Configured for #http_path#

Check these links:

- RDF directory web access: rdf/
- CGI directory web access and ExecCGI option: cgi/index.cgi

  If CGI script is not executed, you should modify your apache httpd.conf file (you must be root/sudoer).
  Then, add a ScripAlias directive :

  ScriptAlias #http_root#/cgi/ "#unix_path#/cgi/"

COPYRIGHT

DOWNLOAD

PlayMOBY tarball is available at the following address:
http://lipm-bioinfo.toulouse.inra.fr/biomoby/playmoby.tar.bz2
If you want to be informed about main updates, please send an email to Sebastien Carrere

HURRY ?

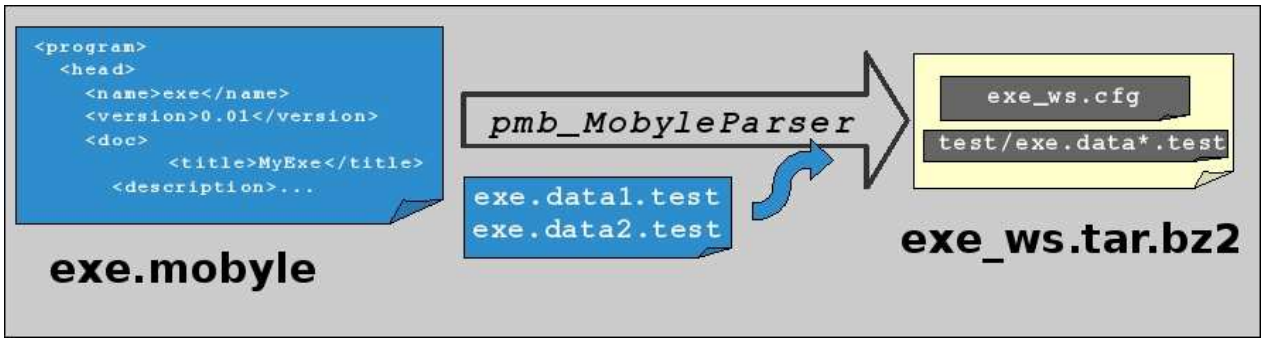
```
% tcsh
% cd /full/path/to
% wget http://lipm-bioinfo.toulouse.inra.fr/biomoby/playmoby.tar.bz2
% tar xfj playmoby.tar.bz2
% cd playmoby
% ./pmb_configure.pl --unix_path=/unix_path# --http_path=/http_path# --auth_uri=/auth_uri# --email=/mask_email# --timeout=/mask_timeout#
% source .setenv
% source .setaliases
% cd services
% $PMBBIN/pmb_MobyleParser.pl --xmlfile=/unix_path#/sample/Nicknames.mobyle --tmp_dir=/tmp --service_contact=/mask_email# --bin_fullpath=/unix_path#/sample/PrintHello
Give a test file for article Nicknames : /unix_path#/sample/nicknames.txt

WelcomeProgram.bz2 has been created.
Next step: deploy your WelcomeProgram wS using PlayMOBY environnement.
% tar xjf WelcomeProgram.bz2
% $PMBBIN/pmb_Deploy4S.pl --all --service=WelcomeProgram --central=opencentral

Opencentral presents some problems to deregister services (very long, timeout ...). I have no explanation at this moment from BioMOBY communityThat's why you will prefer to use public registry "mobycentral" to do your tests also.Nevertheless, don't forget to mention your services as testing ones (using a "test" prefix in service name for example) and overall to clean registry (deregister your test services).
```

SYNOPSIS

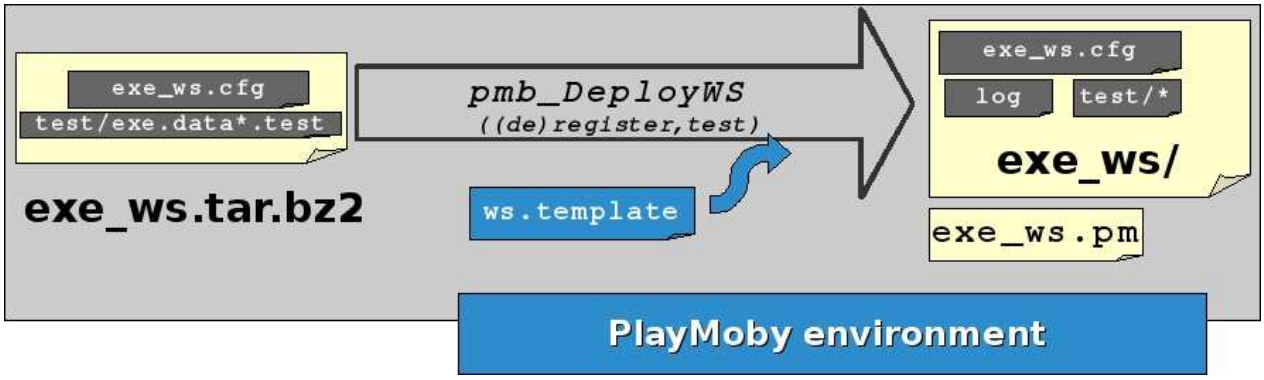
First of all, a little word on how we consider webservices: for us, it is just a script/executable wrapping. That's why our webservices are based on a system call of a standalone script handling infiles and outfiles. By this way, we can continue to use these programs using command line call, or embed them into programming pipeline or CGIs.
Deploying a webservice using PlayMOBY is a three step protocol.
The first one, is to generate/get a Mobyle description of your executable. One aim of Mobyle project (C.Letondal *et al.*) is to describe any program using XML language. As for an example, they plan to describe all EMBOSS suite programs like they have done before with *Pise*.
So we collaborate with the Mobyle team to fit our needs.
We developped a perl module called *Appli.pm* to generate easily a Mobyle valid XML file; the description of your program is embedded into the program itself and is structured into perl hashes.
The main constraints are:

- input and ouput data have to be written into files
- any message written on standard error stream will be catch as an error
- any message written on standard output stream will be lost

The second step consists in analyse the Mobyle data to principally generate a configuration file.
We choose to use a pivot format to avoid direct dependancies with Mobyle: by this way, if Mobyle specifications move in the future, we will just have to modify our Mobyle to configuration file parser without modify the third part protocol (see below).



The third part is the publication and the test of the new webservice. Here again, we choose to separate this step from the previous one in order to be more robust according to the BioMOBY's specifications evolutions.
We use two different webservice templates (synchronous one [doc]and asynchronous one [doc]) to write the perl module.
The generated webservice (the perl module and a configuration file) has to be under the PlayMOBY environment control to be available.



FIRST INSTALL

When you install PlayMOBY environnement for the first time you have to configure it.
The script *pmb_configure.pl* will modify many files (config files, perl modules) to tune PlayMOBY according to your system.
It will also check your system configuration (perl modules, binaries).

```
% tcsh
% ./pmb_configure.pl --http_path=#http_path# --auth_uri=#auth_uri# [--timeout=#mask_timeout#] [--unix_path=#unix_path#]
```

```
--http_path:    Your PlayMOBY URL
                this is the "apache path" to the PlayMOBY directory; because of webservice technology , the rest of the world should access
                to this directory using http protocol, and especially "rdf" directory and "cgi/dispatcher.cgi" should be accessible
                NB1: do not forget to add ExecCGI option onto "cgi" directory
                NB2: try to access these directories using a web browser (using http_path root URL) from an external IP class (to avoid problems of security access)

                   RDF directory: rdf/
                   CGI directory: cgi/index.cgi

--email   :    this PlayMoby instance administrator email

--auth_uri:    the authority URI is a signature for your Webservices deployed under one instance of PlayMOBY

--timeout :    the timeout parameter corresponds to the SOAP request Time out (in minutes) (LWP::UserAgent timeout) ; by default, we use 10 minutes
               there's also another timeout in your apache server, so take care to be synchronized with this one
               if you want to modify apache timeout, contact your administrator (maybe he should prefer to use a virtual host with specific timeout)

--unix_path:    the full path where you untar PlayMOBY tarball; in fact it's the directory where is "pmb_configure.pl" script
```

The script *pmb_configure.pl* also gernerates a setenv (.setenv) file containing two environment variables: PMBBIN and PMBCFG

- $PMBBIN points to the usefull PlayMOBY scripts directory #unix_path#/bin
- $PMBCFG points to configuration directory #unix_path#/cfg (which contains especially central/registry configuration files)

To use these environment variables, type the following command each time you connect to your PlayMOBY environment

```
% tcsh
% source .setenv
```

DEPLOYING A WEBSERVICE

First of all, a little word on how we consider webservices: for us, it is just a script/executable embedding. That's why our webservices are based on a system call of a standalone script handling infiles and outfiles. By this way, we can continue to use these programs using command line call, or embed these into programming pipeline or CGIs.

Deploying a BioMoby webservice using PlayMOBY environment is a 3 steps protocol:

I. Mobyle XML file generation

We based our protocol onto the Mobyle XML format to describe our programs.
To generate Mobyle-xml files we developped a perl Module called *Appli.pm*.
It's based on a description of inputs, outputs and parameters using perl hashes.
Here's a sample code (*ParamParser.pm* module is also a LIPM perl module provided within PlayMOBY tarball):

```
#!/usr/bin/perl

=pod

=head1 NAME

PrintHello - A test program who say hello and more

=head1  SYNOPSIS

PrintHello --nicknames_file=<file> --output_file=<file> [--question="<sentence to add>"] [--mobyle] [--pmbtest]

=head1 DESCRIPTION

    This program takes a list of nicknames as input and print a welcome sentence
    foreach one.
    You can provide a question to add to the welcoming message.
    The mobyle switch produce Mobyle XML data for this program.

=head1 AUTHORS

     Sebastien.Carrere@toulouse.inra.fr

=cut

BEGIN
{
    my ($dirprg, $nameprg) = $0 =~ /(.+)\/(.+)/;
    unshift(@INC, "$dirprg/../lib");
}

use strict;
use ParamParser;
use Appli;
use IO::File;
use Cwd 'abs_path';

MAIN:
{
    my $o_param = New ParamParser('GETOPTLONG', ("nicknames_file=s", "output_file=s", "question=s", "mobyle", "pmbtest"));

    my %h_appli_general = (
        name  => 'WelcomeProgram',
        descr => "This program takes a list of nicknames as input and print a welcome sentence foreach one.",
        category => 'Service',                              #from BioMoby Service Type Ontology
        authors  => 'Sebastien.Carrere@toulouse.inra.fr',
        cmd => abs_path($0),
        doclink => ['http://lipm-bioinfo.toulouse.inra.fr/biomoby']
        );

    my %h_appli_inputs = (
        'Nicknames' => {
                        descr        => "List of nicknames",
                        namespace    => '',                         #from BioMoby Namespace Ontology
                        type_biomoby => 'List_Text',                #from BioMoby Object Type Ontology
                        cmd          => '--nicknames_file=$value'
          }
          );

    my %h_appli_outputs = (
        'WelcomeMessage' => {
                             descr        => "The resulting welcome message",
                             namespace    => '',                        #from BioMoby Namespace Ontology
                             type_biomoby => 'text-formatted',          #from BioMoby Object Type Ontology
                             cmd          => '--output_file=$value'
          }
          );

    my %h_appli_parameters = (
        'a_question' => {
                         datatype     => 'string',                      #Mobyle type
                         descr        => 'A question you want to ask for',
                         type_biomoby => 'String',                      #String,Boolean,DateTime,Integer, Float
                         default      => '',
                         cmd          => '--question=\"$value\"'
          }
          );

    my $o_appli = New Appli(-general => \%h_appli_general);
    $o_appli->SetInputs(%h_appli_inputs);
    $o_appli->SetOutputs(%h_appli_outputs);
    $o_appli->SetParams(%h_appli_parameters);

    if ($o_param->IsDefined('mobyle'))
    {
        print STDOUT $o_appli->GetMobyleXml();
        exit;
    }

    if ($o_param->IsDefined('pmbtest'))
    {
        print STDOUT $o_appli->GetPlaymobyTestXml();
        exit;
    }
    $o_param->AssertFileExists('nicknames_file');
    $o_param->AssertDefined('output_file');

    my $fh_nicknames = new IO::File($o_param->Get('nicknames_file'));
    die($!) if (!defined $fh_nicknames);

    my $fh_output = new IO::File('>' . $o_param->Get('output_file'));

    while (my $nickname = <$fh_nicknames>)
    {
        chomp $nickname;
        print $fh_output "Welcome $nickname! ";
        print $fh_output $o_param->Get('question') . '?'
          if ($o_param->IsDefined('question') && ($o_param->Get('question') ne ''));
        print $fh_output "\n";
    }

    $fh_nicknames->close;
    $fh_output->close;

}
```

The Mobyle XML file produced when the program is executed with --mobyle swith is:

```
<?xml version="1.0" encoding="utf-8"?>
<program>
  <head>
    <name>WelcomeProgram</name>
    <version>0.0</version>
    <doc>
      <title>WelcomeProgram</title>
      <description>This program takes a list of nicknames as input and print a welcome sentence foreach one.</description>
      <authors>Sebastien.Carrere@toulouse.inra.fr</authors>
      <doclink>http://lipm-bioinfo.toulouse.inra.fr/biomoby</doclink>
    </doc>
    <category>Service</category>
  </head>
  <parameters>
    <parameter iscommand="1" ishidden="1">
      <name>WelcomeProgram</name>
      <prompt>command</prompt>
      <type>
        <datatype>
          <class>String</class>
        </datatype>
      </type>
      <format>
        <code proglang="perl">" #unix_path#/sample/PrintHello"</code>
        <code proglang="python">" #unix_path#/sample/PrintHello"</code>
      </format>
    </parameter>
    <parameter ismaininput="1">
      <name>Nicknames</name>
      <prompt>List of nicknames</prompt>
      <type>
        <datatype>
          <class>Text</class>
        </datatype>
        <card>1,1</card>
        <biomoby>
          <datatype>List_Text</datatype>
        </biomoby>
      </type>
      <format>
        <code proglang="perl">" --nicknames_file=$value"</code>
        <code proglang="python">" --nicknames_file=%s" % value</code>
      </format>
    </parameter>
    <parameter ishidden="1">
      <name>WelcomeMessage_name</name>
      <prompt>WelcomeMessage output file name</prompt>
      <type>
        <datatype>
          <class>Filename</class>
        </datatype>
      </type>
      <vdef>
        <value>WelcomeMessage.out</value>
      </vdef>
      <format>
        <code proglang="perl">" --output_file=$value"</code>
        <code proglang="python">" --output_file=%s" % value</code>
      </format>
    </parameter>
    <parameter isout="1">
      <name>WelcomeMessage</name>
      <prompt>The resulting welcome message</prompt>
      <type>
        <datatype>
          <class>Text</class>
        </datatype>
        <card>1,1</card>
        <biomoby>
          <datatype>text-formatted</datatype>
        </biomoby>
      </type>
      <filenames>
        <code proglang="perl">$WelcomeMessage_name</code>
        <code proglang="python">WelcomeMessage_name</code>
      </filenames>
    </parameter>
    <parameter>
      <name>a_question</name>
      <prompt>A question you want to ask for</prompt>
      <type>
        <datatype>
          <class>String</class>
        </datatype>
        <biomoby>
          <datatype>String</datatype>
        </biomoby>
      </type>
      <vdef>
        <value></value>
      </vdef>
      <format>
        <code proglang="perl">" --question="$value""</code>
        <code proglang="python">("," --question=%s" % value)[value is not None]</code>
      </format>
    </parameter>
  </parameters>
</program>
```

Nevertheless, you can use whatever you want to produce a valid Mobyle-xml file.

II. Mobyle parsing and creation of a webservice tarball

This step will produce a standalone tarball: we analyse the Mobyle-xml to generate a Webservice configuration file.
N.B: We use such a configuration file to be independant of Mobyle DTD modification: if the Mobyle file change, we just have to modify our Mobyle parser and not all the downstream programs (using our configuration file).
This step leads to the creation of tarball containing the configuration file but also data to test the webservice and a log file.
To produce this tarball, please use the "pmb_MobyleParser.pl" script:

```
% tcsh
% source .setenv
% $PMBBIN/pmb_MobyleParser.pl --xmlfile=path_to_xml_mobyle_file
```

The following parameters are optional (but if you do not defined them, the system will ask for)

```
--tmp_dir=<temporary directory>
        this directory should be apache writable (/tmp is okay)

--service_contact=<werbservice support email>

--bin_fullpath=<full path to the binary/script embedded in the webservice>
        this parameter is already defined in the Mobyle-xml file, but we redefine it here because you could imagine
        to generate the Mobyle file on a serverA and then deploy your webservice on serverB; in this case,
        there is a chance that the executable is not in the same path.

--async whether or not to deploy your web-service as an asynchronous one (Perl module WSFR::Lite is mandatory)
```

Then the program will ask you for test data for each input. You just have to give raw data an PlayMOBY will convert it into BioMoby XML format.

III. Webservice [de]registration and test

Once the tarball is generated, you have to go into PlayMOBY/services directory.
First of all you have to make a directory where you will store your webservices.
A good practice is to create a directory by service type: for example you can create a *proteic* directory to store your proteic analyses webservices and a *nucleic* directory to store your services dealing with nucleic sequences.
Another good practice is to create a directory by user in the case of a multi-user project.

Then untar your webservice tarball in this new directory.

```
% tar jxf myWebService.bz2
```

The final step is to register and test your webservice. You have to choose in which registry you want to register your service.
A good practice is to first register it in the *opencentral* registry which is a test one. Once all the tests passed, you can deregister your service from *opencentral* and register it in the public production one called *mobycentral*.

To register, deregister and test your webservice, please use the "pmb_DeployWS.pl" script:

```
% tcsh
% source .setenv
% $PMBBIN/pmb_DeployWS.pl
```

```
      The mandatory parameters are :

              --services=<service name>
                      this can be the service directory or the service configuration file "myWebService.cfg" located in the service directory;
                      You can provide a list of services (separator = space)
                      Example: #unix_path#/services/WelcomeProgram

              --central=<central name>
                      this is the central name where the service is/will be registered;
                      By default the public central (mobycentral) is used but we provide also the official test registry information (opencentral)


              --register      : this switch will call the register procedure leading to the registration into the central and the production of a rdf file
or/and        --deregister    : this switch will call the deregister procedure (empty the rdf file, then call the Rdf-Agent)
or/and        --test          : this switch will call the test procedure, i.e. execute the webservice with the test data you provided during step II
                                if you want to use special parameters for your webservice (secondary articles), you have to create a file
                                called parameters into the myWebService/test directory; the syntax of this file is

                                      parameter_name1=value1
                                      parameter_name2=value2
                                      ...


or            --all           : consist in a sequential call of deregister + register + test

or            --pm            : only produce the myWebService.pm file
```

This script will first generate the perl module corresponding to your webservice (myWebService.pm). This module contains few subroutines among which is **myWebService**, the effective function called by the dispatcher.
The test procedure will execute your webservice; then log messages are written in the **myWebservice/log** file.
If you want to stop logging your services, just create a log.stop file in your webservice directory (touch myWebservice/log.stop) .

MONITORING SERVICES

To monitor the availability and stability of your webservices, you can use the program *pmb_CheckWS.pl* within a *cron*.
This program checks:

- RDF file availability
- Central registration
- Webservice answer

There are two ways to test the webservice answer produced with the test inputs (and parameters):

- If you place a ***myWebService.pmb.t*** into the webservice test directory, then *pmb_CheckWS.pl* will use this defined test
  In order to help you in generating such a test file, please have a look to *Appli.pm*. Nevertheless, it looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<playmoby:test>
  <playmoby:service name="WelcomeProgram">
    <playmoby:article name="WelcomeMessage">
      <playmoby:format><![CDATA[

#these are default tests for format consistancy
#you can/should write your own test
#your test MUST return a boolean and a free text

return (0, "empty article") if ($_ eq '');
return (1, "ok");
]]></playmoby:format>
      <playmoby:content><![CDATA[

#these are default tests for content analysis
#you can/should write your own test
#your test MUST return a boolean and a free text

return (0, "empty article") if ($_ eq '');
return (1, "ok");
]]></playmoby:content>
    </playmoby:article>
  </playmoby:service>
</playmoby:test>
```

- Else, *pmb_CheckWS.pl* will compare the service output to a reference one (*myWebService/test/myWebService.xml*). If this file does not exist, the first execution of *pmb_DeployWS --test* or *pmb_CheckWS.pl* will produce it

To use this script :

% ./pmb_CheckWS.pl

```
--ws_dir=<directory>
       directory where are located webservices

--central=<central name>
       this is the central name where the service is/will be registered;
       By default the public central (mobycentral) is used but we provide also the official test registry information (opencentral)

--nomail    :   this switch makes the program only produce xml report without sending any mail (EXCEPT FOR ERRONEOUS SERVICES)
               If this switch is not specified, all service developpers will receive an email foreach one of the services they deployed.
               Nevertheless, the PlayMOBY administrator (see PlayMOBY/cfg/CheckWS.cfg) will receive an email with the URL of the global report.

--bioworkflow   : to generate BioWorkFlow consortium report
```

This script produces 3 XML reports.

- a native PlayMOBY format (dtd)
- a BioWorkFlow XML format (dtd)
- a BioWorkFlow RSS feed

You will find samples of such files at http://lipm-bioinfo.toulouse.inra.fr/biomoby

DEPENDANCIES

PlayMOBY is mainly written in perl and use some linux binaries.

I. Perl Modules

- UNIVERSAL qw(isa)
- version
- Carp
- CGI
- Cwd
- Data::Dumper
- DBD::mysql
- DBI
- Error [distributed with PlayMOBY]
- Error::IOException [distributed with PlayMOBY]
- Exporter
- IO::Dir
- IO::File
- File::Basename
- File::Copy
- File::Listing
- File::Spec
- File::Which [distributed with PlayMOBY]
- Getopt::Long
- Getopt::Std
- HTML::Entities
- MIME::Base64
- URI
- URI::Escape
- LS::ID
- LS::Service::Fault
- LS::Service::Response
- WSRF::Lite [if you want to deploy asynchronous webservices]
- LWP [distributed with PlayMOBY]
- MOBY [distributed with PlayMOBY]
- Scalar::Util
- WSRF::Lite 0.8.2.2
- SOAP::Lite
- Text::Shellwords
- XML::LibXML
- XML::Twig
- XML::RSS [distributed with PlayMOBY]

- String::Diff [distributed with PlayMOBY]

NB: We distribute some of them within PlayMOBY archive for strong dependancies reasons (compatibility of releases).
Moreover we distribute perl Modules developped by our team at LIPM:

- Enum
- General
- LipmError
- LipmError::ParameterException
- LipmError::IOException
- LipmError::ObjectException
- Appli
- OntologyMap and its dictionnary
- Biomoby::Object
- Biomoby::Article
- Biomoby::PrimaryArticle
- Biomoby::SecondaryArticle
- Biomoby::Service
- BiomobyUtils
- MobyleElement
- MobyleParser
- PmbTest
- ParamParser
- Auth (authentication perl module)

II. Linux binaries

- apache web server
- xsltproc
- find
- Mail
- cat
- tar
- bzip2

SAMPLES

Here is a list of sample scripts that can be deployed as web services

- PrintHello *This program takes a list of nicknames as input and print a welcome sentence foreach one.*

- PrintHello_secure *This program takes a list of nicknames as input and print a welcome sentence foreach one.It uses perl module Auth.pm to identify the user if provided, and modify output in consequence.*

- PrintHello_Async *This program takes a list of nicknames as input, sleeps 60 seconds and prints a welcome sentence foreach one.*
- msf2fasta *Converts Msf file into a Fasta file*
- msf2phylip *Converts Msf file into a a Phylip interleaved file*
- tab2fasta *Converts tabulated file into a Fasta file*
- compare_list *Compares two lists of words*
- blastp_swissprot *Blastp against Swiss-Prot database sample*
- a_simple_blastp *Blastp against Swiss-Prot or Enzyme database sample*
- fastaCollection2multifasta *Merge of a list of fasta files into one (UseCase to deal with input collection)*
- multifasta2fastaCollection *Split of a multi-fasta file into a list of single fasta sequence files (UseCase to deal with output collection)*

AUTHORS

Main contributors are members of the LIPM Bioinformatics team:

- Sebastien Letort (ANR LEGOO Genoplante 2006) @ *Sebastien.Letort(at)toulouse.inra.fr*
- Sebastien Carrere @ *Sebastien.Carrere(at)toulouse.inra.fr*
- Jerome Gouzy @ *Jerome.Gouzy(at)toulouse.inra.fr*

CONTACT

For bug report, information request, please contact Sebastien Carrere
Sebastien.Carrere(at)toulouse.inra.fr