

Python Integration

Embedded python from ECL code.

- Tested on Python 2.6.6
- For External Web service call, the IP should be opened from the calling server / node cluster.
- With builds prior to 4.2.2 on CentOS 5 or 6, have to add the following (or equivalent) to hpcc_setenv file (in /opt/HPCCSystems/bin)
`export LD_PRELOAD=/usr/lib64/libpython2.6.so.1.0'`

Prior to release 4.2.0, on some distros the embedded python code will not work correctly on code that contains embedded carriage-returns, which includes code saved using the ECL IDE,. But can be invoked via couple of other options.

Option 1: Using Playground from the Tech Preview / ECL Watch for testing.

Option 2: From Command Line.

Option 1: Use Tech Preview / ECL Watch from ECLWatch Web page.

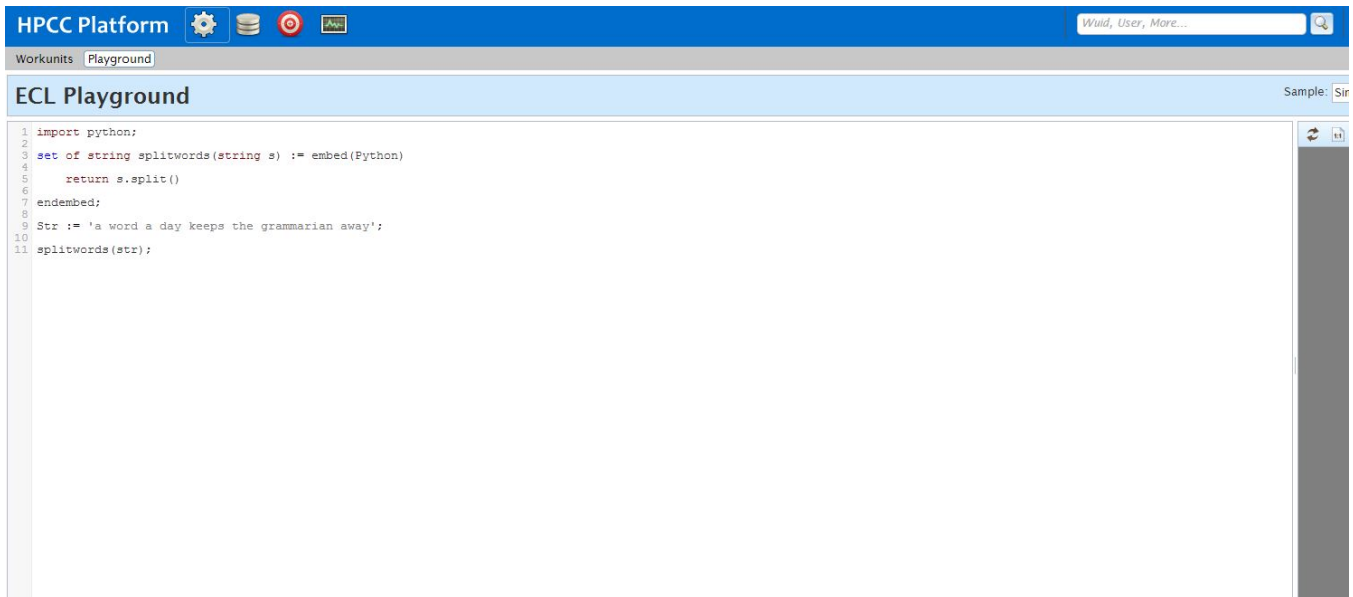
The screenshot shows the ECLWatch web interface. On the left is a navigation menu for 'HPCC Systems' with categories like Clusters, Activity, Scheduler, ECL, Queries, Topology, DFW Workunits, DFW Files, Resources, My Account, and Users/Permissions. The 'Tech Preview' and 'ECL Watch' items are highlighted. The main content area is titled 'Existing Activity on Servers' and contains a table with columns for Cluster, Active workunit, State, Owner, and Job name. The table lists several clusters such as ThorCluster, RoxieCluster, HThorCluster, ECLCCserver, and DFUserver, all showing 'No active workunit'. On the right, there is a header for 'Enterprise Services Platform' and a dropdown menu to 'Sort clusters by name ascending'.

The diagram illustrates the navigation path. At the top is the 'HPCC Platform' header with icons for a gear, database, target, and graph. A red arrow points from the gear icon to a text box that says 'Click on this and you will see the playground'. Below this is a second 'HPCC Platform' header with a 'Workunits' button and a 'Playground' button. The 'Playground' button is circled in red.

Sample Code: (Highlighted in Green is the ECL Code, and the rest is all python code):

```
import python;
set of string splitwords(string s) := embed(Python)
    return s.split()
endembed;
Str := 'a word a day keeps the grammarian away';
splitwords(str);
```

Select the target to <thor> and click on Submit.



The screenshot shows the HPC Platform interface. At the top, there is a blue navigation bar with 'HPC Platform' and several icons. Below it, there are tabs for 'Workunits' and 'Playground'. The main area is titled 'ECL Playground' and contains a code editor with the same ECL code as shown in the previous block. On the right side of the editor, there is a vertical sidebar with a 'Sample: Sir' dropdown menu.

Results: [Result 1]

[a,word,a,day,keeps,the,grammarian,away]

Option 2 from command line:

Save the ECL code in C:\temp\simple_python_embed.ecl

Compile using eclcc:

```
eclcc -E -o C:\temp\simple_python_embed.eclxml C:\temp\simple_python_embed.ecl
```

Submit workunit using eclplus:

```
eclplus query owner=<Owner> server=<Server> cluster=<Cluster> password=<Password> @C:\temp\simple_python_embed.eclxml timeout=36000
```

Results:

Workunit <WU> submitted

[Result 1]

Result_1

[a,word,a,day,keeps,the,grammarian,away]

Calling a Webservice SOAP Call from embedded ecl

Code: (Highlighted in Green is the ECL Code, and the rest is all python code)

```

import python;
set of string getSOAPResponse():= embed(python)
import sys, httplib
SoapMessage = """<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:act="" xmlns:data="" xmlns:soapenv="">
<!--soapenv:Header/-->
<soapenv:Header>
<wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="" xmlns:wsu="">
<wsse:UsernameToken wsu:Id=" ">
<wsse:Username></wsse:Username>
<wsse:Password Type=""></wsse:Password>
<wsse:Nonce EncodingType=""></wsse:Nonce>
<wsu:Created></wsu:Created>
</wsse:UsernameToken>
</wsse:Security>
</soapenv:Header>
<soapenv:Body>
<SOAP REQUEST XML>
</soapenv:Body>
</soapenv:Envelope>
"""
print SoapMessage
#construct and send the header
webservice = httplib.HTTPS("")
webservice.putrequest("POST", "")
webservice.putheader("Host", "")
webservice.putheader("User-Agent", "Python post")
webservice.putheader("Content-type", "text/xml; charset=\"UTF-8\"")
webservice.putheader("Content-length", "%d" % len(SoapMessage))
webservice.putheader("SOAPAction", "")
webservice.endheaders()
webservice.send(SoapMessage)
# get the response
statuscode, statusmessage, header = webservice.getreply()
print "Response: ", statuscode, statusmessage
print "headers: ", header
res = webservice.getfile().read()
L = []
L.append(res)
return L
endembed;
getSOAPResponse();

```

The result in this SOAP Webservice call is an XML and code snippet below uses standard ECL Library to parse the XML.

The XML is parsed using standard ECL Library code.

```

lineRec := {STRING line};
in1 := DATASET([{'<XML>'}], linerec);
outRec := RECORD
    String variable1fromresult;
    String variable2fromresult;
end;
multipleResponseRec := RECORD
    dataset(outRec) oRec;
End;
activityRec t(lineRec l) := TRANSFORM
    SELF.orec := xmlproject('<children>',
        TRANSFORM(outRec,
            self.variable1 := xmltext('variable1'),
            self.variable2 := xmltext('variable2')));
end;
newOutRec := RECORD
    String variable1 := XMLTEXT('variable1');
    String variable2 := XMLTEXT('variable2');
end;
outds := parse(in1, line, t(LEFT), XML('ROOT/PARENT/CHILD'));
Output(outds);
newOutRec tform (RECORDOF(activityRec) l, INTEGER c) := TRANSFORM
    self.variable1 := l.orec[c].variable1;
    self.variable2 := l.orec[c].variable2;
    self := l;
end;
ds := NORMALIZE(outds,LEFT.orec,tform(LEFT,COUNTER));
OUTPUT(ds);

```