Retrieval of IMCCE/Miriade ephemeris from DaCHS Resource Descriptor

The resource descriptor (RD file) can include procedure coded in python. We test here a access to a remote webservice at IMCCE to retrieve ephemeris parameters for a given target, observer and observation date.

As the IMCCE/Miriade web services are using SAOP interface, we use the ZEEP python package to query the service. The IMCCE /Miriade web services interfaces are described on each service page on the Miriade portal.

The example presented below is used in a prototype service built for IRTF (Mauna Kea Observatory telescope) NIR images in support of the NASA Juno Mission. You can try out the service following instructions available here.

Procedure Definition

The definition of the procedure in the RD file is done using the `<procDef>` element. The next code snippet is used for IRTF observations of Jupiter.

```
<resource schema='irtf_orton'>


<!--
  -- Place the <procDef> element after the <meta> elements
  -->


        <procDef type="apply" id="miriadeEphemph">

                <setup>

                        <!-- input parameters to be set with <bind> elements in <apply> element -->

                        <par key="ignoreUnknowns" description="Return Nones for unknown
                                objects?  (if false, ValidationErrors will be raised)">True</par>
                        <par key="logUnknowns" description="Write unresolved object names
                                to the info log">False</par>
                        <par key="target_name" late="True"
                                description="The observed target name (Default is Jupiter).">'p:
jupiter'</par>
                        <par key="observer" late="True"
                                description="The observer name (Default is Mauna Kea Observatory,
for IRTF).">'@568'</par>
                        <par key="obs_time" late="True"
                                description="Observation date time."/>

                        <!-- any piece of python code to be run before the main procedure, like
importing modules -->

                        <code>
                                import zeep
                        </code>

                </setup>

                <code>
                        ## This is the beginning of the core python code, indentation matters as
in python.

                        # Initializing some parameters
                        slat, slon, olat, olon = None, None, None, None
                        np_pos, phase, rap, hemis1, hemis2 = None, None, None, None, None


                        try:
                                # Initialize SOAP client using zeep module
                                client = zeep.Client('http://vo.imcce.fr/webservices/miriade
/miriade.wsdl')

                                # Setting up request parameters (as defined on service description
page)
                                # NB: Here we will use the ephemph webservice.
                                request = {'name': target_name, 'type':'', 'epoch':obs_time.
isoformat(),
                                        'nbd':1, 'step':'', 'tscale':'', 'so':1, 'observer':
observer,
```

```
                                        'mime':'text', 'view':'none', 'rv':0, 'anim':0, 'print':1,
'visu':'',
                                        'output':'--iso,--coord:eq', 'get':''}


                                # Retrieving response from webservice
                                response = client.service.ephemph(request)

                                # Each line of the output text is separated by ';' characters,
                                # and the data line is the first line not starting with '#'
                                # The next command split lines and retrieves the first data line
                                for line in (item for item in response['result'].split(';') if item
[0] != '#'): break

                                # splitting results columns
                                data = line.split()


                                olon = float(data[3])  # Sub-Observer Longitude in Jovian System
III (Sub-Earth Point)
                                olat = float(data[4])  # Sub-Observer Latitude in Jovian System
III (Sub-Earth Point)
                                slon = float(data[7])  # Sub-Solar Longitude in Jovian System III
                                slat = float(data[8])  # Sub-Solar Latitude in Jovian System III
                                np_pos = float(data[9])  # Angle between planetary North pole and
celestial North Pole
                                phase = float(data[11])  # Phase angle
                                rap = float(data[12])  # Apparent radius of target

                                # in case of APIS extension, we need to tell what is the primary
hemisphere (best view)
                                if olat >= 0:
                                        hemis1 = 'north'
                                        hemis2 = 'south'
                                else:
                                        hemis1 = 'south'
                                        hemis2 = 'north'

                        except KeyError:
                                if logUnknowns:
                                        base.ui.notifyInfo("Identifier did not resolve: %s"%
identifier)
                                if not ignoreUnknowns:
                                        raise base.Error("resolveObject could not resolve object"
                                                " %s."%identifier)




                        # Preparing output: whatever you put into the vars dictionary can be used
outside the procedure.
                        # E.g.: vars["subsolar_longitude"] is defined here, and can be used as
@subsolar_longitude outside
                        vars["subsolar_longitude"] = slon
                        vars["subsolar_latitude"] = slat
                        vars["subobserver_longitude"] = olon
                        vars["subobserver_latitude"] = olat
                        vars["np_pos"] = np_pos
                        vars["phase"] = phase
                        vars["rap"] = rap
                        vars["hemis1"] = hemis1
                        vars["hemis2"] = hemis2
                </code>
        </procDef>


<!--
  -- Any other RD stuff...
  -->

</resource>
```

Using Procedures

Then to use the procedure, we use the <apply> element, with proper configuration. The procedure is used here inside a <rowmaker> element. The <bind> elements allow to link content (constant values or computed values) to input parameters.

```
<apply procDef="miriadeEphemph">
        <bind key="target_name">'p:jupiter'</bind>
        <bind key="observer">'@568'</bind>
        <bind key="obs_time">parseISODT('T'.join([@DATE_OBS,@TIME_OBS]))</bind>
</apply>
```