# Embedding Wookie Widgets in other Applications

## Overview and core concepts

To embed a Wookie widget in a host application you need to install (or create) an appropriate plugin for your host environment. This will provide the bridge between your host platform and the Wookie server. Currently available plugins include:

- Elgg
- Drupal
- LAMS
- Moodle
- Wordpress

If your environment is not listed above then you need to implement a plugin - don't worry, it's pretty easy, especially if we have a connector framework available for you (see below) and we're here to help.

Before we get into the details lets looks at some key terminology.

**Plugin**
A *plugin* is an extension to an existing web application that enables it to show widgets that are being served by Wookie. A *plugin* implements the Wookie REST API to discover which widgets are available, to request instances for particular users, and to set participant information.

*Plugins* are usually written in the programming language of the host web application and may make use of an existing "widget" or "plugin" system, extending it to support additional widgets made available by Wookie.

**Connector Framework**

Wookie provides a connector framework for many languages. This framework provides most of the code you need to build a plugin for your platform.

If you intend to write a plugin for your favourite host application please check http://svn.apache.org/repos/asf/incubator/wookie/trunk/connector/ for the latest available conenctor code.

Naturally, we would prefer to work together in order to build as large a set of connector frameworks and plugins as possible, so please help us improve and expand this set of connector frameworks.

**Viewer**
The *viewer* is the current user who is viewing a widget in the browser. Typically an application uses session information to know who the current user is, and this is used to request a particular *widget instance*. It is up to the *plugin* to determine how to identify the *viewer*; for example the user's real id is one possibility; another is an opaque hashcode using the id.

**Widget Instance**
A *widget instance* is a persistent instance of a particular widget created for a user. Each *widget instance* has its own storage area in Wookie. *Widget Instances* are created by invoking the Wookie REST API using an *API Key* and supplying values for the *viewer* and the *shared data key*.

**API Key**
An *API Key* is used to access many of the features of the Wookie REST API. Each individual web application needs its own *API key*. *API Keys* are generated from Wookie's administration interface.

**Shared Data Key**
The *shared data key* is an arbitrary identifier that marks *widget instances* as being sibling instances that can share state information. It is up to the *plugin* to determine this value; typically there is a persistent identifier available for whichever view is being used as the container for a widget.

## Building Plugins with the Connector Framework

The Java connector framework is used within Wookie itself and is therefore considered the reference implementation. Other connector frameworks include PHP, C#, Ruby and Python. However, event the Java framwork is incomplete and some activities are accessed directly through the REST API (see next section). We welcome your help in expanding the framework code (in any language) so that all features can be accessed through it.

See http://svn.apache.org/repos/asf/incubator/wookie/trunk/connector/ for the latest available code.

## Basic usage of the framework

In this section we document the most basic use of the connector framework. All example code is written in Java, but it should be the same in other languages (if not, help us bring them all up to date).

In general to use the framework you will need to:

- get a WookieConnectorService
- set the current user
- get or create a widget instance
- provide the instance URL to the hosting environment and display it

### Getting a connection to a Wookie server

```
/*
 * Get the wookie service connector
 */
public WookieConnectorService getWookieConnectorService(String serverURL, String apiKey, String sharedDataKey )
{
  if (connectorService == null) {
    connectorService = new WookieConnectorService(serverURL, apiKey, sharedDataKey);
  }
  return connectorService;
}
```

### Working with users

Set the current user:

```
  WookieConnectorService conn = getWookieConnectorService(url, apiKey, datakey);
  conn.setCurrentUser("testuser");
```

### Working with widgets

Get a widget instance.

```
  String guid = ".....";
  WidgetInstance instance = conn.getOrCreateInstance(guid);
```

### Displaying widgets

Displaying the widgets is the job of the platform in which you wish to embed widgets. The easiest way to embed a widget is simply to place it in an iframe. To do this you will need a URL for retrieving the widget instance:

```
  String url = instance.getUrl();
  displayWidget(url);
```

## Building Plugins without a Connector Framework

We strongly discourage you from building a plugin from scratch without first building a connector framework for your chosen language. By building a connector framework, or even partially implementing one, in your chosen programming language you will be helping others build more plugins as well as being able to share resources with others using the same connector framework.

The best way to build a new connector framework is to copy the Java one (see above). The rest of this document describes what is going on in this framework.

## Creating a widget gallery

To make it easier for users to add widgets to parts of an application, you may want to provide users with a gallery of widgets.

Wookie provides the **widgets** interface for accessing metadata about currently installed widgets. There are two approaches that can be taken:

**/wookie/widgets?all=true** returns all the current widgets available. This is the method that most plugins will want to use.

**/wookie/widgets** with no parameters returns only the widgets set as defaults for the currently set widget services. This is mostly used for some advanced plugins for authoring tools.

Each method returns metadata including widget icons, titles, categories, and descriptions. This should be sufficient for creating a gallery user interface where a user can select a widget, and your application can use the identifier of the widget to create an instance, as covered in the next section.

You can also obtain localized information about widgets by adding a **locale** parameter to the request. If appropriately localized information about widgets (including localizd icons, titles, descriptions, and license information) will be returned rather than the default content.

## Widget instance lifecycle

### Requesting an instance

Whenever a user - the **viewer** - is to be shown a widget, your application needs to make sure you have a **widget instance** to show the user. To do this you should request a widget instance at **/wookie/widgetinstances**. The format of the request can be found in the Wookie REST API documentation, however the main points to bear in mind are how to construct the request parameters:

**user_id**: the user id is an identifier associated with the current **viewer**, and not the owner of the view. This doesn't need to be a "real" identifier, and can be a hashcode or other opaque value; you just need to be consistent.

**shareddatakey**: this value is used by Wookie to identify "sibling" widget instances that can share state information where appropriate (for example, in a chat widget). To do this an application needs to have a consistent process for generating this value. Typically this is an identifier already used in the application for identifying areas of pages; for example a combination of page identifier and a block or (native) widget identifier.

Note that subsequent requests for a widget instance will always return the same instance for the same given parameters - this means that if your application does not persist the instance data you can request it with each page view. Note that an exception to this rule is locale parameters - subsequent requests for an instance using the same user id and shared data key but with a different locale will always return the same instance rather than a new instance.

The response to a widget instance request contains a URL; typically you would use this information to construct an iframe tag with the correct height and width. Generally this is all that is needed to include a widget served by Wookie.

### Adding Participants

Participants are added to a Widget Instance using the Participants REST API. The format of the request can be found in the Wookie REST API documentation, however the main points to bear in mind are how to construct the request parameters. As well as the usual parameters (API key, shared data key, widget id, user id) this method requires:

**participant_id**: An identifier for the participant in the application. This doesn't need to be a "real" identifier, and can be a hashcode or other opaque value; you just need to be consistent.

**participant_display_name**: A name to display for the participant; typically a nickname or any other name commonly displayed in the application for this participant.

**participant_thumbnail_url**: A URL pointing to the participant's icon or avatar, if they have one. This should be an absolute URL.

### Initial state information

You can populate the preferences and shared state information of a Widget Instance using the REST API. The format of the request can be found in the Wookie REST API documentation, however the main points to bear in mind are how to construct the request parameters. As well as the usual parameters (API key, shared data key, widget id, user id) this method requires parameters for the key and value being set. If you set is_public=true then the property set is in the shared state for the Widget Instance and its sibling instances, otherwise its a private preference just for the current Widget Instance.

## Advanced functionality

### Cloning widget instances

It is possible to clone a Widget Instance; this creates a copy of the Instance but with a new SharedDataKey (which you supply). You can use this to support moving widget instances from one space into another while keeping all their state information. The format of the request can be found in the Wookie REST API documentation.

### Stopping and resuming instances

It is possible to Stop and Resume a Widget Instance; the effect of this is to prevent any further storage events for the Widget Instance (e.g. preference setting, or shared state changes). The format of the request can be found in the Wookie REST API documentation.

### Sharing widgets across applications

The assumption in the documentation above is that each application will want to maintain widget states in isolation from other applications - for example, that it is not desirable to potentially mix up the states of widgets run by users in different contexts.

However, if you are careful about how each plugin issues its SharedDataKey it is possible to operate with shared widgets.

For example, it is possible to have a single shared widget state for the same group of users in two different applications if they issue a matching SharedDataKey for the context, matching UserIds for users, and use the same API Key.

Obviously this is something that needs careful attention in configuring the plugins for the applications.

## Using IMS Basic LTI to embed Wookie Widgets in educational systems

For education applications it is possible to also use Wookie with the IMS Basic LTI specification, which is a specification for connecting e-learning applications with tools. To do this you can use the BasicLTI4Wookie plugin . There is IMS Basic LTI support in popular education platforms such as Blackboard, Sakai, Desire2Learn and WebCT.