WS API technical documentation

Testing Web services

Demo account

Demo account is intended for trying technical aspects of our WS API without the necessity for you to have your own WS API key.
It provides full data access to single DEMO microregion (10x10 km) around following location: http://solargis.info/pvplanner/#c=48.61259,20.827079&z=11

The demo key can be used for making WS API calls for both of our service types: DataDelivery + PvPlanner and also for REST + SOAP access.

REST WS credentials:

key: **demo**

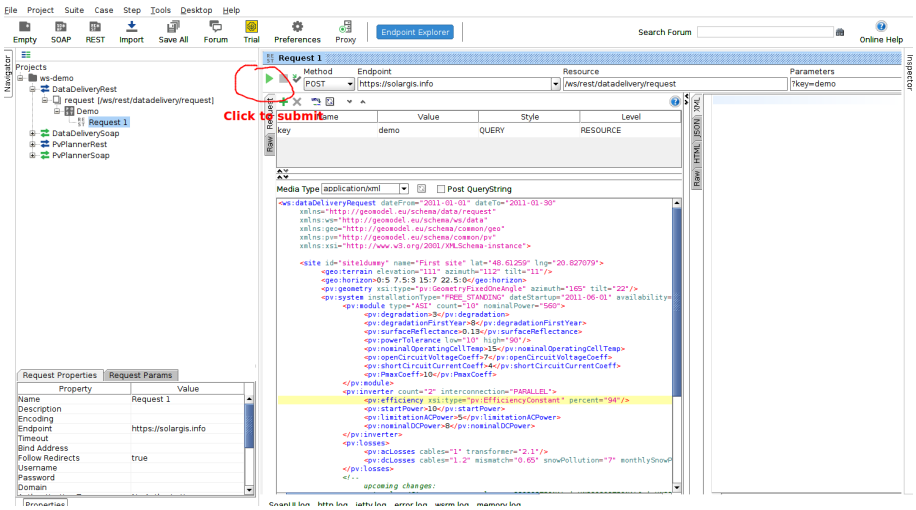
SOAP WS credentials:

`username:` **demo**
`password:` **demo**

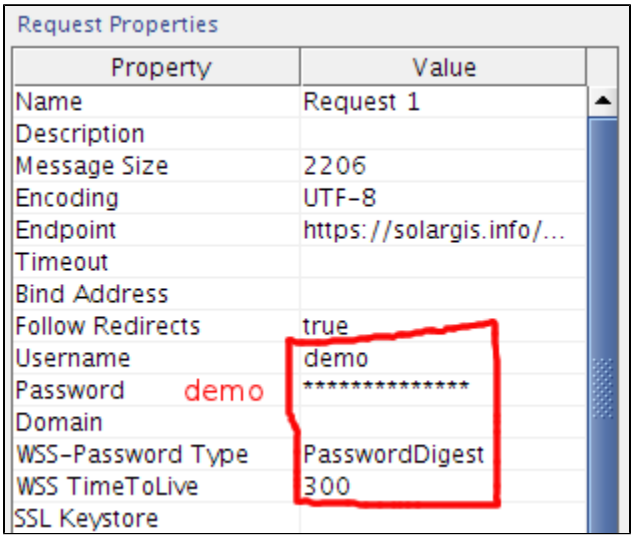
You can use our WS API demo credentials to code and test your own implementation of WS API calls. If you also want to test quality of our data, please ask Solargis sales representatives for free trial access (i.e. access to limited locations / time periods of data with your own WS API free trial key)

SoapUI examples (how to try basic calls to our WS API)

We recommend starting with SoapUI examples - getting them working before you attempt your own implementation. Using SoapUI is very easy:
just install the multiplatform standalone application SoapUI import demo project ws-demo-soapui-project.xml into SoapUI and submit prepared request messages.
The demo project contains DataDelivery and PvPlanner examples preconfigured with #Demo account thus the location is limited to demo region (10x10km)
The REST access examples work out of the box, all that is needed is to just run the requests by clicking on the "Submit" button and you should see successful results of the call. Here is DataDelivery REST access example:



For SOAP access you might need to fill in the following request properties in SoapUI:



Once you get the demo requests working in SoapUI you can then change key for one that was provided to you by our sales representative and alter the XML request according to your permissions (may require change of latitude / longitude / dateFrom / dateTo, processingKeys etc. in the XML request) We recommend that you first create the working request adapted for your specific requirements and you start working on your own implementation only after you validate that the request is working in SoapUI. That way:

- if the request is not working in SoapUI you can easily tell that problem is in the XML request its self and not for example in your implementation
- if the request is working in SoapUI but not in your implementation you can then tell that the problem is in your implementation and not in the XML request it's self

If you were able to get the demo examples working but you are unable to get the requests working with your own API key, then please try recommendations in our our troubleshooting guide.
For adapting the demo requests to your requirements we recommend that you take inspiration from our example request / response documentation (see the next section). It's best if you start from simple request and adapt it to more complicated ones and
that you validate in SoapUI that the request is still working after each incremental change. Detailed explanation of all the parameters in the request can be found in our Solargis API User Guide

PvPlanner Web Service

click here for request and response examples

DataDelivery Web Service

click here for request and response examples

Implementing client for Web services

Java client implementation guide (full guide here)

- Define Spring WebServiceTemplate context:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:oxm="http://www.springframework.org/schema/oxm"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/oxm http://www.springframework.org/schema
/oxm/spring-oxm.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans.xsd">

  <!-- contextPath - package where to find request and response classes to marshall / unmarshal
       if request and response are in different packages, declare marshaller and unmarshaller separately
-->
  <oxm:jaxb2-marshaller id="jaxb2marshaller" contextPath="eu.geomodel.schema.ws.pvplanner"/>

  <bean id="pvPlannerTemplate" class="org.springframework.ws.client.core.WebServiceTemplate"
      p:marshaller-ref="jaxb2marshaller" p:unmarshaller-ref="jaxb2marshaller"
      p:defaultUri="https://solargis.info/ws/soap/pvPlanner"> <!-- web service endpoint uri -->
    <property name="interceptors">
      <list>
        <bean class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor"
            p:securementActions="Timestamp UsernameToken"
            p:securementUsername="demo"
            p:securementPassword="demo"/>
      </list>
    </property>
  </bean>
</beans>
```

- Define client Spring bean:

```
// imports omitted
public class PvPlannerWsClient {

  @Autowired
  @Qualifier("pvPlannerTemplate")
  WebServiceTemplate template;

  public CalculateResponse callPvPlanner(/*some args*/) {
    return (CalculateResponse) template.marshalSendAndReceive(prepareRequest(/*some args*/));
  }

  private CalclulateRequest prepareRequest(/*some args*/) {
    // just sample request
    CalculateRequest request = new CalculateRequest();
    Location location = new Location();
    location.setLat(48.612590); // location of demo site
    location.setLng(20.827079);
    request.setLocation(location);
    PvSystem system = new PvSystem();
    system.setInstalledPower(1);
    system.setModuleType(ModuleTypeEnum.C_SI);
    Settings settings = new Settings();
    settings.setInverterEfficiency(97.5);
    settings.setDcLosses(5.5);
    settings.setAcLosses(1.5);
    settings.setAvailability(99);
    system.setSettings(settings);
    MountingFixedRoofMounted mounting = new MountingFixedRoofMounted();
    mounting.setAzimuth(175);
    mounting.setInclination(45D);
    system.setMounting(mounting);
    request.setSystem(system);
    return request;
  }

}
```

Simple PHP client

We will use request as static XML string. In real environment the request will be generated dynamically.

We wil do a simple **curl** call to access web service in REST mode.

```
<?php
header('Content-type: text/xml');
$xml = '<ws:dataDeliveryRequest dateFrom="2014-01-01" dateTo="2014-01-30"
    xmlns="http://geomodel.eu/schema/data/request"
    xmlns:ws="http://geomodel.eu/schema/ws/data"
    xmlns:geo="http://geomodel.eu/schema/common/geo"
    xmlns:pv="http://geomodel.eu/schema/common/pv"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <site id="site1dummy" name="First site" lat="48.61259" lng="20.827079">
        <geo:terrain elevation="111" azimuth="112" tilt="11"/>
        <geo:horizon>0:5 7.5:3 15:7 22.5:0</geo:horizon>
        <pv:geometry xsi:type="pv:GeometryFixedOneAngle" azimuth="165" tilt="22"/>
        <pv:system installationType="FREE_STANDING" dateStartup="2011-06-01" availability="99">
            <pv:module type="ASI" count="10" nominalPower="560">
                <pv:degradation>3</pv:degradation>
                <pv:degradationFirstYear>8</pv:degradationFirstYear>
                <pv:surfaceReflectance>0.13</pv:surfaceReflectance>
                <pv:powerTolerance low="10" high="90"/>
                <pv:nominalOperatingCellTemp>15</pv:nominalOperatingCellTemp>
                <pv:openCircuitVoltageCoeff>7</pv:openCircuitVoltageCoeff>
                <pv:shortCircuitCurrentCoeff>4</pv:shortCircuitCurrentCoeff>
                <pv:PmaxCoeff>10</pv:PmaxCoeff>
            </pv:module>
            <pv:inverter count="2" interconnection="PARALLEL">
                <pv:efficiency xsi:type="pv:EfficiencyConstant" percent="94"/>
                <pv:startPower>10</pv:startPower>
                <pv:limitationACPower>5</pv:limitationACPower>
                <pv:nominalDCPower>8</pv:nominalDCPower>
            </pv:inverter>
            <pv:losses>
                <pv:acLosses cables="1" transformer="2.1"/>
                <pv:dcLosses cables="1.2" mismatch="0.65" snowPollution="7" monthlySnowPollution="4 2 3 4 5 7 8
4 7 4 5 1"/>
            </pv:losses>
            <pv:topology xsi:type="pv:TopologySimple" type="PROPORTIONAL" relativeSpacing="1.5"/>
        </pv:system>
    </site>
    <processing key="GHI DIF DNI PVOUT" summarization="HOURLY" terrainShading="true"/>
</ws:dataDeliveryRequest>';
$url = 'https://solargis.info/ws/rest/datadelivery/request?key=demo';
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
curl_setopt($ch, CURLOPT_POSTFIELDS, $xml);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

$response = curl_exec($ch);
curl_close($ch);
echo $response;
?>
```

Simple Python client

Code examples are valid for both Python versions 2 and 3.

Data delivery Web service (API for getting time series data)

```
import requests

if __name__ == '__main__':
    request_xml = '''<ws:dataDeliveryRequest dateFrom="2014-04-28" dateTo="2014-04-28"
    xmlns="http://geomodel.eu/schema/data/request"
    xmlns:ws="http://geomodel.eu/schema/ws/data"
    xmlns:geo="http://geomodel.eu/schema/common/geo"
    xmlns:pv="http://geomodel.eu/schema/common/pv"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <site id="demo_site" name="Demo site" lat="48.61259" lng="20.827079">
    </site>
    <processing key="GHI" summarization="HOURLY" terrainShading="true">
    </processing>
    </ws:dataDeliveryRequest>'''
    api_key = 'demo'
    url = 'https://solargis.info/ws/rest/datadelivery/request?key=%s' % api_key
    headers = {'Content-Type': 'application/xml'}
    with requests.post(url, data=request_xml.encode('utf8'), headers=headers) as response:
        print(response.text)
        # parse and consume successful response, or inspect error code and a message from the server
```

In real production environment you will automatically modify XML request in run-time (e.g. changing location, period etc.). You can do this by using of XML request templates when only particular data will by replaced (e.g. lat, lng, name, dateFrom). In such case the python native The ElementTree XML API can be helpful for XML manipulation. Creating new XML requests from scratch can be easier by using some "XML data binding" technology. First you generate python objects from Solargis XSD schema documents. Then you can use the python objects for marshaling (serializing python objects into XML text) and unmarshalling (deserializing XML text into python objects) either for the request or response. The PyXB package can be used (http://pyxb.sourceforge.net/).

PvPlanner web service (API for getting long-term average data)

```
import requests

if __name__ == '__main__':
    request_xml = '''<calculateRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:geo="http://geomodel.eu/schema/common/geo"
          xmlns:pv="http://geomodel.eu/schema/common/pv"
          xmlns="http://geomodel.eu/schema/ws/pvplanner">
        <site lat="48.612590" lng="20.827079">
            <!-- optional terrain data -->
            <geo:terrain elevation="246" azimuth="176" tilt="3.1" />
            <!-- optional custom horizon data can replace the natural terrain horizon -->
            <!--<geo:horizon>11.11:18.0 7.5:15.53 15.0:10.94 22.5:10.59 30.0:13.06 37.5:14.47 45.0:14.47 52.5:
13.76 60.0:12.35 67.5:11.29 75.0:8.12 82.5:4.59 90.0:1.41 97.5:0.35 105.0:0.35 112.5:0.35 120.0:0.35 127.5:0.35
135.0:0.0 142.5:0.0 150.0:0.35 157.5:1.41 165.0:2.47 172.5:2.47 180.0:2.82 187.5:3.18 195.0:2.82 202.5:2.47
210.0:2.47 217.5:2.47 225.0:3.18 232.5:3.18 240.0:2.47 247.5:2.12 255.0:2.12 262.5:2.82 270.0:3.88 277.5:6.71
285.0:8.47 292.5:10.24 300.0:11.29 307.5:12.71 315.0:14.12 322.5:15.53 330.0:16.24 337.5:16.94 345.0:17.29
352.5:17.29</geo:horizon>-->
            <pv:geometry xsi:type="pv:GeometryFixedOneAngle" azimuth="175" tilt="45"/>
            <pv:system installedPower="1" installationType="ROOF_MOUNTED" availability="99">
                <pv:module type="CSI">
                </pv:module>
                <pv:inverter>
                    <pv:efficiency xsi:type="pv:EfficiencyConstant" percent="97.5"/>
                </pv:inverter>
                <pv:losses dc="5.5" ac="1.5"/>
            </pv:system>
        </site>
    </calculateRequest>'''
    api_key = 'demo'
    url = 'https://solargis.info/ws/rest/pvplanner/calculate?key=%s' % api_key
    headers = {'Content-Type': 'application/xml'}
    with requests.post(url, data=request_xml.encode('utf8'), headers=headers) as response:
        print(response.text)
        # parse and consume successful response, or inspect error code and a message from the server
```

Security notes

SSL connection certificate

Production WS endpoint (https://solargis.info/ws) is secured by a certificate signed by globally trusted certification authority DigiCert - more info here

No additional configuration is needed, because DigiCert root certificates are trusted by default in most environments.

Web Services Security

WS request is validated against WS-Security standard `UsernameToken` with password digest and nonce and `Timestamp` element.

Example SOAP message header with security token:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.
xsd" soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="UsernameToken-2"
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:Username>demo</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
#PasswordDigest">Kpr8NgKlQI4wE7Ayo9jm0Z2+hIA=</wsse:Password>
        <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary">g9t2BO4uHOUQoFH5ugQIng==</wsse:Nonce>
        <wsu:Created>2011-06-21T11:37:38.771Z</wsu:Created>
      </wsse:UsernameToken>
      <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
xsd" wsu:Id="Timestamp-1">
        <wsu:Created>2011-06-21T11:37:38.755Z</wsu:Created>
        <wsu:Expires>2011-06-21T11:42:38.755Z</wsu:Expires>
      </wsu:Timestamp>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
   <!-- Request Message -->
  </soapenv:Body>
</soapenv:Envelope>
```

Namespaces

Be careful to define all needed namespaces for request message