CEF Programmer's Guide

This document describes how to use the current set of CEF libraries & modules. It is intended for developers that would either like to integrate their website using CEF or would like to implement the core CEF protocol on some other platform or language.

An overview of CEF can be found in the CEF Technical Overview document.
A business applicability discussion of CEF can be found in the CEF Scenarios document.
A description of the core CEF protocol can be found in the CEF Protocol document.

NOTE: The code snippets presented in this document are in the python programming language. But they are fairly high level and hence the reader does not need a very intimate understanding of the python programming language.

This document is organized as follows :-

CEF Modules

CEF consists of the following main modules. These in turn invoke other modules as required. These modules can be downloaded from the sourceforge website

- CEFSend library - this library provides the functionality to do the actual sending of a user's session and transferring the user's browser to the other side.
- CEFReceive library - this library provides the actual functionality to receive the user session transferred via the browser by the CEFSend cgi.
- CEFSend cgi - This cgi module is a wrapper , that receives inbound CEF transfer requests from the browser , marshalls the information needed and invokes the actual CEFSend library above. The implementation of this CGI is considered just a sample and it can be tweaked as needed by the local site.
- CEFReceive cgi - This cgi module is a wrapper, that receive inbound CEF transfers from the portal via the browser. It marshalls the information it receives from the web server and then invokes the core CEFReceive library to do the actual receive processing.
- CEFPortal - provides the ability to send and receive messages to and from the portal. This is used for backend communication with the portal.
- ApacheAPI session manager / content protector - this c module plugs into the Apache Web Server and provides the ability to protect a static web site. It can receive session transfers from a CEFSend module. It can also throw events and send messages to the portal's CEFPortal module

Sending a User to a Remote Server

Sending a user and their session from the portal to the remote server requires invoking CEFSend on the portal (which the user would do by clicking on a link). Below is an explaination of the various steps in the process.

Instantiating CEFSend

Before instantiating an object of the CEFSend Python module, you have to make sure that you have the value for at least of one the following parameters:
- target_app_id
- target_app_url
- target_cef_url

The two most common ways of using CEFSend, is to either instantiate it with the *target_app_id* only, or with both *target_app_url* and *target_cef_url.*

If you supply *target_app_id* to CEFSend, it will figure out the target server's address based on the *AppId2ServerIdIni* file. If the application happens to be on the current server, the application's URL will be looked up from *AppId2LocalUrlIni*. This situation is useful when you want to keep track of which application is on which server and what the local URL is for each application.

Example 1:
send = CEFSend.CEFSend(target_app_id=music_101)

Example 2:
```
send = CEFSend.CEFSend(
        target_cef_url="http://server/receive.py",
        target_app_url="/music_101/index.html")
```

Sending User-defined Arguments

There are several ways to pass user-defined arguments to the remote server:
- through the constructor
- through the add_args() method
- through the add_dict() method

Example 1:
When instantiating CEFSend, you can either pass the user-defined arguments as a dictionary, as part of the *params_dict* parameter, or as key/value-based arguments:

```
Send = CEFSend.CEFSend(target_app_id='music_101',
                    arg1='value1',
                    arg2='value3',
                    ...)
```
or
```
dict = {'arg1': 'val1', 'arg2': 'val2'}

send = CEFSend.CEFSend(target_app_id='music_101',
                    params_dict=dict)
```

Example 2:
You can also add extra arguments by using either the *add_args()* method which can take a variable number of key/value parameters or the *add_dict()* method which takes the key-value pairs from the given dictionary (note that the module currently only handles values that are scalars or arrays/lists).

send.add_args(arg1='val1', arg2='val2')
or

send.add_dict(dict)

Other, miscellaneous operations

In case your program needs to know whether the target URL is on a remote server, you can use the *is_remote_server()* method and take any special actions. This is usually necessary because if you're transferring to another server, you will also have to pass along some information (such as the session ID) that the other server can use to identify the user with.

```
if send.is_remote_server():
  send.add_args(sid=session.get_sid())
```

Before completing the transfer, you can use the *get_dest_url()*method to log the destination URL for audit or debugging purposes:

```
log_write("transferring to " + send.get_dest_url())
```

Completing the Transfer

The transfer through the browser is initiated with the *transfer()* method. Make sure that your program does not emit any HTTP headers otherwise the output of this method will just display in the browser instead of redirecting it to the target server. After calling transfer(), make sure to cease execution of the current program without sending any output to the browser.

Example 1:

```
send.transfer()
return # or sys.exit(0) – stop execution
```

CEF Send Parameters

CEF Send module can take the following parameters when it is invoked

Attribute Name	Description	Required
target_app_id	Contains the application ID that the module should redirect to. (Either this parameter or "target_app_url" must be supplied")	yes
target_app_url	If supplied, it will overide the URL for the application ID specified in AppId2LocalUrlIni.	
target_cef_url	If supplied, it will override the "target_cef_url" specified in the target server's configuration file.	
source_app_id	Contains the ID of the application that's initiating the transfer. Useful when redirecting to the login program so that after authentication, the user will be	

	redirected back to this application.	
source_app_url	Used for the same purpose as “source_app_id”. It overrides the URL for the application ID specified in AppId2LocalUrlIni	
source_cef_url	Used for the same purpose as “source_app_id”. It overrides the “target_cef_url” specified in the target server’s configuration file.	
server_from	If supplied, it overrides the “id” parameter in cef.ini.	
proxy	It must be set to true on the receiving web site of the transfer	
external_redirect	By default, this flag is on and the module always does an external redirect. If it’s set to false, an internal redirect will be performed but only if the target server is the same as the originating server.	
http_post	By default, this flag is off and CEFSend does an HTTP GET. If the target URL becomes too large because of too many parameters, this module will automatically switch to using HTTP POST. You can force HTTP POST by setting this flag to true .	
send_source_cef_url	By default, this flag is off. If you turn it on, the module will send the current server’s source_cef_url to the target server.	
call_local_receiver	By default this flag is off. If you turn it on, local transfers will be forced through the cef_receive program. Normally, during local transfers, the target application is directly invoked. This flag should only be used under special circumstances.	

CEF Send Methods/ APIs

These represent the methods that the CEF Send library supports. Refer to the CEF Protocol discussion for an HTTP/XML format of the CEF APIs

transfer():
Does the transfer to the requested application through the HTTP header.

get_dest_url():

Retrieves the destination URL. Useful if the invoking program wants to know where the user will be transferred to.

is_remote_server():
Returns true if the destination URL is not located under the current server & port.

add_args():
Adds the passed arguments to the destination URL

add_dict():
Adds the contents of the passed dictionary to the destination URL as key/value pairs.

get_target_cef_url(server_id):
Return the requested server's target CEF URL by looking up its internal mappings / config files

Sample CEF Send cgi

This sample CEFSend cgi implements the steps mentioned above to provide the complete CEFSend functionality. The complete source can be downloaded from this link at sourceforge

It reuses the functionality provided by CEFSend library above and invokes it based on the parameters that it received. Programmers are not forced to used this CGI program for CEF. Anyone can embed the CEFSend library module in their programs and thereby eliminate the overhead of starting another CGI program to do the transfer.

In general, this program goes through the following steps:

1) Collect the passed parameters. Make sure *target_app_id* was passed.

2) Invoke the Session Manager module to make sure the user is authenticated and can access the requested application:

```
import SessionMgr
session = SessionMgr.SessionMgr(target_app_id,
                    args_to_login_pgm=param_dict(if any))
if not session.validate_authentication():
  return
```

3) Instantiate the CEFSend. module

```
import CEFSend
send = CEFSend.CEFSend(target_app_id=app_id,
              params_dict=args_passed_as_dict)
```

4) If the transfer is to a remote server and "send_sid=1" was passed, add the session ID to the argument list:

```
if send.is_remote_server() and args['sid'] == 1:
  send.add_args(sid=session.get_sid())
  send.add_args(sessionToken=session.getSessionToken()) # broker's token
```

5) Transfer the user to the remote server:

```
send.transfer()
```

Receiving a User from a Remote Server

This section explains the steps that need to be performed on the receiver side on the remote server when a secure transfer is received

Instantiating CEFReceive

As soon as your receiving CGI script is invoked, instantiate CEFReceive without any arguments:

```
receive = CEFReceive.CEFReceive()
```

The *dict* member variable holds all the passed arguments as a Python dictionary.

Retrieve the user’s session ID:

```
sid = receive.dict[‘sid’]
del(receive.dict[‘sid’])
```

Transferring to the Requested Application locally

After you have done any necessary validation on the passed parameters, instantiate CEFSend:

```
send = CEFSend.CEFSend(params_dict=receive.dict)
```

If the target application is on a remote server, let the request pass through:

```
if send.is_remote_server():
  send.add_args(sid=sid)
  send.transfer()
  return # or sys.exit(0) – cease program execution
```

Otherwise, validate the given session ID, establish a session, and transfer to the requested application:

```
# validate the session ID & establish a local session at this
# point, then transfer to the application:
send.transfer()
return # or sys.exit(0) – cease program execution
```

· If the “form” parameter wasn’t passed, extract all the arguments passed to the program

· Make the parameter dictionary available to the invoking program through the “dict” member variable.

CEF Receive Parameters

Below are the parameters that can be passed to the CEF Receive module

Parameter Name	**Description**	**Required**
form	A dictionary containing the arguments passed to the program. (If not supplied, this module will extract all the CGI arguments)	

CEF Receive APIs

Below are the APIs that CEF Receive supports. Refer to the CEF Protocol section for the HTTP/XML representation of these APIs

transfer():
Uses CEFSend.py to transfer the user to the requested application locally as opposed to on another server.

Sample CEF Receive cgi

This CGI program contains an example implementation for using CEF to receive a user from a remote server. It reuses the functionality provided by CEFReceive library. Programmers are not forced to use this CGI script. CEFReceive.py can be embedded in any program and thus provide an easy way of CEF enabling any application.
The source for this sample can be downloaded from sourceforge

In general, this program goes through the following steps:
1) Invoke the CEFReceive.py to decode the passed parameters

```
receive = CEFReceive.CEFReceive()
```

2) Instantiate the CEFSend.py module for transfer to the local application:

```
send = CEFSend.CEFSEnd(params_dict=receive.dict, proxy=1)
```

3) If the target application is on a remote server or the login program is being invoked, do the transfer and exit:

```
if send.is_remote_server() or target_app_id == ‘login’:
  send.transfer()
  return
```

4) The requested application is on this server, create a session for the user:

```
import SessionMgr # replace this with your own session manager
```

```
*** import your session manager & profile manager over here ***
sid = receive.dict['sid']
sessionToken = receive.dict['sessionToken']

SessionMgr.write_session_file(sid, profile)
SessionMgr.create_session_cookie(sid)

# you can also add profile retrieval functions here
```

5) Local Transfer to the application:

```
send.transfer()
```

CEF Configuration

In order for CEF to work between the portal and the remote web site, some configuration work needs to be done. This configuration needs to happen both on the portal and the remote website
Below are the details of some of the configuration files and the configuration steps that may need to performed

- cef.ini – the main configuration file for both CEF & the Secure Portal
- AppId2ServerId.ini – Maps application IDs to servers (used on the sending server)
- AppId2LocalUrl.ini – Maps applications IDs to URLs on the local server (used on the receiving server)
- Servers/server_id_files – A set of files which contain the configuration for remote servers. The “server_id” is taken from the AppId2ServerId.ini file based on the given application ID.

cef.ini

This is the main configuration file for the Secure Portal and CEF.
It contains the following groups of configuration parameters:

- main - the main configuration for CEF
- cck - this group contains information used to hold the backchannel conversation with the portal
- login - for the Secure Portal login programs. contains login url and related information about the central portal
- session - timeout, format and other params applicable for the session manager & the Apache API plug-in
- url_defaults - default URLs in case only partial ones are provided to the CEF libraries

Group: main

Parameter Name	Description
Id	The current server’s unique ID. This ID is passed along as the “server_from” or “proxy_n” parameter when the current server transfers a user to another server.
AppId2ServerIdIni	Points to the AppId2ServerId.ini file. If it doesn’t contain an absolute path name, the given file will be looked up from the Portal’s data directory

	(Portal/data)
AppId2LocalUrlIni	Points to the AppId2LocalUrl.ini file. If it doesn't contain an absolute path name, the given file will be looked up from the Portal's data directory (Portal/data)
ServerDir	Points to the directory that contains the "server_id" files for remote servers. The "server_id" is looked up from AppId2ServerId.ini based on the given application ID
source_cef_url	Points to the current server's CEF receiving program (cef_receive.py). If this parameter is not set, the current server is used with the default CEF receive script's location
central_cef_url	Points to the CEF receiving program on the Secure Portal. If this parameter is set, all transfers from one remote server to another and vice versa will go through the Secure Portal. This should be the default mode of operation

Group: cck

This group provides the url and other details where CEF should hold a back channel communication. Hence it is only relevant on the remote servers

Parameter Name	**Description**
url	Points to the backchannel conversation URL. (Make sure this server/url is running and can accept http connections)

Group: login

This group defines the login url and related information for the secure portal. This group is only used & relevant on the Secure Portal and ignored on all other remote servers

Parameter Name	**Description**
login_url	The fully qualified URL for the user id/password-based login program.
cert_login_url	The fully qualified URL for the certificate-based login program.
login_tpl	Points to the login HTML template. If it doesn't contain an absolute path name, the given file will be looked up from the Portal's data directory (Portal/data)

login_wml	WML version of "login_tpl" for WAP (Wireless Application Protocol)
account_sel_url	The fully qualified url that can be used for user account selection afther the user has logged in
account_sel_tpl	Point to the account selection HTML template. If it doesn't contain an absolute path name, the given file will be looked up from the Portal's data directory (Portal/data)
account_sel_wml	WML version of "account_sel_tpl" for WAP (Wireless Application Protocol)
main_tpl	This is an optional parameter that may point to an HTML template that enforces the site's standards (such as banners, footers, etc). If it's set, the contents of "login_tpl" and "account_sel_tpl" will be merged into it as the "content" parameter using expandHTML.py

Group: session

This group is used on all servers for creating and managing local sessions. It's used by both the Python session manager and the Apache API plug-in.

Parameter Name	**Description**
timeout_secs	Indicates the number of seconds that it takes for an idle session to time out.
cookie_prefix	Prefix of the session cookie that gets embedded in the browser. The port number of the current server is appended to this name.
dir	Points to the directory that contains the session files. If it doesn't contain an absolute path name, the given directory is assumed to be under the Portal's data directory (Portal/data)
remove_secs	Idle sessions are removed after this many seconds.

Group: url_defaults

In case only partial URLs are provided to the CEF modules (for example only a server URL without a path), these parameters will be used to complete the given URL.

Parameter Name	**Description**

root	If set, it contains the name of the root directory for CEF in the browser's destination URL. For example, if this parameter is set to "pysol" and the browser's URL contains "/~user/pysol/Portal", the URL root to the Portal directory will become "/~user/pysol" and it will be prepended to all the script URLs below. This is how the same configuration file can be used for all users without any change on the development server.
login_script	Points to the default login CGI script (/Portal/cgi-bin/login.py)
cert_login_script	Points to the default certificate login CGI script (/Portal/cgi-bin/cert_login.py)
cert_receive_script	Points to the default CEF receiving CGI script (/Portal/cgi-bin/cef_receive.py)
account_sel_script	Points to the default account selection CGI script (/Portal/cgi-bin/account_sel.py)

AppId2LocalUrl.ini

This file maps application IDs (target_app_id) to URLs (target_app_url) on the local server. It's used by CEFSend.py to find out the location for a given application ID.
Location: derived from cef.ini based on the *AppId2LocalUrlIni* parameter (usually *Portal/data/AppId2LocalUrl.ini*)
Used by: CEFSend.py

Format: A key/value pair of application IDs (target_app_id) and local URLs (target_app_url).

Example:
```
music_101  = /music/music_101.html
math_301   = /match/cgi-bin/math.html
login      = /cgi-bin/login.cgi
menu       = /menu/cgi-bin/menu.cgi
```

AppId2ServerId.ini

This file maps application IDs (target_app_id) to server IDs. It's used by CEFSend.py to figure out what server a given application is located on.

Location: derived from cef.ini based on the *AppId2ServerIdIni* parameter (usually *Portal/data/AppId2ServerId.ini*)
Used by: CEFSend.py

Format: A key/value pair of application IDs and server IDs. Note that the configuration file for the server ID is looked up from "ServerDir" (specified in cef.ini)

Example:
music_101 = http_www.pybiz.com
math_301 = https_www.pybiz.com
login = https_www.pybiz.com
menu = https_www.pybiz.com

The server ID naming convention is to use the server's actual URL without any forward slashes and the colons ":" converted to underscores "_".

Server ID Files

These files contain the necessary information about remote servers that CEFSend.py can use to transfer users there. For each server ID defined in AppId2ServerId.ini, you must have a server ID file. If there's no configuration file for a server ID, the program will try to guess the value of *target_cef_url* based on some defaults and the server ID itself.

Location: Directory is derived from cef.ini based on the *ServerDir* parameter.
The file name is derived form AppId2ServerIdIni file based on the given application ID.

Used by: CEFSend.py

Configuration Parameters:

Parameter Name	Description
target_cef_url	The url of the receiving CEF program on the remote server Example: https://www.pybiz.com:4444/cef_receive.cgi

Apache API Session Mgr/ Content Protector module

The Apache API Session Mgr/Content Protector module is a C module that can be loaded into an Apache Web Server. This module provides static web sites the ability to participate in CEF Session Transfers and also do basic session management. It also can be used to throw events of interest back to the central portal. Thus it can be thought of the apache api version of the CEF Send/Receive, the CEF Portal and the session manager.

Apache Api Configuration Parameters

This section describes each parameter that is included in the Apache Server's configuration file to customize the behavior of this Apache API.module

Parameter Name	Description	Required
key_file	Points to the file that contains this server's private SSL key for communicating with the central portal. e.g. /usr/local/ssl/private/portal.key	
cert_file	Points to the file that contains this server's SSL certificate for communicating with the portal.(if the	

	key_file is set, this must be set too) e.g /usr/local/ssl/certs/portal.cert	
auth_broker_url	The fully qualified URL to post authorization messages back to the portal. The URL scheme can be either http or https (for SSL communication) e.g http://www.pybiz.com:345/cgi-bin/auth.cgi	
cred_broker_url	The fully qualified URL on the portal to post credit check kinds of messages. The URL scheme can be either http or https (for SSL communication) https://www.pybiz.com/credit/cgi-bin/credit_check.py	
bill_broker_url	The fully qualified URL on the portal to post billing messages. The URL scheme can be either http or https (for SSL communication) https://www.pybiz.com:3000/cgi-bin/billing.cgi	
public_url_start	A list of URL prefixes. Any access to a URL that begins with any of these prefixes will not be protected. For further info, see the end of this section. (this parameter cannot be used with protected_url_start) /free_contents/ /index.html	
public_url_end	A list of URL endings (extensions). Any access to a URL that ends with any of these endings will not be protected. For further info, see the end of this section.(this parameter cannot be used with protected_url_end) e.g. .gif .jpg	
protected_url_start	A list of URL prefixes. Any access to a URL that begins with any of these prefixes will be protected. For further info, see the end of this section. (this parameter cannot be used with public_url_start)e.g. /order_status /services	
protected_url_end	A list of URL endings (extensions). Any access to a URL that ends with any of these endings will be protected. For further info, see the end of this section.(this parameter cannot be used with public_url_end) .doc .ppt .xls	
bill_url	A list of URL prefixes. Any access to a URL that begins with any of these prefixes will trigger a billing event to the portal regardless of the user’s session file settings. e.g. /billing_docs/ /films/billable/	
portal_data_dir	Points to a data/config directory. The following parameters will be read in from cef.ini in this directory: the session directory, the cookie prefix, and the session timeout seconds. e.g /usr/pysol/Portal/data	
no_session_url	A URL that users will be redirected to in case they don’t have a session or the session has expired http://www.pybiz.com/no_session.html	

bill_fail_url	A URL that users will be redirected to in case the billing failed for the URL they tried to access http://www.pybiz.com/bill_fail.html	
auth_fail_url	A URL that users will be redirected to in case the authorization failed for the URL they tried to access http://www.pybiz.com/auth_fail.html	

In the Apache configuration file, you'll have to specify our dynamically loadable module first with the *LoadModule* directive before specifying any of the configuration parameters. Each configuration parameter name must be preceded by the *Security* directive. Here's an example:

```
LoadModule security_module libexec/mod_security.so

Security  key_file        /usr/local/ssl/private/portal.key
Security  cert_file       /usr/local/ssl/certs/portal.cert
Security  auth_broker_url   http://www.pybiz.com:345/cgi-bin/auth.cgi
Security  cred_broker_url   https://www.pybiz.com/credit/cgi-bin/credit_check.py
Security  bill_broker_url   https://www.pybiz.com:3000/cgi-bin/billing.cgi

Security  public_url_start  /free_contents/  /index.html
Security  public_url_start  /misc/
Security  public_url_end    .gif .jpg .txt
Security  bill_url          /films/billable  /cgi-bin

Security  portal_data_dir   /usr/pysol/Portal/data

Security  skip_broker_url   http://www.pybiz.com/Portal/cgi-bin/auth_fail.py
Security  no_session_url    http://www.pybiz.com/nosession.html
Security  bill_fail_url     http://www.pybiz.com/bill_fail.html
Security  auth_fail_url     http://www.pybiz.com/auth_fail.html
```

Notes on some of the configuration parameters:

- A note on the *public_url_start*, *public_url_end*, *private_url_start*, and *private_url_end* parameters. If none of these parameters are specified, it means that the Apache plug-in will protect the entire server. If you don't want anything to be protected, don't load the plug-in into the server.
- If you specify only the *public* parameters, everything will be protected except what is specified in these parameters. If you specify only the *private* parameters, nothing will be protected except what is specified in these parameters.
- If the *public* and *private* parameters are used together, the *public* parameters take precedence over the two *private* parameters. Any URLs specified in *public_url_start* or *public_url_end* will not be protected. If a request comes into a URL whose prefix is specified in "protected_url_start" and whose postfix is specified in *public_url_end*, the *public* parameter will take precedence and the user will be allowed to view the content. To see how the plug-in processes these parameters, see step #2 in the Steps Inside the Apache API section.
- The URL for the *no_session_url*, *bill_fail_url*, & *auth_fail_url* parameters is only pulled from the Server's config file currently. These URLs could also be obtained from the session file or from a cookie in the browser once the exact parameter name (in the session file or the cookie) is known. In

general, any of the server configuration parameters could be obtained either from the cookie or the session file as long as their alternate location is exactly specified.

Relevant Session File Contents

This section describes each parameter that is obtained from the users' session files The session file is essentially assumed to a file on disk for each user. It contains session information about the user obtained during a session transfer alongwith any pre-configured information from the portal. The format of the file is in XML. Below is the required information in the session file. Other than this, the session file can contain any other information that is relevant to the application,. CEF does not mandate anything else to be stored or in any format within the session file.

Parameter Name	**Description**
last_access_time	Date & time the session was last accessed. The Apache API determines whether the session has expired based on this parameter & the timeout value that's specified in cef.ini. This value is updated on each server access.
auth_url	A list of URL prefixes that the user is allowed to access without invoking the portal for an authorization check. If the user tries to access a URL that's not specified here, the portal will be contacted for authorization.
bill_url	A list of URL prefixes whose access will trigger a credit check and a billing event to the portal.
notify_broker	This is a flag indicating whether the server should notify the portal about all events that take place on the user's behalf (including non-billable events)

Steps Inside the API

The following steps are performed by the module for each request that the server gets:

1. Retrieve the URL for the current request
2. Check whether the current URL should be protected. The following conditions are tested in the order given and if any of them evaluates to true, the request is allowed to pass through without any interference
    1. If *public_url_start* was specified and the current URL begins with any of the prefixes in it.
    2. If *protected_url_start* was specified but none of the prefixes in it match the beginning of the current URL.
    3. If *public_url_end* was specified and the current URL ends with any of the values in it.
    4. If *protected_url_end* was specified but none of the values in it match the ending of the current URL.

    5. If all conditions above evaluated to false, the current URL is considered to be protected.
3. Retrieve the user's session ID from the browser's cookie based on the *sid_cookie_prefix* from cef.ini (the current server's port number is appended to this parameter's value)
4. If the session ID cookie is not found, the user will be redirected to the URL that's specified in *no_session_url*.
5. Read in the session file & make sure the session is still valid
6. If the session has timed out, redirect the user to the URL that's specified in *no_session_url*.
7. Update the last_access_time in the session.
8. Check if the current URL begins with any of the prefixes that are specified in *auth_url* in the user's session file. If a match is not found, the portal will be contacted for authorization. If the portal turns down the request or a communication failure occurs, the user will be redirected to the URL that's specified in *auth_fail_url*. (if the authorization url is not configured, redirect to the URL specified in skip_broker_url)
9. Check if the current URL begins with any of the URLs that are specified in the server's or the session file's *bill_url*. If a match is found, the portal will be contacted to check the credit & to bill the user. If the portal turns down the request or a communication failure occurs, the user will be redirected to the URL that's specified in *bill_fail_url*.(if either the credit or billing portal is not configured, redirect to the URL specified in skip_broker_url)
10. All checks have been done, the user can now access the page.

Miscellaneous Notes

It's assumed that the user's session file is in the format that the Python session manager (SessionMgr.py) keeps it – in XML with the first 16 bytes in the file reserved for the last access time.

Request Examples

This section gives an example of the XML structure for each request type. Of course this request type will change depending on the type of application using it and their needs. In this case the samples are shown in the biztalk format, with a header envelope containing a body. The header contains routing information like the server from which the message was sent and the server to which the message is going,. This allows for automated dispatch and processing of messages on the sender and receiver side.

Authorization Request

```
<?xml version="1.0"?>
 <biztalk xmlns="urn:schemas-biztalk-org:biztalk-0.81.xml">
 <body>
   <interaction xmlns="urn:schemas_authorize.xml">
     <context xmlns="urn:schemas_authorize.xml">
       <version>00-00-01</version>
       <timestamp>947191793</timestamp>
       <timeout>45</timeout>
       <route>
         <from>
           <session>98dced86af2a93b7c393d59aaf</session>
           <user>CN=chetan_patel</user>
           <process>remote_server_A</process>
           <family>remote server A</family>
```

```
          </from>
          <to>
            <path>http://www.pybiz.com/portal/authorize/cgi-bin/authorize.py</path>
            <process>portal1</process>
            <family>cef portal</family>
          </to>
        </route>
        <request_type>authorize</request_type>
      </context>
      <body>
        <authorize xmlns="urn:schemas-portal/portal-0.50/authorize.xml">
           <billing_model>Flat</billing_model>
           <date>29.06.1999</date>
           <price>100</price>
           <name>rock music</name>
           <currency>USD</currency>
           <description>rock music flat billing service</description>
           <album>Born in the USA</album>
           <artist>Bruce Springsteen</artist>
           <provider>pybiz</provider>
           <id>2876Afc4deh</id>
           <music_url>http://www.pybiz.com/music/coolalbum.rm</music_url>
        </authorize>
      </body>
    </interaction>
  </body>
</biztalk>
```

Credit-check Request

```
<?xml version="1.0"?>
  <biztalk xmlns="urn:schemas-biztalk-org:biztalk-0.81.xml">
  <body>
    <interaction xmlns="urn:schemas_credit_check.xml">
      <context xmlns="urn:schemas_credit_check.xml">
        <version>00-00-01</version>
        <timestamp>947191793</timestamp>
        <timeout>45</timeout>
        <route>
          <from>
            <session>98dced86af2a93b7c393d59aaf</session>
            <user>CN=chetan_patel</user>
            <process>remote_server_A</process>
            <family>remote server A</family>
          </from>
          <to>
            <path>http://www.pybiz.com/portal/creditcheck/cgi-bin/creditcheck.py</path>
            <process>portal1</process>
            <family>cef portal</family>
          </to>
        </route>
        <request_type>credit_check</request_type>
      </context>
      <body>
        <credit_check xmlns="urn:schemas-portal/portal-0.50/credit_check.xml">
          <id>947191716.809</id>
          <payment>
            <units>5</units>
            <currency>USD</currency>
          </payment>
          <date>06.12.2000</date>
          <service_id>123</service_id>
        </credit_check>
      </body>
    </interaction>
  </body>
</biztalk>
```

Usage Request

```
<?xml version="1.0"?>
<biztalk xmlns="urn:schemas-biztalk-org:biztalk-0.81.xml">
  <body>
    <interaction xmlns="urn:schemas-usage-event.xml">
      <context xmlns="urn:schemas-usage-event-context.xml">
        <version>00-00-01</version>
        <timestamp>947191793</timestamp>
        <timeout>45</timeout>
        <route>
          <from>
            <session>98dced86af2a93b7c393d59aaf</session>
            <user>CN=chetan_patel</user>
            <process>remote_server_A</process>
            <family>remote server A</family>
          </from>
          <to>
            <path>http://www.pybiz.com/portal/billing/cgi-bin/BillingProxy.py</path>
            <process>portal1</process>
            <family>cef portal</family>
          </to>
        </route>
        <request_type>usage_event</request_type>
      </context>
      <body>
        <usage_event xmlns="urn:schemas-portal/portal-0.50/usage_event.xml">
          <id>947191716.809</id>
          <item_id>http://pybiz.com/billing/cgi-bin/billing.cgi</item_id>
          <units_consumed>45</units_consumed>
        </usage_event>
      </body>
    </interaction>
  </body>
</biztalk>
```