowdown. More precisely, if slowdown is activated, removes the username credential if present and resets the internal slowdown state of the login session.

- **since**: 5.14.0.0

29.64 setAccessAreaLevel()

- **Signature**:

```
public void setAccessAreaLevel(String value)
```

Allows to override the "AccessArea" level for the SES session once the authentication is successfully completed. With this function used in a "do.generic" step in the model, it is possible to adjust the level according, for example, to the login path taken in the model (login with 2 or with 3 credentials). <p> NOTE: The value MUST be "Customer" or "Member", anything else will be ignored.

- **parameter**: value

The "AccessArea" level.

- **since**: 4.1.29

29.65 setCred()

- **Signature**:

```
public void setCred(String type, String value){
```

Sets a credential to a value. If the credential already exists it will be overwritten. The new credential will be marked as not verified.

- **parameter**: type The semantic type of the credential.
- **parameter**: value The value of the credential (must not be null).
- **since**: 4.0.14

29.66 setCredentialsFromBasicAuth()

- **Signature**:

```
public void setCredentialsFromBasicAuth(boolean failOnMissingCredentials)
```

Sets username and password credential from basic auth header of current request;

29.67 setLanguage()

- **Signature**:

```
public void setLanguage(String lang)
```

Allows to set the language for the user interface (which can also be set using the "lang" URL parameter; see "SLS Administration Guide" for details).

- **parameter**: lang

The two-letter language value, such as "en", "de", "it". NOTE: there must be a corresponding message resource file for the selected language, or this function will have no visible effect.

- **since**: 4.10.2

29.68 setObject()

- **Signature**:

```
public void setObject(String key, Object value)
```

Allows to store any kind of object in the SLS session.

- **parameter**: key

The key under which the value is to be stored.

- **parameter**: value

The object to store under that key.

- **since**: 5.4.0.1

29.69 setPasswordChangeRequired()

- **Signature**:

```
public void setPasswordChangeRequired()
```

Sets a pseudo-credential of type "REQUIRENEWPWD" in the session, in order to signal that the user has to change the password. After calling this function, "function.mustChangePassword()" will return "true".

- **since**: 4.25.0

29.70 setRequestedPage()

- **Signature**:

```
public void setRequestedPage(String url)
```

Sets the requested page in the current session to the given value. This means that the SLS will redirect to that URL after a successful login.

If the function "session.setUriFragmentIdentifier(String)" had been invoked, that URI fragment identifier will be added to the URL defined by the "url" parameter of this function in the final location header sent to the client.

- **parameter**: url

The URL to which a redirect should be performed.

- **since**: 4.1.16

29.71 setStringCred()

- **Signature**:

```
public void setStringCred(String name, String value){
```

Sets a string credential to a value. If the credential already exists it will be overwritten. The new credential will be marked as not verified.

- **parameter**: name The name of the string credential.
- **parameter**: value The value of the credential (must not be null).
- **since**: 4.24.0

29.72 setTenant()

- **Signature**:

```
public void setTenant(String tenant)
```

Allows to force the usage of a certain tenant. If the given tenant alias does not exist in the configuration, or multi-tenancy isn't configured and enabled at all, nothing will happen.

- **parameter**: tenant

The tenant that should be used.

29.73 setUriFragmentIdentifier()

- **Signature**:

```
public void setUriFragmentIdentifier(String uriFragmentIdentifier)
```

Sets the URI fragment identifier to be used with the requested page in the current session. This value will be appended to the current requested page URL 1:1.

This function will affect the redirects performed by the model states "do.success" and "do.redirect" (even if the requested page used by the "do.success" state was overridden by invoking "session.setRequestedPage(String url)").

- **parameter**: uriFragmentIdentifier

Allows to specify a URL fragment identifier which will be added to the requested page redirect URL 1:1.

- **since**: 5.17.0.0

29.74 setUserInfoValue()

- **Signature**:

```
public void setUserInfoValue(String key, String value)
```

Set a key value pair in the UserInfo properties object that will be stored in the User-Info cookie. Key and value have to be a non empty string.

- **parameter**: key The key name.
- **parameter**: value The value to be set.
- **since**: 4.7.4

29.75 setValue()

- **Signature**:

```
public void setValue(String key, String value)
```

Stores a string value in the SLS session.

- **parameter**: key

The key under which to store the value.

- **parameter**: value

The string value to store.

- **since**: 4.0.17

Chapter 30

sesticket

Once the SES ticket credential provider in the SLS processed a SES ticket, it is made available in the variable "sesticket", which implements the "ITicket" interface from the SES ticket API, with the following methods.

Chapter 31

sp

SAML Service Provider Adapter: Functions.

31.1 createRandomId()

- **Signature**:

```
public String createRandomId()
```

Create a random ID string, suitable e.g. as a transient NameID in SAML Assertions.

```
${sp.createRandomId()}
```

- **return**:

A random ID.

- **since**: 4.30.0

31.2 getAssertionAttributeByFriendlyName()

- **Signature**:

```
public Attribute getAssertionAttributeByFriendlyName(String friedlyName)
```

Gets attribute with given friendly name from given assertion.

```
${sp.getAssertionAttributeByFriendlyName(assertion,'uid')}
```

- **parameter**: friedlyName

attribute friendly name

- **return**:

first found attribute or null if not found

- **since**: 4.30.0

31.3 getAssertionAttributeByName()

- **Signature**:

```
public Attribute getAssertionAttributeByName(String name)
```

Gets attribute with given name from given assertion.

```
${sp.getAssertionAttributeByName(assertion,'urn:oid:0.9.2342.1920.100.1.1')}
```

- **parameter**: name

attribute name

- **return**:

first found attribute or null if not found

- **since**: 4.30.0

31.4 getAttributeStringValue()

- **Signature**:

```
public String getAttributeStringValue(Attribute attribute)
```

Gets (single) string value from given attribute.

```
${sp.getAttributeStringValue(attr)}
```

- **parameter**: attribute

attribute

- **return**:

first or only value or null if no values

- **since**: 4.30.0

31.5 getAttributeStringValues()

- **Signature**:

```
public String[] getAttributeStringValues(Attribute attribute)
```

Gets all string values from given attribute.

```
${sp.getAttributeStringValues(attr)}
```

- **parameter**: attribute

attribute

- **return**:

values

- **since**: 4.30.0

31.6 getIdpInfo()

- **Signature**:

```
public IIdpInfo getIdpInfo(String aliasOrEntityId)
```

Get info about the configured IdP with given EntityID or alias.

```
${sp.getIdpInfo('acme-idp')}
```

```
${sp.getIdpInfo('acme-idp')}
${sp.getIdpInfo('https://idp.acme.org/idp/')}
```

- **parameter**: aliasOrEntityId

IdP alias or EntityID

- **return**:

info or null if no IdP with given alias or EntityID

- **since**: 4.39.0

31.7 getIdpInfos()

- **Signature**:

```
public IIdpInfo[] getIdpInfos()
```

Get info about all configured IdPs as an array in which each element represents an IdP info object.

Typically used in a JSP for displaying a selection of IdPs to choose from for login.

The interface IIdpInfo has the following getters: <ul> String getEntityId() String getAlias() String getIdpLoginUrlAtSp() int getIndex()

```
${sp.getIdpInfos().0.getIdpLoginUrlAtSp()}
```

- **return**:

Array of IdP info.

- **since**: 4.30.0

31.8 getMetadata()

- **Signature**:

```
public String getMetadata()
```

Get SAML 2.0 SP Metadata as a multi-line XML string, useful e.g. for displaying in a JSP.

Uses the key pair alias as defined in config properties. If a future key pair alias is defined via config property, the corresponding certificate is additionally included.

```
${sp.getMetadata()}
```

- **return**:

Metadata or empty string if could not get metadata.

- **since**: 4.30.0

31.9 getMetadata()

- **Signature**:

```
public String getMetadata(String keyPairAlias)
```

Get SAML 2.0 SP Metadata as a multi-line XML string, using the given key pair alias to retrieve the certificate.

If a future key pair alias is defined via config property, the corresponding certificate is additionally included.

```
${sp.getMetadata('sp')}
```

- **parameter**: keyPairAlias

The key pair alias to use for retrieving the SP certificate.

- **return**:

Metadata or empty string if could not get metadata.

- **since**: 4.37.0

31.10 getSamlMessage()

- **Signature**:

```
public SAMLObject getSamlMessage()
```

Get the SAML message last received by the SP with any previous HTTP request during the current login session.

Use e.g. for free handling of SAML messages in the login model, by directly accessing the OpenSAML 2.0 API, which directly mirrors the XML structure of the SAML message.

Implementation note and interaction with idp.getSamlMessage(): If sp.getSamlMessage() is called before do.saml.sp.handlemsg, the message is retrieved from a SAML credential created in the login session when receiving the HTTP request;

31.11 getSamlMessageAgeSecs()

- **Signature**:

```
public int getSamlMessageAgeSecs()
```

Determine the age in seconds of the last SAML message received with any previous HTTP request during the current login session. More precisely, this calculates the time difference in seconds between now and the time indicated in the IssueInstant attribute of the SAML message.

```
${sp.getSamlMessageAgeSecs()}
```

- **return**:

age in seconds or 0 if could not determine (no SAML message or no IssueInstant in SAML message)

- **since**: 4.30.0

31.12 getSamlMessageBinding()

- **Signature**:

```
public String getSamlMessageBinding()
```

Get the binding with which the last SAML message was received by the SP with any previous HTTP request during the current login session.

Possible return values include "Redirect" and "POST".

```
${sp.getSamlMessageBinding()}
```

- **return**:

SAML binding as string if any SAML message has been received, null otherwise

- **since**: 4.39.0

31.13 getSamlMessageIssuer()

- **Signature**:

```
public String getSamlMessageIssuer()
```

Get the issuer of the last SAML message received with any previous HTTP request during the current login session. More precisely, this gets the value of the Issuer attribute in the SAML message, i.e. the EntityID of the issuer.

```
${sp.getSamlMessageIssuer()}
```

- **return**:

issuer or null if could not determine (no SAML message or no Issuer in SAML message)

- **since**: 4.30.0

31.14 getSamlMessageIssuerAlias()

- **Signature**:

```
public String getSamlMessageIssuerAlias()
```

Get the IdP alias of the issuer of the last SAML message received with any previous HTTP request during the current login session. More precisely, this gets the value of the Issuer attribute in the SAML message, i.e. the EntityID of the issuer and maps it to the corresponding SP alias, defaulting to the EntiyID if no SP alias has been defined in the configuration or no IdP corresponds to the EntityID.

```
${sp.getSamlMessageIssuerAlias()}
```

- **return**:

alias or null if could not determine (no SAML message or no Issuer in SAML message)

- **since**: 4.30.0

31.15 hasSamlMessage()

- **Signature**:

```
public boolean hasSamlMessage()
```

Determine if the SP has received a SAML message with any previous HTTP request during the current login session.

```
${sp.hasSamlMessage()}
```

- **return**:

true if got a SAML message, false otherwise

- **since**: 4.30.0

31.16 isKnownIdp()

- **Signature**:

```
public boolean isKnownIdp(String aliasOrEntityId)
```

Determines if there is a configured IdP with given alias or EntityID.

```
${sp.isKnownIdp('acme-idp')}
```

```
${sp.isKnownIdp('acme-idp')}
${sp.isKnownIdp('https://idp.acme.org/idp/')}
```

- **parameter**: aliasOrEntityId

IdP alias or EntityID

- **return**:

true or false

- **since**: 4.30.0

31.17 isMessageLogoutRequest()

- **Signature**:

```
public boolean isMessageLogoutRequest()
```

Determine if the has received a SAML LogoutRequest message with any previous HTTP request during the current login session.

```
${idp.isMessageLogoutRequest()}
```

- **return**:

true if got LogoutRequest, false otherwise

- **since**: 5.5.0.0

31.18 isMessageResponseAssertion()

- **Signature**:

```
public boolean isMessageResponseAssertion()
```

Determine if the IdP has received a SAML Response message, i.e. a "login" response message with optionally (if login was successful) a SAML Assertin within, with any previous HTTP request during the current login session.

```
${idp.isMessageResponseAssertion()}
```

- **return**:

true if got a Response, false otherwise

- **since**: 5.5.0.0

31.19 selectIdp()

- **Signature**:

```
public void selectIdp(String entityIdOrAlias)
```

Allows to select the IdP to be used for the login dynamically. Invoke this function before the model state "do.saml.idp.sendmsg" in order to ensure that the request will be sent to that Identity Provider.

- **parameter**: entityIdOrAlias

Must be either the Entity ID of the IdP, or the alias of that IdP as defined in the property configuration.

```
${sp.selectIdp('acme-idp')}
```

- **since**: 4.32.0

Chapter 32

sp_assertion

SAML Service Provider Adapter: Script (JEXL/Groovy) wrapper for the SAML assertion sent from the IdP. This wrapper is used to access the assertion before the login is completed. <p> NOTE: Before any of these functions can be used, the model state "do.saml.sp.handlemsg" must have been invoked. Otherwise, no assertion object exists in the session.

32.1 getAssertionAttribute()

- **Signature**:

```
public AttributeWrapper getAssertionAttribute(String name) throws SLSException
```

Returns the first found attribute with the given name, from all AuthnStatements in the Assertion.

- **parameter**: name

The full/long name of the attribute, typically a URN.

- **return**:

The attribute or null if not found. @throws SLSException If could not marshal or unmarshal assertion attributes.

- **since**: 4.32.0

32.2 getAssertionAttributeByFriendlyName()

- **Signature**:

```
public AttributeWrapper getAssertionAttributeByFriendlyName(String ↩
    friendlyName) throws SLSException
```

Returns the first found attribute with the given friendly name, from all AuthnStatements in the Assertion.

- **parameter**: friendlyName

The friendly of the attribute.

- **return**:

The attribute or null if not found. @throws SLSException If could not marshal or unmarshal assertion attributes.

- **since**: 4.32.0

32.3 getAssertionAttributes()

- **Signature**:

```
public List<AttributeWrapper> getAssertionAttributes() throws SLSException
```

Returns a list of all attributes from all AuthnStatements in the Assertion. See prefix "apidoc_saml_attribute" for available methods on the objects in the list.

- **return**:

List of all attributes (never null, may be empty). @throws SLSException If could not marshal or unmarshal assertion attributes.

- **since**: 4.32.0

32.4 getAuthenticationMethod()

- **Signature**:

```
public String getAuthenticationMethod()
```

Returns the authentication method in the AuthnContext of the first AuthnStatement in the Assertion.

- **return**:

The authentication method or null if not found in assertion.

32.5 getOpenSamlAssertion()

- **Signature**:

```
public Assertion getOpenSamlAssertion()
```

Returns a reference to the OpenSAML-API object instance which represents the assertion that was received from an Identity Provider. At this point, the assertion has already been verified.

- **return**:

An instance of the class "org.opensaml.saml2.core.Assertion".

- **since**: 4.30.0

Chapter 33

sp_authn_request

SAML Service Provider Adapter: Script (JEXL/Groovy) wrapper for the SAML AuthnRequest created by the SP. This wrapper is used to access the SP request before it is sent to the IdP. <p> NOTE: Before any of these functions can be used, the model state "do.saml.sp.createmsg" must have been invoked. Otherwise, no request object exists in the session.

33.1 addEncryptedData()

- **Signature**:

```
public void addEncryptedData(String alias, byte[] data)
```

Allows to encrypt and piggy-back custom data into the authentication request. This custom data can then be decrypted and extraced by an SLS Identity Provider using the corresponding JEXL funtion "getDecryptedData(String alias)". This function encrypts the given data and stores it as a Base64 string in the otherwise unused "ProviderName" field.

- **parameter**: alias

The alias for the "crypto." properties group.

- **parameter**: data

The plain data to encrypt.

33.2 addToIdpList()

- **Signature**:

```
public void addToIdpList(String providerID, String name, String loc)
```

Adds an "IDPList" element to the current "AuthnRequest", if none exists yet, or uses the existing one. Adds a new "IDPEntry" instance to that list with the given values.

- **parameter**: providerID

The entity ID of the IDP (mandatory, must correspond to entity ID in IdP metadata).

- **parameter**: name

Optional (may be null or empty): A human readable name for the IdP.

- **parameter**: loc

Optional (may be null or empty): The authentication URL to be used at the IdP.

- **since**: 4.30.0

33.3 addToIdpList()

- **Signature**:

```
public void addToIdpList(String providerID)
```

Adds an "IDPList" element to the current "AuthnRequest", if none exists yet, or uses the existing one. Adds a new "IDPEntry" instance to that list with the given values.

- **parameter**: providerID

The entity ID of the IDP (mandatory, must correspond to entity ID in IdP metadata).

- **since**: 4.32.0

33.4 addToIdpList()

- **Signature**:

```
public void addToIdpList(IDPEntry entry)
```

Adds an "IDPList" element to the current "AuthnRequest", if none exists yet, or uses the existing one. Adds a new "IDPEntry" instance to that list with the given value.

- **parameter**: entry

The IDPEntry to add.

- **since**: 4.32.0

33.5 addToIdpList()

- **Signature**:

```
public void addToIdpList(List<IDPEntry> entries)
```

Adds an "IDPList" element to the current "AuthnRequest", if none exists yet, or uses the existing one. Adds new "IDPEntry" instances to that list with the given values.

- **parameter**: entries

The IDPEntries to add.

- **since**: 4.32.0

33.6 getOpenSamlRequest()

- **Signature**:

```
public AuthnRequest getOpenSamlRequest()
```

Returns a reference to the OpenSAML-API object instance which represents the authentication request that will be sent to the IdP. At this point, the request is not yet signed (this will happen automatically before it is actually sent, during the model state "do.saml.sp.sendmsg").

- **return**:

An instance of the class "org.opensaml.saml2.core.AuthnRequest".

- **since**: 4.30.0

33.7 setProxyCount()

- **Signature**:

```
public void setProxyCount(int value)
```

Sets the numeric "ProxyCount" element in the "Scoping" node of this AuthnRequest. This counter allows to restrict proxying to the given number of proxy instances.

- **parameter**: value

The proxy count value (must be > than zero).

Chapter 34

spnego

SPNEGO Adapter: Functions.

34.1 traceIncomingSpnegoToken()

- **Signature**:

```
public void traceIncomingSpnegoToken(String inTokenBase64)
```

Traces the same detailed information about token parsing and decryption (plus extraction of MS PAC data if any) of a given incoming SPNEGO Token that is traced after successful token verification by the JDK at login, but can be called already before that verification in order to maybe more easily analyze the reason(s) why JDK verification would fail.

Only traces anything if trace log is active.

- **parameter**: inTokenBase64

The base64-encoded incoming SPNEGO Token, typically taken from the Authorization HTTP request header by removing the leadding "Negotiate " prefix.

- **since**: 5.16.0.0

34.2 traceSubject()

- **Signature**:

```
public void traceSubject(boolean withSecrets)
```

Traces the SLS Subject in detail, optionally including secrets; what is traced corresponds essentially to the configured keytabs.

Note that at SLS startup this information is also traced (without secrets).

Only traces anything if trace log is active.

- **parameter**: withSecrets

If true also log secrets (keys), otherwise not.

- **since**: 5.16.0.0

Chapter 35

webauthn

WebAuthn Adapter: Functions.

35.1 clearMemoryStore()

- **Signature**:

```
public void clearMemoryStore()
```

If using the memory store for testing / initial integration this method can be used to clear the store of all stored credentials.

- **since**: 5.15.0.0

35.2 deleteStoredCredential()

- **Signature**:

```
public void deleteStoredCredential(String credentialId)
```

Deletes the credential with the given ID, but only if the credential is for the current user (username from the verified username credential) and for the current RP ID (config property *webauthn.rp.id*) in order to prevent users from deleting credentials they do not own.

- **parameter**: credentialId

Credential ID, must not be null

- **since**: 5.15.0.0

35.3 exportMemoryStore()

- **Signature**:

```
public String exportMemoryStore()
```

If using the memory store for testing / initial integration this method can be used to export the store contents to a string formatted as a multi-line JSON string - note that this is explicitly intended for test setups and initial integration, as there is no guarantee that exported data could be reimported in future SLS versions, nor is good performance with a large number of stored credentials an explicit goal.

- **return**:

The memory store contents exported as multi-line JSON string suitable for import with the importMemoryStore() function.

35.4 getPublicKeyJavascript()

- **Signature**:

```
public String getPublicKeyJavascript()
```

Use in WebAuthn JSPs to get Javascript for the publicKey: Gets the public key rendered from JSON to a Javascript string.

Depending on the flow this returns the public key for registration or authentication.

- **return**:

Javascript string

- **since**: 5.15.0.0

35.5 getPublicKeyJson()

- **Signature**:

```
public Map<String,Object> getPublicKeyJson()
```

Use after the do.webauthn.createmsg action to make custom changes to the public key - after the do.webauth.create model action: Gets the public key as a Map as parsed by Groovy's JsonSlurper for easy manipulation - you can just use dots "." to access members down the JSON hierarchy and modify as needed. See also the chapter "Groovy Scripting" in the SLS Admin Guide for some samples how to elegantly modify the returned map.

Depending on the flow this returns the public key for registration or authentication.

Note that byte arrays must be stored as base64url-encoded strings prefixed with "Uint8Array:".

```
#{challenge = webauthn.getPublicKeyJson().challenge}
```

- **return**:

Map with string keys and arbitrary object values (incl. nested maps)

- **since**: 5.15.0.0

35.6 getReceivedMessageClientDataJson()

- **Signature**:

```
public Map<String,Object> getReceivedMessageClientDataJson()
```

Convenience function to get the response.clientDataJSON field of the received message already base64url-decoded and parsed to a map.

Note the map is read-only in the sense that modifying it has no effect on the overall received message; but you could still modify the returned value, encode it to base64url and set in the message JSON at credential.clientDataJSON.

- **return**:

clientDataJSON map or null if no received message or could not parse it

- **since**: 5.15.0.0

35.7 getReceivedMessageFlow()

- **Signature**:

```
public String getReceivedMessageFlow()
```

Returns the flow naturally associated with the received message, i.e. "registration" or "authentication".

- **return**:

Flow as string or null if no received message or could not parse it

- **since**: 5.15.0.0

35.8 getReceivedMessageJson()

- **Signature**:

```
public Map<String,Object> getReceivedMessageJson()
```

Use in login model to make custom changes to the received WebAuthn message - after displaying the registration/authentication JSP and getting the posted JSON message, and before the do.webauthn.handlemsg model action: Gets the received message as a Map as parsed by Groovy's JsonSlurper for easy manipulation - you can just use dots "." to access members down the JSON hierarchy and modify as needed.

- **return**:

Map with string keys and arbitrary object values (incl. nested maps) or null if no received message

- **since**: 5.15.0.0

35.9 getStoredCredentials()

- **Signature**:

```
public List<WebAuthnScriptCredentialData> getStoredCredentials()
```

Returns a list of all credentials registered in the store to the current user (username from the verified username credential) and the current RP ID (config property *webauthn.rp.id*).

If there is no current user (i.e. no verified username credential), an empty list is returned, as well as when there are no registered credentials.

The credential data contains two fields, *credentialFriendlyName* (to display to the user) and *credentialId* (to later delete a credential if desired by the user).

- **return**:

List of credential data, never null but may be empty

- **since**: 5.15.0.0

35.10 importMemoryStore()

- **Signature**:

```
public void importMemoryStore(String store)
```

If using the memory store for testing / initial integration this method can be used to import the store contents as a string formatted as a multi-line JSON string - note that this is explicitly intended for test setups and initial integration, as there is no guarantee that exported data could reimported in future SLS versions, nor is good performance with a large number of stored credentials an explicit goal.

- **parameter**: store

exported store (JSON string)

- **since**: 5.15.0.0

Chapter 36

webauthn_message

WebAuthn Adapter: Script (JEXL/Groovy) wrapper for the WebAuthn message created by the RP. This wrapper is used to read internals of the message after successful registration/authentication. <p> NOTE: Before any of these functions can be used, the model state "do.webauthn.handlemsg" must have been invoked successfully. Otherwise, no message object exists in the session.

36.1 getDataObject()

- **Signature**:

```
public Object getDataObject()
```

Returns a reference to the WebAuthn4j object instance which represents the data extracted from the validated WebAuthn message, either an instance of RegistrationData or AuthenticationData.

Only returns non-null if called after successfully validating the message in the model state "do.webauthn.handlemsg".

Note
Since this returns an object from a 3rd party library, backwards compatibility cannot be guaranteed, future versions of the WebAuthn4j library may behave differently. If you frequently need to obtain data using these WebAuthn4j objects, please consider notifying USP so that eventually a more specific function could be provided for that use case.

- **return**:

An instance of the class "com.webauthn4j.data.RegistrationData" or the class "com.webauthn4j.data.AuthenticationData", or null if "do.webauthn.handlemsg" has not run or not successfully validated the message.

- **since**: 5.15.0.0

36.2 isAttestedCredentialDataIncluded()

- **Signature**:

```
public boolean isAttestedCredentialDataIncluded()
```

Returns true if there is a message with authenticator data and the flag AT (Attested Credential Data Included) is true, false in all other cases.

- **return**:

true or false

- **since**: 5.15.0.0

36.3 isExtensionDataIncluded()

- **Signature**:

```
public boolean isExtensionDataIncluded()
```

Returns true if there is a message with authenticator data and the flag ED (Extension Data Included) is true, false in all other cases.

- **return**:

true or false

- **since**: 5.15.0.0

36.4 isUserPresent()

- **Signature**:

```
public boolean isUserPresent()
```

Returns true if there is a message with authenticator data and the flag UP (User Present) is true, false in all other cases.

- **return**:

true or false

- **since**: 5.15.0.0

36.5 isUserVerified()

- **Signature**:

```
public boolean isUserVerified()
```

Returns true if there is a message with authenticator data and the flag UV (User Verified) is true, false in all other cases.

- **return**:

true or false

- **since**: 5.15.0.0

