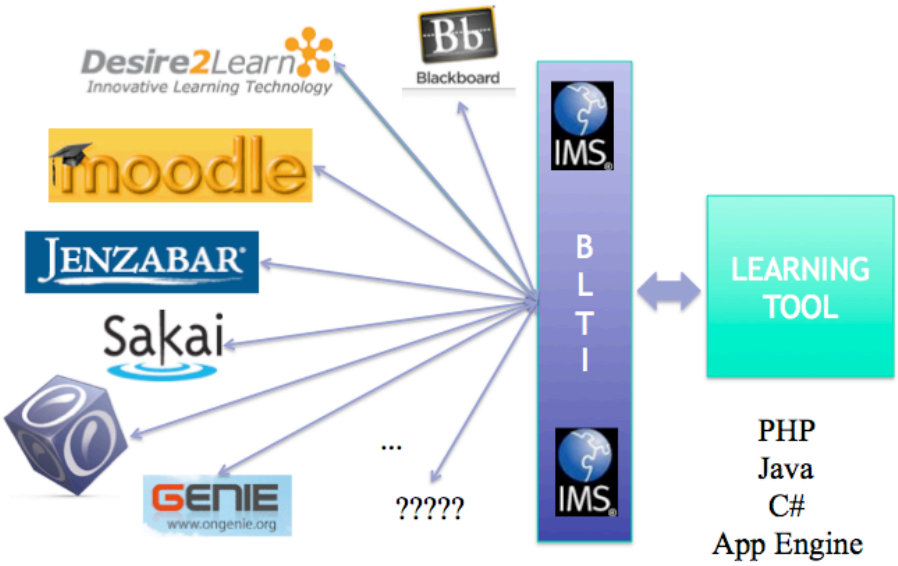
Building a Learning Application for Google App Engine

January 21, 2011
Version 0.0.1
Charles Severance (www.dr-chuck.com)

Overview

The IMS Basic Learning Tools Interoperability standard is available in nearly all of the major Learning Management Systems including Desire2Learn, Backboard, Jenzabar, Sakai, Moodle, ATutor, OLAT, Genie, and others.



Basic LTI allows you to build a learning application in the cloud and plug it into all of these Learning Management Systems with a single integration.

http://www.imsglobal.org/developers/BLTI/

In addition because Google provides free hosting for small cloud applications, using Basic LTI to build tools and host them on Google App Engine is an excellent combination. It means that all one needs to do is learn a bit of coding and learn Basic LTI and you can begin to build learning applications that you can use and easily share with your fellow teachers and students.

There are many resources to help you learn Python and the Google App Engine:

http://www.pythonlearn.com/ (Includes a free book)
http://www.appenginelearn.com/ (Book from O'Reilly and Associates)

This document assumes you know your way around Python and App Engine applications using Python.

Introduction

Pre-Requisites

You should also have Subversion (SVN) installed on your system and available at the command line since you will be getting the source out of SVN. You can install SVN on your computer from http://subversion.tigris.org/.

Getting Started

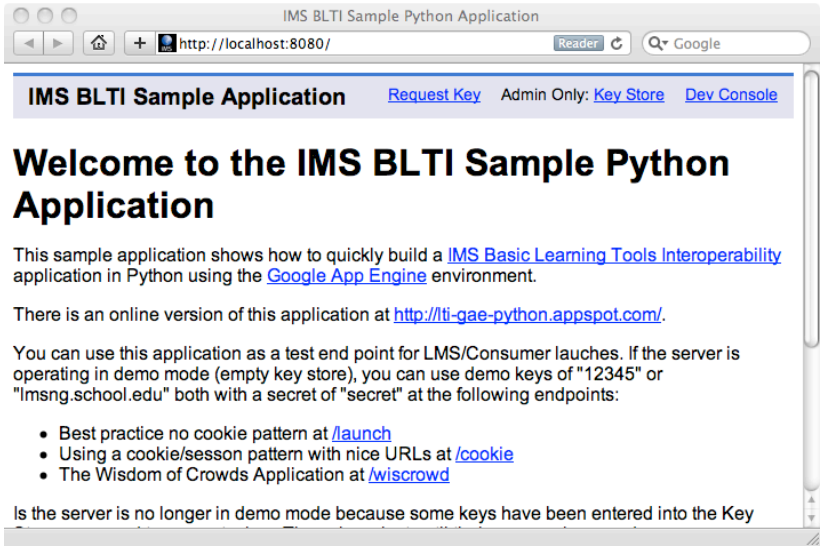
You can check out the code for the sample application from the source code repository using Subversion by typing the following in a command line interface. The checkout should produce a subdirectory called **wiscrowd** from wherever you run the checkout (co) command.

```
svn co http://ims-dev.googlecode.com/svn/trunk/basiclti/python-appengine/
```

Once the checkout completes, this is like any other App Engine application and can be started as follows using the Google App Engine run-time.[1]

```
dev_appserver.py python-appengine
```

You should be able to navigate to the main page at http://localhost:8080/.



[1] If you need to learn about programming the Google App Engine see www.appenginelearn.com

From here, you can set up a temporary key in **memcache** or a permanent key in the data store. You can only set up a permanent key if you are the owner of the application. There are also some demo keys that work until you add the first key to the data store.

There are three Basic LTI End Points in the application. The first is a tool that does not depend on cookies to maintain session (/launch), the second depends on cookies to maintain session (/cookie), and the third is the guessing game from the Wisdom of Crowds by James Surowiecki.

To Cookie or Not to Cookie?

As you write a Basic LTI application, you must make a decision as to whether or not you will use cookies to maintain session state. The best practice is not to use cookies, but this results in ugly URLs at times. However if you use session, and you launch tools in more than one window, the later launches overwrite session, causing a confusing user interface, or even introducing a security problem.

You might use a session if you are sure that your tool will be running in more than one window or more than one iframe and you will run it in its own window and you want pretty URLs.

Another disadvantage of cookies is that they may be turned off by the user or browser policies may prohibit cookies within an iframe.

Launch Setup

A session with a user starts with a Basic LTI launch from a LMS to your launch URL. This ends up in the **post** method in the **LaunchHandler** in **index.py**.

```
def post(self):
    # LTI_Launch.USE_COOKIE = True
    launch = LTI_Launch(self)
    if launch.complete: return

    if launch.loaded:
        self.response.out.write('<p>Done:'+launch.getUserName())
        self.response.out.write(' <a href="'+launch.getUrl()+
                            '" target="_new">Open</a></p>')
        val = launch.get('val',0)
        self.response.out.write('<p>Val '+str(val)+'</p>')
        launch['val'] = val + 1
    else:
        self.response.out.write('<p>This tool must be
                    launched using IMS Basic LTI.</p>')
```

The **LTI_Launch()** call establishes the launch session context. It either processes an incoming launch and establishes a session, or retrieves the

session. The **LTI_Launch()** returns launch data and some indications as to the results of the session creation or retrieval. The following are several indications that may come from the launch setup:

- **launch.complete** - This means that **LTI_Launch()** completely handled the launch. Usually this is because there was an error in the LMS data and **LTI_Launch()** has returned an error message or redirected the browser back to the LMS with an error message. If this is **True**, no further work is needed.

- **launch.error** - If True, indicates an error happened in the launch. Further detail may be found in **launch.errormsg** - it is possible for both **launch.error** and **launch.complete** to be true. If you encounter launch.error - perhaps you want to log a message.

- **launch.loaded** - If **True**, **LTI_Launch()** successfully created or re-established a launch session.

- **launch.new** - if **True**, this is a brand-new launch session that was just created.

- **launch.fromcookie** - If **True**, this launch was re-established from a session cookie. If you have requested that cookies be used to maintain session, you still need to check **launch.fromcookie** because on the first launch request from the LMS, the cookie is not set, and if you are in an iframe in a browser that stops cookies from being set in an iframe, you may never see **launch.fromcookie** as **True**. So you still should check this value on each request even if you have requested that cookies be used and handle the 'not from cookie case'.

Session Management

Once you receive your launch from the LMS and **LTI_Launch()** establishes a session, you need to make sure that you pass the session identifier forward on all requests unless you are sure the session cookie is set (**launch.fromcookie is True**).

If **launch.fromcookie** is false, then you must add the session identifier to all GET URLs and add it to all post data as a hidden field. This way we pass the session identifier from launch to launch. There are two convenience methods to help you pass the value forward.

If you need to generate a GET URL, you can use the **launch.getUrl()** call to get the current request's URL with the session identifier added as a parameter. You can also provide a URL as follows:

```
launch.getUrl(self.request.path + '?action=reset')
```

If you are creating a POST form, you can call **launch.getInputTag()** to get an HTML hidden input tag with the session id. If you put the **launch** into your template context, the following template code will pass the session forward.

```
<form method="post">
{{ launch.getInputTag }}
<p>Enter Guess: <input type="text" name="guess"/></p>
...
```

Remember that you need to take these steps even if you have requested that LTI_Launch() use cookies because the browser may not accept the cookie. Here is a bit of sample code that shows the general idea:

```
if launch.fromcookie:
    self.response.out.write('<p>(Cookie is Working) ')
    self.response.out.write('<a href="'+self.request.url+
                            '" target="_new">Open</a></p>')
else:
    self.response.out.write('<p>(Cookie are off) ')
    self.response.out.write('<a href="'+launch.getUrl()+
                           '" target="_new">Open</a></p>')
```

If the current session came from a cookie, we can just launch using the same URL (pretty) and if we did not get a launch from the cookie, we launch using the same URL plus the session ID using the **launch.getUrl()** method.

The following is a cookie setting scenario when cookies have been requested and are working:

The LMS launch comes in with no cookie. Our application creates a session, and sets a cookie on the response. But since this is the first launch we do not yet have a cookie set so to be sure, we add the session id to the next GET or POST request.

When the next request comes in, we get the session ID from the GET or POST data and check the cookie - if the browser sent back the cookie and the session cookie matches the session ID in the GET or POST data, then we know we have cookies properly set, and **launch.fromcookie** is set to **True** and so we know we do not have to keep adding the session ID to the GET URL or POST data.

So we need to add the session ID information until we can verify that cookies are working.

The recommendation is not to use cookies unless your application wants pretty URLs and will run it its own window. This is why **LTI_Launch.USE_COOKIE** defaults to **False**.

You can look at the **LauchHandler** and the **CookieHandler** code in the **index.py** file to see the cookie and non-cookie scenarios.

Storing Session Data in the Launch

The **launch** variable is also a session variable and can be treated as a Python dictionary to store data from one request to another.

```
val = launch.get('val',0)
self.response.out.write('<p>Val '+str(val)+'</p>')
launch['val'] = val + 1
```

The launch data is stored in the App Engine memcache for two hours by default.

The Launch Context

When the LMS launches the tool, it provides data about the user, course, resource, and institution. Not all fields are required, so you should check the return values from these methods.

launch.isAdmin()
Is True if the current user is the administrator of this instance of CloudSocial. This person must be the administrator per the Google App Engine.

launch.isInstructor()
The current user is the instructor in the current course.

launch.getUserKey()
Returns the primary key for the user scoped by the **oauth_consumer_key**. Returns **None** if this is not available.

launch.getUserName()
Looks at the various user fields (not all will be defined) and finds a reasonable name to display. Returns **None** if this is not available.

launch.getUserShortName()
Looks at the various user fields (not all will be defined) and finds a reasonable short name to display. Returns **None** if this is not available.

launch.getUserEmail()
Returns the User's E-mail address if available. Returns **None** if this is not available.

launch.getUserImage()

Returns the avatar/profile image if available. Returns **None** if this is not available.

launch.getResourceKey()
Returns the resource link primary key for the user scoped by the **oauth_consumer_key**. The Basic LTI Launch protocol requires these fields so this should be defined. Returns **None** if this is not available.

launch.getConsumerKey()
Returns the **oauth_consumer_key**. The Basic LTI Launch protocol requires this fields so this should be defined. Returns **None** if this is not available.

launch.getResourceTitle()
Returns the resource link title. Returns **None** if this is not available.

launch.getCourseKey()
Returns the course primary key for the user scoped by the **oauth_consumer_key**. Returns **None** if this is not available.

launch.getCourseName()
Looks at the various course fields (not all will be defined) and finds a reasonable short name to display. Returns **None** if this is not available.

launch.dump()
Returns a string dumping the values in the launch enclosed in <pre> tags.