

---

# Flask SAML2 Documentation

*Release 0.2.0*

**Tidetech**

**Aug 07, 2019**



## CONTENTS:

<b>1</b>	<b>Installing</b>	<b>3</b>
1.1	Dependencies . . . . .	3
<b>2</b>	<b>Examples</b>	<b>5</b>
<b>3</b>	<b>Identity providers</b>	<b>7</b>
3.1	IdentityProvider . . . . .	7
3.2	SPHandler . . . . .	9
3.3	Configuration . . . . .	11
<b>4</b>	<b>Service providers</b>	<b>13</b>
4.1	ServiceProvider . . . . .	13
4.2	IdPHandler . . . . .	16
4.3	Configuration . . . . .	17
4.4	Example . . . . .	18
<b>5</b>	<b>Exceptions</b>	<b>19</b>
<b>6</b>	<b>Internal modules</b>	<b>21</b>
6.1	Encoding and decoding . . . . .	21
6.2	Signing and digest tools . . . . .	21
6.3	XML tools . . . . .	24
6.4	Utilities . . . . .	25
<b>7</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



Flask SAML2 helps you build Identity Providers and Service Providers (SP) for your applications. Each application will likely be either an Identity Provider (IdP) or a Service Provider (SP), so follow along with the relevant sections of the documentation for the particular needs of your application.



## INSTALLING

Install `flask-saml2` using `pip`:

```
$ pip install flask-saml2
```

### 1.1 Dependencies

`flask-saml2` relies on some libraries that have external dependencies. These external dependencies must be installed before `flask_saml2` and its dependencies can be installed.

#### 1.1.1 OpenSSL

`flask-saml2` relies on the `pyopenssl` library, which requires the `openssl` library to be installed. Please consult the documentation on [installing pyopenssl](#) for installation requirements.

#### 1.1.2 lxml

`flask-saml2` relies on `lxml`. Please consult the [Installing lxml](#) and install all of the external dependencies for `lxml` before installing `flask-saml2`.





## **EXAMPLES**

The `flask_saml2` repository comes with an example implementation of an Identity Provider and a Service Provider, configured to work with one another.

To run the example implementation, clone the `flask_saml2` repository and follow the instructions in the README.

The example uses a hard coded list of users in the Identity Provider. A real implementation would most likely use an external user database, with authentication perhaps managed by [Flask-Login](#).



## IDENTITY PROVIDERS

When users need to authenticate themselves with a Service Provider (SP), the SP will redirect the user to an Identity Provider (IdP). The users will authenticate with the Identity Provider, and will be redirected back to the Identity Provider with a payload that identifies the user.

Flask SAML2 implements all parts of the IdP workflow, except for authenticating your users against your user database (or however your users are managed). Developers should create an *IdentityProvider* subclass for their application that integrates with some other form of authentication, such as *Flask-Login*. Once a user is authenticated with the IdP, relevant user details will be composed into a payload which will be sent via the users browser back to the SP.

The method *IdentityProvider.create\_blueprint()* generates a Flask *Blueprint*, which needs to be registered in your application via *app.register\_blueprint(idp.create\_blueprint())*.

Any Service Providers the IdP handles need to be registered as well. These will be instances of *SPHandler*.

An functional example IdP and Flask application that uses a static list of users can be found in the *examples/* directory of the repository.

### 3.1 IdentityProvider

#### **class** flask\_saml2.idp.IdentityProvider

Developers should subclass *IdentityProvider* and provide methods to interoperate with their specific environment. All user interactions are performed through methods on this class.

Every subclass should implement *is\_user\_logged\_in()*, *login\_required()*, *logout()*, and *get\_current\_user()* as a minimum. Other methods can be overridden as required.

#### **get\_idp\_config()**

Get the configuration for this IdP. Defaults to *SAML2\_IDP* from *flask.Flask.config*. The configuration should be a dict like:

```
{
    # Should the IdP automatically redirect the user back to the
    # Service Provider once authenticated.
    'autosubmit': True,
    # The X509 certificate and private key this IdP uses to
    # encrypt, validate, and sign payloads.
    'certificate': ...,
    'private_key': ...,
}
```

To load the *certificate* and *private\_key* values, see

- *certificate\_from\_string()*

- `certificate_from_file()`
- `private_key_from_string()`
- `private_key_from_file()`

**Return type** `dict`

**`get_idp_entity_id()`**

The unique identifier for this Identity Provider. By default, this uses the metadata URL for this IdP.

See `get_metadata_url()`.

**Return type** `str`

**`get_idp_certificate()`**

Get the public certificate for this IdP. If this IdP does not sign its requests, returns None.

**Return type** `Optional[X509]`

**`get_idp_private_key()`**

Get the private key for this IdP. If this IdP does not sign its requests, returns None.

**Return type** `Optional[PKey]`

**`get_idp_autosubmit()`**

Should the IdP autosubmit responses to the Service Provider?

**Return type** `bool`

**`get_idp_signer()`**

Get the signing algorithm used by this IdP.

**Return type** `Optional[Signer]`

**`get_idp_digester()`**

Get the method used to compute digests for the IdP.

**Return type** `Digester`

**`get_service_providers()`**

Get an iterable of service provider config dicts. `config` should be a dict specifying a SPHandler subclass and optionally any constructor arguments:

```
>>> list(idp.get_service_providers())
[{'CLASS': 'my_app.service_providers.MySPSPHandler',
  'OPTIONS': {'acs_url': 'https://service.example.com/auth/acs/'},
}]
```

Defaults to `current_app.config['SAML2_SERVICE_PROVIDERS']`.

**Return type** `Iterable[Tuple[str, dict]]`

**`get_sso_url()`**

Get the URL for the Single Sign On endpoint for this IdP.

**`get_slo_url()`**

Get the URL for the Single Log Out endpoint for this IdP.

**`get_metadata_url()`**

Get the URL for the metadata XML document for this IdP.

#### **login\_required()**

Check if a user is currently logged in to this session, and `flask.abort()` with a redirect to the login page if not. It is suggested to use `is_user_logged_in()`.

#### **is\_user\_logged\_in()**

Return True if a user is currently logged in. Subclasses should implement this method

**Return type** `bool`

#### **logout()**

Terminate the session for a logged in user. Subclasses should implement this method.

#### **get\_current\_user()**

Get the user that is currently logged in.

**Return type** `~User`

#### **get\_user\_nameid(user, attribute)**

Get the requested name or identifier from the user. `attribute` will be a `urn:oasis:names:tc:SAML:2.0:nameid-format-style` urn.

Subclasses can override this to allow more attributes to be extracted. By default, only email addresses are extracted using `get_user_email()`.

#### **get\_user\_email(user)**

Get the email address for a user.

#### **get\_sp\_handlers()**

Get the SPHandler for each service provider defined.

**Return type** `Iterable[SPHandler]`

#### **render\_template(template, \*\*context)**

Render an HTML template. This method can be overridden to inject more context variables if required.

**Return type** `str`

#### **get\_metadata\_context()**

Get any extra context for the metadata template. Suggested extra context variables include 'org' and 'contacts'.

**Return type** `dict`

#### **is\_valid\_redirect(url)**

Check if a URL is a valid and safe URL to redirect to, according to any of the SPHandlers. Only used from the non-standard logout page, for non-compliant Service Providers such as Salesforce.

**Return type** `bool`

#### **create\_blueprint()**

Create a blueprint for this IdP. This blueprint needs to be registered with a Flask application to expose the IdP functionality.

## 3.2 SPHandler

An `flask_saml2.idp.IdentityProvider` handles requests from Service Providers via `flask_saml2.idp.SPHandler` instances.

See [Configuration](#) for configuration options.

**class** flask\_saml2.idp.SPHandler(*idp*, \*, *entity\_id*, *acs\_url=None*, *certificate=None*, *display\_name=None*)

Handles incoming SAML requests from a specific Service Provider for a running Identity Provider.

Sub-classes should provide Service Provider-specific functionality.

**assertion\_template**

alias of flask\_saml2.idp.xml\_templates.AssertionTemplate

**response\_template**

alias of flask\_saml2.idp.xml\_templates.ResponseTemplate

**build\_assertion**(*request*, *issue\_instant*)

Build parameters for the assertion template.

**Return type** dict

**build\_response**(*request*, *issue\_instant*)

Build parameters for the response template.

**Return type** dict

**encode\_response**(*response*)

Encodes the response XML template suitable for sending to the SP.

**format\_assertion**(*assertion\_params*)

Make a AssertionTemplate to respond to this SP.

**Return type** XmlTemplate

**format\_response**(*response\_params*, *assertion*)

Make a ResponseTemplate to respond to this SP.

**Return type** XmlTemplate

**get\_assertion\_id**()

Generates an ID for this assertion.

**get\_audience**(*request*)

Gets the audience assertion parameter from the request data.

**Return type** str

**get\_response\_id**()

Generate an ID for the response.

**get\_response\_context**(*request*, *response*, *relay\_state*)

Make a dictionary of parameters for the response template.

**get\_subject**()

Get the subject of the assertion, based on the currently authenticated user and SPHandler.subject\_format.

**extract\_request\_parameters**(*request*)

Fetches various parameters from the request into a dict.

**Return type** dict

**validate\_request**(*request*)

Validates the SAML request against the configuration of this Service Provider handler . Sub-classes should override this and raise a *CannotHandleAssertion* exception if the validation fails.

**Raises:**

**CannotHandleAssertion:** if the ACS URL specified in the SAML request doesn't match the one specified in the SP handler config.

**validate\_user()**

Validates the User. Sub-classes should override this and throw a `CannotHandleAssertion` exception if the validation does not succeed.

**decode\_saml\_string(saml\_string)**

Decode an incoming SAMLRequest into an XML string.

**Return type** `bytes`

**parse\_authn\_request(saml\_request)**

Get a `AuthnRequestParser` to handle this request.

**Return type** `AuthnRequestParser`

**parse\_logout\_request(saml\_request)**

Get a `LogoutRequestParser` to handle this request.

**Return type** `LogoutRequestParser`

**make\_response(request)**

Process the request and make a `ResponseTemplate`.

**Return type** `XmlTemplate`

**is\_valid\_redirect(url)**

Is this URL a valid redirect target back to this service provider?

### 3.2.1 Specific implementations

Some handlers for common Service Providers have been bundled with this project:

```
class flask_saml2.idp.sp.salesforce.SalesforceSPHandler(idp, *, entity_id,
                                                         acs_url=None, certificate=None, display_name=None)
```

Salesforce.com SPHandler implementation.

```
class flask_saml2.idp.sp.google_apps.GoogleAppsSPHandler(idp, *, entity_id,
                                                            acs_url=None, certificate=None, display_name=None)
```

Google Apps SPHandler implementation.

## 3.3 Configuration

The IdP needs two configuration options by default, `SAML2_IDP` and `SAML2_SERVICE_PROVIDERS`. `SAML2_IDP` configures the IdP itself, while `SAML2_SERVICE_PROVIDERS` specifies all the SPs this IdP supports.

```
from flask_saml2.utils import certificate_from_file, private_key_from_file

SAML2_IDP = {
    'autosubmit': True,
    'certificate': certificate_from_file('keys/idp_certificate.pem'),
    'private_key': private_key_from_file('keys/idp_private_key.pem'),
}

SAML2_SERVICE_PROVIDERS = [
```

(continues on next page)

(continued from previous page)

```
{
    'CLASS': 'myapp.SPHandler',
    'OPTIONS': {
        'display_name': 'Example Service Provider',
        'entity_id': 'http://service.example.com/saml/metadata.xml',
        'acs_url': 'http://service.example.com/saml/acs/',
        'certificate': certificate_from_file('keys/example_sp_certificate.pem'),
    },
},
]
```

SAML2\_IDP is documented in *IdentityProvider.get\_idp\_config()*.

SAML2\_SERVICE\_PROVIDERS is a list of SPs the IdP will authenticate users for. Each SP is represented as a dict. CLASS is the dotted Python path to a *SPHandler* subclass, and OPTIONS is a dict of keyword arguments to its constructor. Refer to *SPHandler* for more information on constructor arguments.



## SERVICE PROVIDERS

A Service Provider (SP) is a website that users visit, that uses a separate Identity Provider (IdP) to authenticate users.

Flask SAML2 provides all of the functionality required to implement your own SP that can authenticate using one or more external IdPs. These IdPs can be written using `flask_saml2.idp`, or come from external providers.

The method `ServiceProvider.create_blueprint()` generates a Flask `Blueprint`, which needs to be registered in your application via `app.register_blueprint(sp.create_blueprint())`.

Any Identity Providers the SP can authenticate with need to be registered as well. These will be instances of `IdPHandler`.

An functional example SP and Flask application can be found in the `examples/` directory of the repository.

### 4.1 ServiceProvider

**class** `flask_saml2.sp.ServiceProvider`

Developers should subclass `ServiceProvider` and provide methods to interoperate with their specific environment. All user interactions are performed through methods on this class.

There are no methods that must be overridden, but overriding `get_default_login_return_url()` and `get_logout_return_url()` is recommended.

**session\_auth\_data\_key** = `'saml_auth_data'`

What key to store authentication details under in the session.

**blueprint\_name** = `'flask_saml2_sp'`

The name of the blueprint to generate.

**login\_successful** (`auth_data, relay_state`)

Called when a user is successfully logged on. Subclasses should override this if they want to do more with the returned user data. Returns a `flask.Response`, which is usually a redirect to `get_default_login_return_url()`, or a redirect to the protected resource the user initially requested. Subclasses may override this method and return a different response, but they *must* call `super()`.

**Return type** `Response`

**get\_sp\_config()**

Get the configuration for this SP. Defaults to `SAML2_SP` from `flask.Flask.config`. The configuration should be a dict like:

```
{
    # The X509 certificate and private key this SP uses to
    # encrypt, validate, and sign payloads.
```

(continues on next page)

(continued from previous page)

```
{
    'certificate': ...,
    'private_key': ...,
}
```

To load the certificate and private\_key values, see

- `certificate_from_string()`
- `certificate_from_file()`
- `private_key_from_string()`
- `private_key_from_file()`

**Return type** `dict`

**`get_sp_entity_id()`**

The unique identifier for this Service Provider. By default, this uses the metadata URL for this SP.

See `get_metadata_url()`.

**Return type** `str`

**`get_sp_certificate()`**

Get the public certificate for this SP.

**Return type** `Optional[X509]`

**`get_sp_private_key()`**

Get the private key for this SP.

**Return type** `Optional[PKey]`

**`get_sp_signer()`**

Get the signing algorithm used by this SP.

**Return type** `Optional[Signer]`

**`get_sp_digester()`**

Get the digest algorithm used by this SP.

**Return type** `Digester`

**`should_sign_requests()`**

Should this SP sign its SAML statements. Defaults to True if the SP is configured with both a certificate and a private key.

**Return type** `bool`

**`get_identity_providers()`**

Get an iterable of identity provider config dicts. “config” should be a dict specifying an IdPHandler subclass and optionally any constructor arguments:

```
>>> list(sp.get_identity_providers())
[{'CLASS': 'my_app.identity_providers.MyIdPIdPHandler',
  'OPTIONS': {
    'entity_id': 'https://idp.example.com/metadata.xml',
  },
}]
```

Defaults to `current_app.config['SAML2_IDENTITY_PROVIDERS']`.

**Return type** `Iterable[Tuple[str, dict]]`

**get\_login\_url()**

The URL of the endpoint that starts the login process.

**Return type** `str`

**get\_acs\_url()**

The URL for the Assertion Consumer Service for this SP.

**Return type** `str`

**get\_sls\_url()**

The URL for the Single Logout Service for this SP.

**Return type** `str`

**get\_metadata\_url()**

The URL for the metadata xml for this SP.

**Return type** `str`

**get\_default\_login\_return\_url()**

The default URL to redirect users to once they have logged in.

**Return type** `Optional[str]`

**get\_login\_return\_url()**

Get the URL to redirect the user to now that they have logged in.

**Return type** `Optional[str]`

**get\_logout\_return\_url()**

The URL to redirect users to once they have logged out.

**Return type** `Optional[str]`

**is\_valid\_redirect\_url(url)**

Is this URL valid and safe to redirect to? Defaults to only allowing URLs on the current server.

**Return type** `str`

**make\_idp\_handler(config)**

Construct an *IdPHandler* from a config dict from *get\_identity\_providers()*.

**Return type** `IdPHandler`

**get\_idp\_handlers()**

Get the *IdPHandler* for each service provider defined.

**Return type** `Iterable[IdPHandler]`

**get\_default\_idp\_handler()**

Get the default IdP to sign in with. When logging in, if there is a default IdP, the user will be automatically logged in with that IdP. Return `None` if there is no default IdP. If there is no default, a list of IdPs to sign in with will be presented by the login view.

**Return type** `Optional[IdPHandler]`

**get\_idp\_handler\_by\_entity\_id(entity\_id)**

Find the *IdPHandler* instance with a matching entity ID.

**Return type** `IdPHandler`

**get\_idp\_handler\_by\_current\_session()**

Get the *IdPHandler* used to authenticate the currently logged in user.

**Return type** `IdPHandler`

**`login_required()`**

Check if a user is currently logged in to this session, and `flask.abort()` with a redirect to the login page if not. It is suggested to use `is_user_logged_in()`.

**`is_user_logged_in()`**

Check if the user is currently logged in / authenticated with an IdP.

**Return type** `bool`

**`logout()`**

Terminate the session for a logged in user.

**`render_template(template, **context)`**

Render an HTML template. This method can be overridden to inject more context variables if required.

**Return type** `str`

**`set_auth_data_in_session(auth_data)`**

Store authentication details from the `IdPHandler` in the browser session.

**`clear_auth_data_in_session()`**

Clear the authentication details from the session. This will effectively log the user out.

**`get_auth_data_in_session()`**

Get an `AuthData` instance from the session data stored for the currently logged in user.

**Return type** `AuthData`

**`make_absolute_url(url)`**

Take a local URL and make it absolute by prepending the current `SERVER_NAME`.

**Return type** `str`

**`get_metadata_context()`**

Get any extra context for the metadata template. Suggested extra context variables include 'org' and 'contacts'.

**Return type** `dict`

**`create_blueprint()`**

Create a Flask `flask.Blueprint` for this Service Provider.

**Return type** `Blueprint`

## 4.2 IdPHandler

A `flask_saml2.idp.ServiceProvider` handles requests from Identity Providers via `flask_saml2.idp.IdPHandler` instances.

See [Configuration](#) for configuration options.

```
class flask_saml2.sp.IdPHandler(sp, *, entity_id, display_name=None, sso_url=None,
                               slo_url=None, certificate=None, **kwargs)
```

Represents an Identity Provider that the running Service Provider knows about. This class should be subclassed for Identity Providers that need specific configurations.

**`get_idp_sso_url()`**

Get the Single Sign On URL for this IdP.

**get\_idp\_slo\_url()**  
Get the Single Log Out URL for this IdP.

**get\_sp\_acs\_url()**  
Get the Attribute Consumer Service URL on the current SP this IdP should send responses to.

**get\_authn\_request** (*template=<class 'flask\_saml2.sp.xml\_templates.AuthnRequest'>, **\*\*parameters***)  
Make a AuthnRequest to send to this IdP.

**get\_logout\_request** (*auth\_data, template=<class 'flask\_saml2.sp.xml\_templates.LogoutRequest'>, **\*\*parameters***)  
Make a LogoutRequest for the authenticated user to send to this IdP.

**make\_login\_request\_url** (*relay\_state=None*)  
Make a LoginRequest url and query string for this IdP.

**Return type** `str`

**decode\_saml\_string** (*saml\_string*)  
Decode an incoming SAMLResponse into an XML string.

**Return type** `bytes`

**encode\_saml\_string** (*saml\_string*)  
Encoding an XML string into a SAMLRequest.

**Return type** `str`

**get\_response\_parser** (*saml\_response*)  
Make a ResponseParser instance to handle this response.

**get\_auth\_data** (*response*)  
Create an AuthData instance from a SAML Response. The response is validated first.

**Return type** `AuthData`

## 4.3 Configuration

The SP needs two configuration options by default, `SAML2_SP` and `SAML2_IDENTITY_PROVIDERS`. `SAML2_SP` configures the Service Provider itself, while `SAML2_IDENTITY_PROVIDERS` specifies all the IdPs the SP can authenticate with.

```
from flask_saml2.utils import certificate_from_file, private_key_from_file

SAML2_SP = {
    'certificate': certificate_from_file('keys/sp_certificate.pem'),
    'private_key': private_key_from_file('keys/sp_private_key.pem'),
}

SAML2_IDENTITY_PROVIDERS = [
    {
        'CLASS': 'myapp.IdPHandler',
        'OPTIONS': {
            'display_name': 'Example Identity Provider',
            'entity_id': 'https://idp.example.com/saml/metadata.xml',
            'sso_url': 'https://idp.example.com/saml/login/',
            'slo_url': 'https://idp.example.com/saml/logout/',
            'certificate': certificate_from_file('keys/idp_certificate.pem'),
```

(continues on next page)

(continued from previous page)

```
    },  
    },  
]
```

SAML2\_SP is documented in `ServiceProvider.get_sp_config()`.

SAML2\_IDENTITY\_PROVIDERS is a list of IdPs the SP can use for authentication. Each IdP is represented as a dict. CLASS is the dotted Python path to a `IdPHandler` subclass, and OPTIONS is a dict of keyword arguments to its constructor. Refer to `IdPHandler` for more information on constructor arguments.

## 4.4 Example

To make your application into a Service Provider, create a `ServiceProvider` subclass, instantiate it, and register it's `Blueprint` with your `Flask` application:

```
from flask import Flask  
from flask_saml2.sp import ServiceProvider  
  
class MyServiceProvider(ServiceProvider):  
    def get_default_login_return_url(self):  
        return url_for('dashboard')  
  
    def get_logout_return_url(self):  
        return url_for('index')  
  
sp = ServiceProvider()  
  
app = Flask()  
app.register_blueprint(sp.create_blueprint(), url_prefix='/saml/')  
app.run()
```

## EXCEPTIONS

All the SAML-specific exceptions this library can throw.

**exception** flask\_saml2.exceptions.**SAML2Exception**

Base exception for all flask\_saml2 exceptions.

**exception** flask\_saml2.exceptions.**MessageException** (*msg*)

An exception with a nicely formatted error message.

**exception** flask\_saml2.exceptions.**CannotHandleAssertion** (*msg*)

This SP or IdP handler can not handle this assertion.

**exception** flask\_saml2.exceptions.**UserNotAuthorized** (*msg*)

User not authorized for SAML 2.0 authentication.

**exception** flask\_saml2.exceptions.**ImproperlyConfigured** (*msg*)

Someone done goofed when configuring this application.





## INTERNAL MODULES

These modules are used by the *IdentityProvider* and *ServiceProvider* classes. They may be useful to you if you are writing a custom handler to support a particular upstream IdP or downstream SP.

### 6.1 Encoding and decoding

Utilities to encode and decode zlib and base64 data.

`flask_saml2.codex.decode_base64_and_inflate(b64string)`  
Turn a base64-encoded zlib-compressed blob back in to the original bytes. The opposite of `deflate_and_base64_encode()`.

**Return type** `bytes`

`flask_saml2.codex.deflate_and_base64_encode(string_val)`  
zlib-compress and base64-encode some data. The opposite of `decode_base64_and_inflate()`.

**Return type** `bytes`

`flask_saml2.codex.decode_saml_xml(data)`  
Decodes some base64-encoded and possibly zipped string into an XML string.

**Return type** `bytes`

### 6.2 Signing and digest tools

Functions and classes that deal with signing data and making digests.

**class** `flask_saml2.signing.Digester`

Base class for all the digest methods. SAML2 digest methods have an identifier in the form of a URL, and must produce a text digest.

Subclasses should set the `uri` attribute and provide a `make_digest()` method.

Implemented digest methods: `ShalDigester`.

Example:

```
>>> from flask_saml2.signing import ShalDigester
>>> digester = ShalDigester()
>>> digester(b'Hello, world!')
'1DpwLQbzRZmu4fja jvn3KWax1pk='
```

**uri = None**

The URI identifying this digest method

**make\_digest (data)**

Make a binary digest of some binary data using this digest method.

**Return type** `bytes`

**class flask\_saml2.signing.Signer**

Sign some data with a particular algorithm. Each Signer may take different constructor arguments, but each will have a uri attribute and will sign data when called.

Implemented signers: `RsaSha1Signer`.

Example:

```
>>> from flask_saml2.signing import RsaSha1Signer
>>> from flask_saml2.utils import private_key_from_file
>>> key = private_key_from_file('tests/keys/sample/idp-private-key.pem')
>>> signer = RsaSha1Signer(private_key)
>>> signer(b'Hello, world!')
'YplgloQDPLiozAWoY9ykgQ4eicojNnU+KjRrwGp67jHM5FGkQZ71Pk1Bgo631WA5B1hopQByRh/
↳elqtEEN+vRA=='
```

**uri = None**

The URI identifying this signing method

**class flask\_saml2.signing.SignedInfoTemplate (params={})**

A `<SignedInfo>` node, such as:

```
<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#
↳"></ds:CanonicalizationMethod>
  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></
↳ds:SignatureMethod>
  <ds:Reference URI="#${REFERENCE_URI}">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
↳signature"></ds:Transform>
      <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></
↳ds:Transform>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></
↳ds:DigestMethod>
    <ds:DigestValue>${SUBJECT_DIGEST}</ds:DigestValue>
  </ds:Reference>
</ds:SignedInfo>
```

**generate\_xml ()**

Generate the XML node for this template. Generally accessed through `xml`.

**class flask\_saml2.signing.SignatureTemplate (params={})**

A `<Signature>` node, such as:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  ${SIGNED_INFO}
  <ds:SignatureValue>${RSA_SIGNATURE}</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>${CERTIFICATE}</ds:X509Certificate>
```

(continues on next page)

(continued from previous page)

```

        </ds:X509Data>
    </ds:KeyInfo>
</ds:Signature>
    
```

**classmethod** `sign` (*subject*, *certificate*, *digester*, *signer*, *reference\_uri*)

Create a *SignatureTemplate* by signing a subject string.

#### Parameters

- **subject** (*str*) – The string to sign. This is usually the canonical string representation of the XML node this `<Signature>` verifies.
- **certificate** (*X509*) – The certificate to sign the data with
- **digester** (*Digester*) – The algorithm used to make the digest
- **signer** (*Signer*) – The algorithm used to sign the data
- **reference\_uri** (*str*) – The ID of the element that is signed

See also: *SignableTemplate.sign()*

**generate\_xml** ()

Generate the XML node for this template. Generally accessed through *xml*.

**class** `flask_saml2.signing.SignableTemplate` (*params*={})

An *XmlTemplate* that supports being signed, by adding an `<Signature>` element.

**signature\_index** = 1

The element index where the signature should be inserted

**id\_parameter** = None

The parameter that contains the element ID

See *get\_id()* and *sign()*

**sign** (*certificate*, *digester*, *signer*)

Cryptographically sign this template by inserting a `<Signature>` element.

The ID of the node to sign is fetched from *get\_id()*.

#### Parameters

- **certificate** (*X509*) – The certificate to sign the data with
- **digester** (*Digester*) – The algorithm used to make the digest
- **signer** (*Signer*) – The algorithm used to sign the data

**Return type** *ElementBase*

**make\_signature** (*certificate*, *digester*, *signer*)

Create XML `<Signature>` node for the subject text.

**Return type** *SignatureTemplate*

**add\_signature** (*signature*)

Insert a `<Signature>` into this node.

**get\_id** ()

Get the ID of the root node, required to *sign()* this node. By default, grabs the ID from the parameter named in *id\_parameter*.

**Return type** *str*

`flask_saml2.signing.sign_query_parameters` (*signer*, *bits*)  
 Sign the bits of a query string.

```
>>> signer = ... # A Signer instance
>>> bits = [('Foo', '1'), ('Bar', '2')]
>>> sign_query_parameters(signer, bits)
"Foo=1&Bar=2&SigAlg=...&Signature=..."
```

**Return type** `str`

## 6.3 XML tools

### 6.3.1 XML parsing

The `flask_saml2.xml_parser` provides tools for parsing XML documents from an IdP or a SP. If the documents are signed, they will be verified as part of parsing.

**class** `flask_saml2.xml_parser.XmlParser` (*xml\_string*, *certificate*)  
 Parse a possibly-signed XML document. Subclasses must implement `is_signed()`.

**certificate** = `None`  
 The certificate the document is signed with

**xml\_string** = `None`  
 The input XML document as a string

**xml\_tree** = `None`  
 The parsed XML document

**parse\_request** (*xml\_string*)  
 Parse the SAML request. :raises: `ValueError`

**Return type** `None`

**is\_signed** ()  
 Is this request signed? Looks for a `<ds:Signature>` element. Different sources will generate different signed XML documents, so this method must be implemented differently for each source.

**parse\_signed** (*xml\_tree*, *certificate*)  
 Replaces all parameters with only the signed parameters. You should provide an x509 certificate obtained out-of-band, usually via the SAML metadata. Otherwise the signed data will be verified with only the certificate provided in the request. This is INSECURE and more-or-less only useful for testing.

**Return type** `ElementBase`

### 6.3.2 XML templates

**class** `flask_saml2.xml_templates.XmlTemplate` (*params*={})  
 Base XML template class. A template can represent a single node, a tree, or a whole XML document.

**namespace** = `None`  
 XML namespace for this node or document

**property** `xml`  
 The XML node this template constructed. Generated using `generate_xml()`.

**generate\_xml()**

Generate the XML node for this template. Generally accessed through `xml`.

**Return type** `ElementBase`

**get\_xml\_string()**

Render the XML node to a string. The string representation is rendered as canonical c14n XML, to make verification and signing possible.

**Return type** `str`

**element** (*tag*, \*, *namespace=None*, *attrs=None*, *children=None*, *text=None*)

Shortcut for creating an ElementTree Element, with optional attributes, children, and text.

**Parameters**

- **str** (*text*) – tag to give XML element
- **str** – Namespace to use for the element. Defaults to `get_namespace()` if None.
- **dict** (*attrs*) – Element attributes. If an attribute value is None, the attribute is ignored.
- **list** (*children*) – Element children. If an item in children is None, the item is ignored.
- **str** – Element text content, if any.

**Return type** `ElementBase`

**Returns** `xml.etree.ElementTree.Element`

**get\_namespace\_map()**

Get all the namespaces potentially used by this node, as a etree nsmmap.

**Return type** `Mapping[str, str]`

**get\_namespace()**

Get the namespace URI for this node. Looks up the namespace alias `namespace` in `get_namespace_map()`.

**Return type** `str`

**class** `flask_saml2.xml_templates.NameIDTemplate` (*params={}*)

A <NameID> node, such as:

```
<NameID Format="{SUBJECT_FORMAT}" SPNameQualifier="{SP_NAME_QUALIFIER}">
    ${SUBJECT}
</NameID>
```

**generate\_xml()**

Generate the XML node for this template. Generally accessed through `xml`.

## 6.4 Utilities

**class** `flask_saml2.utils.cached_property` (*func*, *name=None*, *doc=None*)

A decorator that converts a function into a lazy property. The function wrapped is called the first time to retrieve the result and then that calculated result is used the next time you access the value:

```
class Foo(object):
    @cached_property
    def foo(self):
```

(continues on next page)

(continued from previous page)

```
# calculate something important here
return 42
```

The class has to have a `__dict__` in order for this property to work.

`flask_saml2.utils.import_string(path)`

Import a dotted Python path to a class or other module attribute. `import_string('foo.bar.MyClass')` will return the class `MyClass` from the package `foo.bar`.

**Return type** `Any`

`flask_saml2.utils.get_random_id()`

Generate a random ID string. The random ID will start with the `'_'` character.

**Return type** `str`

`flask_saml2.utils.utcnow()`

Get the current time in UTC, as an aware `datetime.datetime`.

**Return type** `datetime`

`flask_saml2.utils.certificate_to_string(certificate)`

Take an x509 certificate and encode it to a string suitable for adding to XML responses.

**Parameters** `certificate` (`X509`) – A certificate, perhaps loaded from `certificate_from_file()`.

**Return type** `str`

`flask_saml2.utils.certificate_from_string(certificate, format=1)`

Load an X509 certificate from a string. This just strips off the header and footer text.

**Parameters**

- **str** – A certificate string.
- **format** – The format of the certificate, from `crypto` — [Generic cryptographic module](#).

**Return type** `X509`

`flask_saml2.utils.certificate_from_file(filename, format=1)`

Load an X509 certificate from `filename`.

**Parameters**

- **filename** (`Union[str, Path]`) – The path to the certificate on disk.
- **format** – The format of the certificate, from `crypto` — [Generic cryptographic module](#).

**Return type** `X509`

`flask_saml2.utils.private_key_from_string(private_key, format=1)`

Load a private key from a string.

**Parameters**

- **str** – A private key string.
- **format** – The format of the private key, from `crypto` — [Generic cryptographic module](#).

**Return type** `PKey`

`flask_saml2.utils.private_key_from_file(filename, format=1)`

Load a private key from `filename`.

**Parameters**

- **filename** (`Union[str, Path]`) – The path to the private key on disk.
- **format** – The format of the private key, from `crypto` — Generic cryptographic module.

**Return type** `PKey`





## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### f

- `flask_saml2`, [1](#)
- `flask_saml2.codex`, [21](#)
- `flask_saml2.exceptions`, [19](#)
- `flask_saml2.idp`, [5](#)
- `flask_saml2.idp.sp`, [11](#)
- `flask_saml2.idp.sp.google_apps`, [11](#)
- `flask_saml2.idp.sp.salesforce`, [11](#)
- `flask_saml2.signing`, [21](#)
- `flask_saml2.sp`, [12](#)
- `flask_saml2.utils`, [25](#)
- `flask_saml2.xml_parser`, [24](#)
- `flask_saml2.xml_templates`, [24](#)



## INDEX

### A

`add_signature()` (*flask\_saml2.signing.SignableTemplate method*), 23  
`assertion_template` (*flask\_saml2.idp.SPHandler attribute*), 10

### B

`blueprint_name` (*flask\_saml2.sp.ServiceProvider attribute*), 13  
`build_assertion()` (*flask\_saml2.idp.SPHandler method*), 10  
`build_response()` (*flask\_saml2.idp.SPHandler method*), 10

### C

`cached_property` (*class in flask\_saml2.utils*), 25  
`CannotHandleAssertion`, 19  
`certificate` (*flask\_saml2.xml\_parser.XmlParser attribute*), 24  
`certificate_from_file()` (*in module flask\_saml2.utils*), 26  
`certificate_from_string()` (*in module flask\_saml2.utils*), 26  
`certificate_to_string()` (*in module flask\_saml2.utils*), 26  
`clear_auth_data_in_session()` (*flask\_saml2.sp.ServiceProvider method*), 16  
`create_blueprint()` (*flask\_saml2.idp.IdentityProvider method*), 9  
`create_blueprint()` (*flask\_saml2.sp.ServiceProvider method*), 16

### D

`decode_base64_and_inflate()` (*in module flask\_saml2.codex*), 21  
`decode_saml_string()` (*flask\_saml2.idp.SPHandler method*), 11  
`decode_saml_string()` (*flask\_saml2.sp.IdPHandler method*), 17

`decode_saml_xml()` (*in module flask\_saml2.codex*), 21  
`deflate_and_base64_encode()` (*in module flask\_saml2.codex*), 21  
`Digester` (*class in flask\_saml2.signing*), 21

### E

`element()` (*flask\_saml2.xml\_templates.XmlTemplate method*), 25  
`encode_response()` (*flask\_saml2.idp.SPHandler method*), 10  
`encode_saml_string()` (*flask\_saml2.sp.IdPHandler method*), 17  
`extract_request_parameters()` (*flask\_saml2.idp.SPHandler method*), 10

### F

`flask_saml2` (*module*), 1  
`flask_saml2.codex` (*module*), 21  
`flask_saml2.exceptions` (*module*), 19  
`flask_saml2.idp` (*module*), 5  
`flask_saml2.idp.sp` (*module*), 11  
`flask_saml2.idp.sp.google_apps` (*module*), 11  
`flask_saml2.idp.sp.salesforce` (*module*), 11  
`flask_saml2.signing` (*module*), 21  
`flask_saml2.sp` (*module*), 12  
`flask_saml2.utils` (*module*), 25  
`flask_saml2.xml_parser` (*module*), 24  
`flask_saml2.xml_templates` (*module*), 24  
`format_assertion()` (*flask\_saml2.idp.SPHandler method*), 10  
`format_response()` (*flask\_saml2.idp.SPHandler method*), 10

### G

`generate_xml()` (*flask\_saml2.signing.SignatureTemplate method*), 23  
`generate_xml()` (*flask\_saml2.signing.SignedInfoTemplate method*), 22  
`generate_xml()` (*flask\_saml2.xml\_templates.NameIDTemplate method*), 25

<code>generate_xml()</code> ( <i>flask_saml2.xml_templates.XmlTemplate</i> method), 24	<code>get_idp_signer()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8
<code>get_acs_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_idp_slo_url()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 16
<code>get_assertion_id()</code> ( <i>flask_saml2.idp.SPHandler</i> method), 10	<code>get_idp_sso_url()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 16
<code>get_audience()</code> ( <i>flask_saml2.idp.SPHandler</i> method), 10	<code>get_login_return_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15
<code>get_auth_data()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 17	<code>get_login_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15
<code>get_auth_data_in_session()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 16	<code>get_logout_request()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 17
<code>get_authn_request()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 17	<code>get_logout_return_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15
<code>get_current_user()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 9	<code>get_metadata_context()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 9
<code>get_default_idp_handler()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_metadata_context()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 16
<code>get_default_login_return_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_metadata_url()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8
<code>get_id()</code> ( <i>flask_saml2.signing.SignableTemplate</i> method), 23	<code>get_metadata_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15
<code>get_identity_providers()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 14	<code>get_namespace()</code> ( <i>flask_saml2.xml_templates.XmlTemplate</i> method), 25
<code>get_idp_autosubmit()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8	<code>get_namespace_map()</code> ( <i>flask_saml2.xml_templates.XmlTemplate</i> method), 25
<code>get_idp_certificate()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8	<code>get_random_id()</code> (in module <i>flask_saml2.utils</i> ), 26
<code>get_idp_config()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 7	<code>get_response_context()</code> ( <i>flask_saml2.idp.SPHandler</i> method), 10
<code>get_idp_digester()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8	<code>get_response_id()</code> ( <i>flask_saml2.idp.SPHandler</i> method), 10
<code>get_idp_entity_id()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8	<code>get_response_parser()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 17
<code>get_idp_handler_by_current_session()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_service_providers()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8
<code>get_idp_handler_by_entity_id()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_slo_url()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8
<code>get_idp_handlers()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15	<code>get_sls_url()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 15
<code>get_idp_private_key()</code> ( <i>flask_saml2.idp.IdentityProvider</i> method), 8	<code>get_sp_acs_url()</code> ( <i>flask_saml2.sp.IdPHandler</i> method), 17
	<code>get_sp_certificate()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 14
	<code>get_sp_config()</code> ( <i>flask_saml2.sp.ServiceProvider</i> method), 13

[get\\_sp\\_digester\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [14](#)  
[get\\_sp\\_entity\\_id\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [14](#)  
[get\\_sp\\_handlers\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[get\\_sp\\_private\\_key\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [14](#)  
[get\\_sp\\_signer\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [14](#)  
[get\\_sso\\_url\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [8](#)  
[get\\_subject\(\)](#) ([flask\\_saml2.idp.SPHandler](#) [method](#)), [10](#)  
[get\\_user\\_email\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[get\\_user\\_nameid\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[get\\_xml\\_string\(\)](#) ([flask\\_saml2.xml\\_templates.XmlTemplate](#) [method](#)), [25](#)  
[GoogleAppsSPHandler](#) ([class](#) [in flask\\_saml2.idp.sp.google\\_apps](#)), [11](#)

**I**

[id\\_parameter\(\)](#) ([flask\\_saml2.signing.SignableTemplate](#) [attribute](#)), [23](#)  
[IdentityProvider](#) ([class](#) [in flask\\_saml2.idp](#)), [7](#)  
[IdPHandler](#) ([class](#) [in flask\\_saml2.sp](#)), [16](#)  
[import\\_string\(\)](#) ([in module flask\\_saml2.utils](#)), [26](#)  
[ImproperlyConfigured](#), [19](#)  
[is\\_signed\(\)](#) ([flask\\_saml2.xml\\_parser.XmlParser](#) [method](#)), [24](#)  
[is\\_user\\_logged\\_in\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[is\\_user\\_logged\\_in\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [16](#)  
[is\\_valid\\_redirect\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[is\\_valid\\_redirect\(\)](#) ([flask\\_saml2.idp.SPHandler](#) [method](#)), [11](#)  
[is\\_valid\\_redirect\\_url\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [15](#)

**L**

[login\\_required\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [8](#)  
[login\\_required\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [16](#)  
[login\\_successful\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [13](#)  
[logout\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[logout\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [16](#)

**M**

[make\\_absolute\\_url\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [16](#)  
[make\\_digest\(\)](#) ([flask\\_saml2.signing.Digester](#) [method](#)), [22](#)  
[make\\_idp\\_handler\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [15](#)  
[make\\_login\\_request\\_url\(\)](#) ([flask\\_saml2.sp.IdPHandler](#) [method](#)), [17](#)  
[make\\_response\(\)](#) ([flask\\_saml2.idp.SPHandler](#) [method](#)), [11](#)  
[make\\_signature\(\)](#) ([flask\\_saml2.signing.SignableTemplate](#) [method](#)), [23](#)  
[MessageException](#), [19](#)

**N**

[NameIDTemplate](#) ([class](#) [in flask\\_saml2.xml\\_templates](#)), [25](#)  
[namespace](#) ([flask\\_saml2.xml\\_templates.XmlTemplate](#) [attribute](#)), [24](#)

**P**

[parse\\_authn\\_request\(\)](#) ([flask\\_saml2.idp.SPHandler](#) [method](#)), [11](#)  
[parse\\_logout\\_request\(\)](#) ([flask\\_saml2.idp.SPHandler](#) [method](#)), [11](#)  
[parse\\_request\(\)](#) ([flask\\_saml2.xml\\_parser.XmlParser](#) [method](#)), [24](#)  
[parse\\_signed\(\)](#) ([flask\\_saml2.xml\\_parser.XmlParser](#) [method](#)), [24](#)  
[private\\_key\\_from\\_file\(\)](#) ([in module flask\\_saml2.utils](#)), [26](#)  
[private\\_key\\_from\\_string\(\)](#) ([in module flask\\_saml2.utils](#)), [26](#)

**R**

[render\\_template\(\)](#) ([flask\\_saml2.idp.IdentityProvider](#) [method](#)), [9](#)  
[render\\_template\(\)](#) ([flask\\_saml2.sp.ServiceProvider](#) [method](#)), [16](#)

`response_template` (*flask\_saml2.idp.SPHandler* attribute), 10

## S

`SalesforceSPHandler` (class in *flask\_saml2.idp.sp.salesforce*), 11

`SAML2Exception`, 19

`ServiceProvider` (class in *flask\_saml2.sp*), 13

`session_auth_data_key` (*flask\_saml2.sp.ServiceProvider* attribute), 13

`set_auth_data_in_session()` (*flask\_saml2.sp.ServiceProvider* method), 16

`should_sign_requests()` (*flask\_saml2.sp.ServiceProvider* method), 14

`sign()` (*flask\_saml2.signing.SignableTemplate* method), 23

`sign()` (*flask\_saml2.signing.SignatureTemplate* class method), 23

`sign_query_parameters()` (in module *flask\_saml2.signing*), 23

`SignableTemplate` (class in *flask\_saml2.signing*), 23

`signature_index` (*flask\_saml2.signing.SignableTemplate* attribute), 23

`SignatureTemplate` (class in *flask\_saml2.signing*), 22

`SignedInfoTemplate` (class in *flask\_saml2.signing*), 22

`Signer` (class in *flask\_saml2.signing*), 22

`SPHandler` (class in *flask\_saml2.idp*), 9

## U

`uri` (*flask\_saml2.signing.Digester* attribute), 21

`uri` (*flask\_saml2.signing.Signer* attribute), 22

`UserNotAuthorized`, 19

`utcnow()` (in module *flask\_saml2.utils*), 26

## V

`validate_request()` (*flask\_saml2.idp.SPHandler* method), 10

`validate_user()` (*flask\_saml2.idp.SPHandler* method), 11

## X

`xml()` (*flask\_saml2.xml\_templates.XmlTemplate* property), 24

`xml_string` (*flask\_saml2.xml\_parser.XmlParser* attribute), 24

`xml_tree` (*flask\_saml2.xml\_parser.XmlParser* attribute), 24

`XmlParser` (class in *flask\_saml2.xml\_parser*), 24

`XmlTemplate` (class in *flask\_saml2.xml\_templates*), 24