Data API

The Live Forms Data API provides a simple protocol for viewing and managing Live Forms resources such as forms, applications, themes, schemas, etc. The API extends the Google Data API framework, is HTTP and XML-based, and uses the Atom Syndication Format with a few extensions following Atom's standard extension model. Java and .Net are the language-specific wrappers around the API.

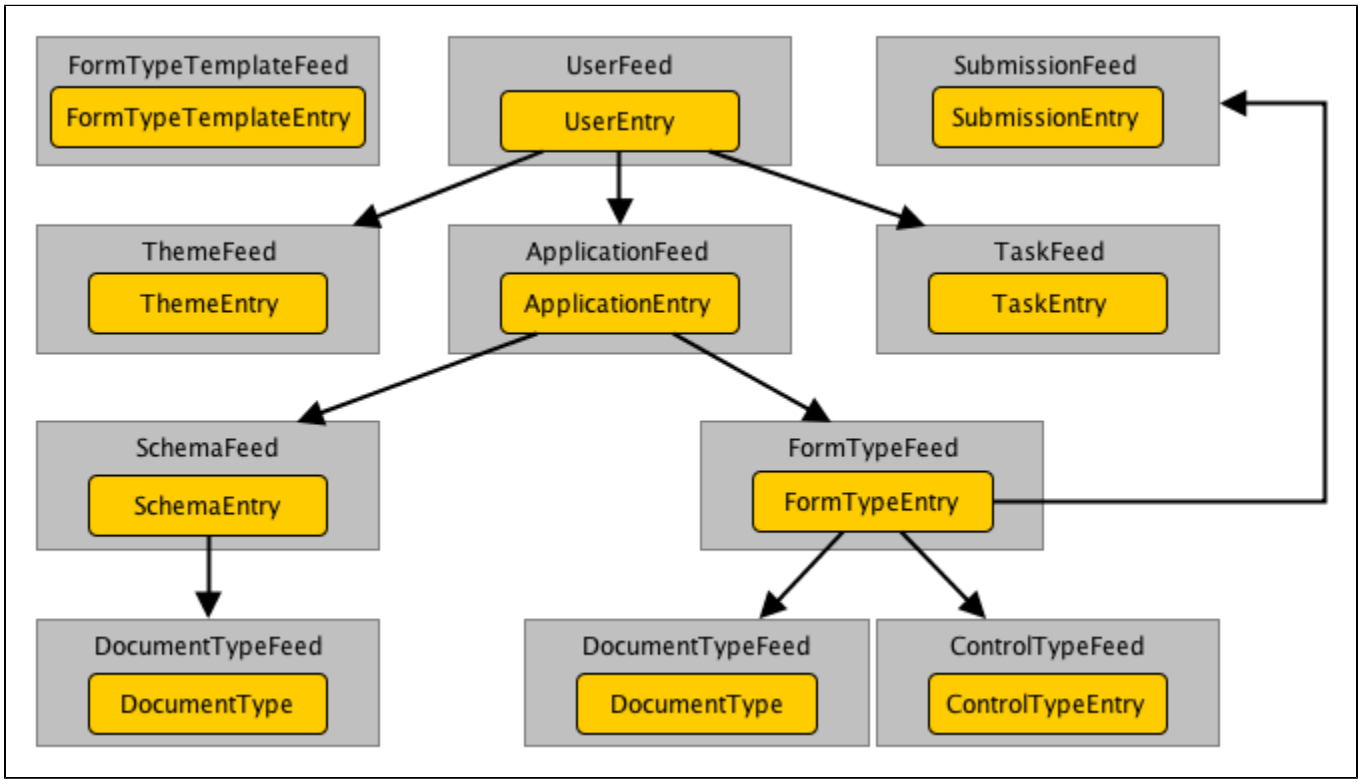
Atom also provides the Atom Publishing Protocol (APP), an HTTP-based application protocol for publishing and editing resources on the web. The APP specification is an emerging standard being developed by the IETF that allows you to send an HTTP GET request to ask for a particular resource such as a form or schema; a representation of that resource is returned in the Atom Syndication format. You can also create, edit and delete resources using standard HTTP POST, PUT and DELETE methods, respectively. Atom provides a protocol in line with the REST approach to web service interfaces.

On this page:

Data API Feeds

There are various different types of Live Forms resources that can be viewed and manipulated by the Data API. Most of these resources are things that a designer sees when creating new forms and navigating the application using the Live Forms UI, for instance users, applications, form, themes, schemas, etc. So before delving into the API specifics it is a good idea to understand these different types of resources and how they relate to each other.

Here is a diagram showing all the available Data API feeds



Here is a quick run-down of each one of them:

- **UserFeed** - There is not much that can be done with this collection at this point except for being an optional entry point for applications, tasks and themes.
- **FormTypeTemplateFeed** - The collection used to list and manage published forms or flows.
- **SubmissionFeed** - The collection used to query form or flow submissions and get associated XML documents, PDF snapshots,

attachments, etc.
- **ThemeFeed** - The collection used list and manage themes in Live Forms.
- **ApplicationFeed** - The collection used to list and manage applications in Live Forms.
- **TaskFeed** - The collection used to manage the tasks for the current logged-in user.
- **SchemaFeed** - The collection used to manage the XSDs uploaded to an application.
- **DocumentType** - The collection used to manage top-level XSD elements in Live Forms, called DocumentTypes. An XSD has a feed containing all the DocumentTypes (top level elements) that can be added to forms and flows and a form or flow has a feed containing all the DocumentTypes added (in the designer's Data Sources pane).
- **FormTypeFeed** - The collection used to manage forms AND flows. This is the core collection in the API that is used to use, design, manage, instantiate, etc forms and flows.
- **ControlTypeFeed** - The collection used to list the controls, and their metadata, contained in a form or flow.

It is important to understand this diagram since the source code for the client application maps almost one-to-one to this structure.

Tutorials

You can try any one of several Data API Tutorials to get started.

Client Libraries

Although our APIs are based on the Atom Publishing Protocol and the Atom Syndication Protocol and can be accessed on any language/platform that can interact with HTTP end points and can process XML documents, we provide a **Java Client** and a **.Net Client** that can be used to easily connect to Live Forms from Java and Windows or Mono, respectively.

You will need to install a client library in order to use the API. See Installing the Client Library and Dependencies below.

Java Client API

> The Design-time Integration tutorial is a very good place to get a quick overview of how to use the Java API.

Javadocs

If you installed Live Forms locally, the installation includes the Javadocs .jar file. The **com.frevvo.forms javafiles** are located in the <*installdrive*> :*frevvo**ext**client* directory. Many IDE's let you view Javadocs.

Java Client API FAQ

For a list of Frequently Asked question about common tasks using the API with a Live Forms server, please see FAQ - Java Client API.

Installing the Client Library and Dependencies

The Live Forms Java Client Library has the following dependencies:

- com.frevvo.forms.java-4.1.4.jar
- com.google.gdata.media-1.40.3.jar
- com.google.gdata.core-1.40.3.jar
- com.google.gdata.client.meta-1.40.3.jar
- com.google.gdata.client-1.40.3.jar
- commons-codec-1.2.jar
- commons-httpclient-3.1.jar
- commons-logging-1.0.4.jar
- google-collections-1.0.jar
- mail-1.4.1.jar
- activation-1.1.jar
- json-1.0.0.jar

For your convenience all these required jars can be found in the Live Forms Tomcat bundle in the **/frevvo/ext/client** folder. Make sure you include all of them in your classpath when adding them to your application's classpath.

```
~/frevvo-4.1.1/ext/client >ls
activation-1.1.jar                  com.google.gdata.core-1.40.3.jar      google-collections-1.0.jar
com.frevvo.forms.java-4.1.1-javadoc.jar com.google.gdata.media-1.40.3.jar      json-1.0.0.jar
com.frevvo.forms.java-4.1.1.jar     commons-codec-1.2.jar                 mail-1.4.1.jar
com.google.gdata.client-1.40.3.jar  commons-httpclient-3.1.jar
com.google.gdata.client.meta-1.40.3.jar commons-logging-1.0.4.jar
~/frevvo-4.1.1/ext/client >
```

The actual jar versions may be different depending on the version of Live Forms being used.

Authentication & Session Management

When interacting with Live Forms, an application first needs to establish a session using the Data API. This is done by using the **com.frevvo.forms.client.FormsService** class and by providing proper credentials to authenticate.

The example below is taken from the tutorial Getting Started with the Jave Data Client Library.

The **commandLoop()** method in the **Contacts**"' class below shows how this is done in the Contacts application. Note that the FormsService instance is state-full and the same instance needs to be used throughout the session (this is specially the case when rendering form urls in the browser since an API key is automatically appended and bound to the FormsService session - as soon as you logout() the API key becomes invalid). In the case of this command-line application the session spans the life-time of the executable, but in the case of a web application the FormsService instance is usually stored in the HTTP session that the user has with the application.

```
public class Contacts {
    ...
    private void commandLoop() throws IOException, ServiceException {
        // create the FormsService to hold the session with frevvo
        FormsService s = new FormsService(getProtocol(), getHost(), getPort(), null);
        try {
            // login to frevvo
            s.login(getUsername() + '@' + getTenant(), getPassword());

            // save the service to be used in subsequent interactions
            this.service = s;

            // Auto-upload the contact application, if not already there
            if (getContactForm() == null) {
            uploadContactApplication(s);
            }

            // start the interactive shell
            ShellFactory.createConsoleShell(getPrompt(), null, this).commandLoop();
         } finally {
            this.service = null;
            s.logout();
         }
    }
    ...
}
```

Since there is some overhead associated with logging in and out, you need to keep the **FormsService** instance around for as long as you need to interact with Live Forms before logging out. Logging out ensures that Live Forms will release any unneeded resources and the user count will be decremented, as the number of users is affected by your Live Forms license.

LoginAs

Live Forms supports an additional way of logging into Live Forms using the Data API: the loginAs() method. This new method allows you to login to Live Forms as any of the existing tenant users provided you can pass in the tenant's admin user and password. This is quite convenient when you want login to Live Forms using the same user that is logged into your application without having to know their password.

The following snippet shows how to login as a tenant user:

```
...
    String tenantAdmin = getUsername() + '@' + getTenant();
    String tenantAdminPwd = getPassword();
    String username = getAsUsername();
    FormsService s = new FormsService(getProtocol(), getHost(), getPort(), null);
    s.loginAs(username, tenantAdmin, tenantAdminPwd);
    ...
```

The **loginAs(String, String, String)** usage above assumes that you are logging in as a user that was previously created in the specific tenant.

When your tenant was configured with the **DelegatingSecurityManager**, you can use the overloaded **loginAs()** method to automatically create virtual users in Live Forms. For instance:

```
...
    String tenantAdmin = getUsername() + '@' + getTenant();
    String tenantAdminPwd = getPassword();
    String username = getAsUsername();
    FormsService s = new FormsService(getProtocol(), getHost(), getPort(), null);
    s.loginAs(username, tenantAdmin, tenantAdminPwd, true, null, null, null,
null,null);
    ...
```

This will automatically create a new, **non-designer** user (i.e. will be able to participate in flows but not create forms/flows), if it doesn't yet exist. If you want Live Forms to auto create a new virtual user that is also a **designer** you need to pass in the **frevvo.Designer** role when calling loginAs(). For instance:

```
...
    String tenantAdmin = getUsername() + '@' + getTenant();
    String tenantAdminPwd = getPassword();
    String username = getAsUsername();
    List<String> roles = new ArrayList<String>();
    roles.add("frevvo.Designer");
    FormsService s = new FormsService(getProtocol(), getHost(), getPort(), null);
    s.loginAs(username, tenantAdmin, tenantAdminPwd, true, roles, null,
null,null,null);
    ...
```

.Net Client API

The frevvo .NET API is a .NET language-specific wrapper around frevvo's core GData API framework. Refer to the browser URL APIs and java APIs discussed in this chapter for a list and explanation of the methods exposed in the frevvo data API framework.

Visit the Data API Client Libraries Releases compatibility matrix to download the proper library version for your installed Live Forms server.

The download is a zipfile containing the following files:

- <version>_dotNetApiRelease.zip - frevvo .NET client library
- <version>_help_html.zip - HTML documentation
- <version>_help.chm - chm formatted documentation

.Net Client API FAQ

For a list of Frequently Asked question about common tasks using the API with Live Forms server, please see the API .Net Client FAQ topic.

Browser URL API

The client libraries are built on top of the browser API. However since these URLs can be used directly from your browser, it is a great way to experiment with the Live Forms API before you start writing code.

This section describes the basic protocol used to interact with Live Forms Data APIs including examples of what Atom requests may look like, what kind of responses to expect, and so on. It is intended for anyone wanting an understanding the general idea of the format and protocol used by the Live Forms Data API and it assumes that you understand the basics of XML, namespaces, Atom feeds, and the main HTTP requests GET, POST, PUT and DELETE, as well as the RESTful concept of a resource. Using the API at this level your client application can interact with the Liv e Forms Server using any programming language that lets you issue HTTP requests and consume XML-based responses.

You can use the Resource URIs listed Browser API Reference Guide in the browser address bar when you're logged into Live Forms. The URIs return data from the Live Forms server as either web pages or XML files. To use the URIs, you edit the URL in the address bar when you're logged into a Live Forms tenant to include the URI of the resource feed you want to use. For example:

- FormTypeFeed: ../frevvo/web/tn/{tenantId/**api/formtypes?ownerId={appId}**
- FormTypeEntry: ../frevvo/web/tn/{tenantId}/**api/formtype/{id}**

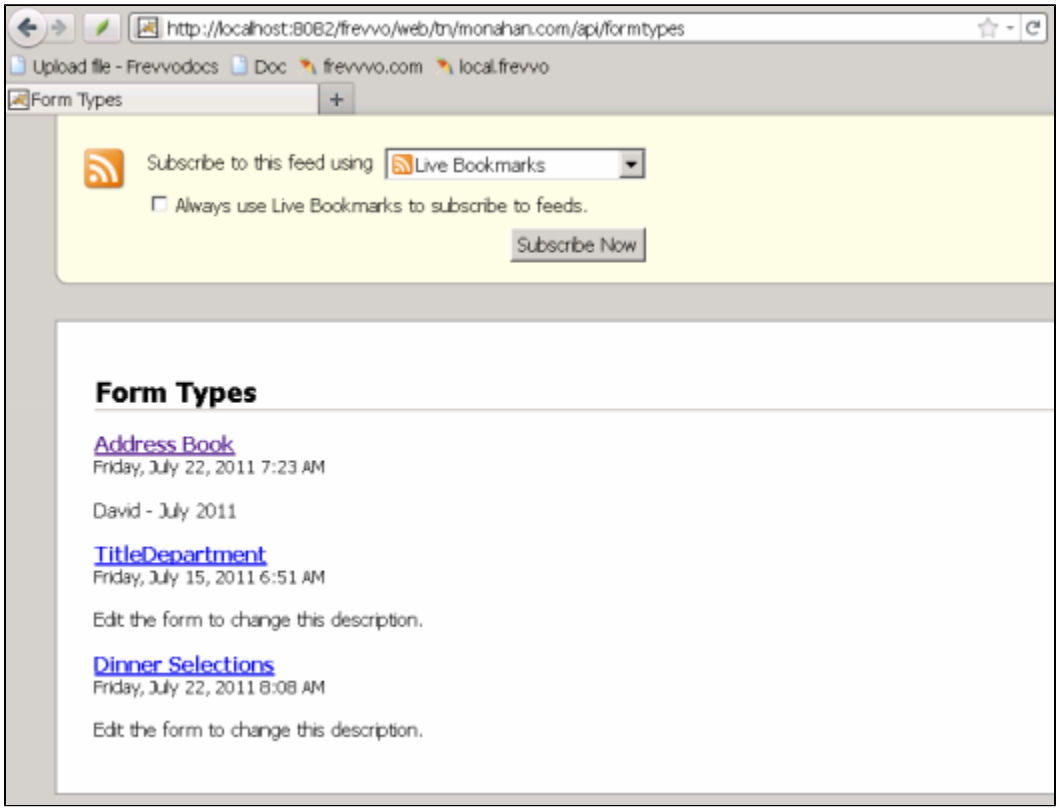
The browser URL API provides:

- A useful tool for debugging
- An easy way to try out the API without having to write code
- A way to experiment with and learn the API

This example shows how to use the **formtypes** feed to have the Live Forms server return a list of all the forms belonging to the user who's currently logged in.

1. After logging in, edit the Live Forms URL in the browser address bar to insert **api/formtypes** immediately after the tenant name - which in this example is *monahan.com*.
2. So in this example, we change the URL "from:" http: //localhost:8082/frevvo/web/tn/monahan.com **user/david/app**
3. *to*: http: //localhost:8082/frevvo/web/tn/monahan.com/**api/formtypes**, and press **Enter**.

The browser returns the page below.



If you click on the **Address Book** link and either open or save the XML file, you see the XML for the form, the first portion of which is shown below.

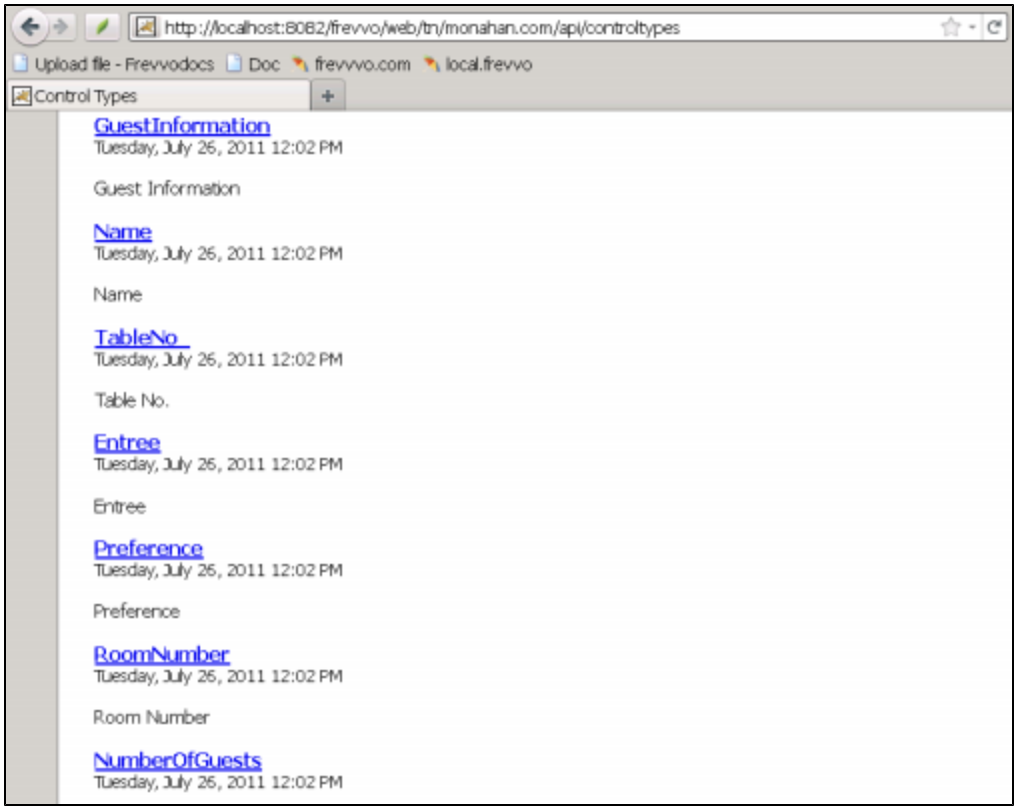
Depending on which browser you're using, the browser may return the XML rather than the page in the illustration above.

```
<entry xmlns:fd="http://schemas.frevvo.com/fdata/2008"
        xmlns="http://www.w3.org/2005/Atom"
        xml:lang="en"
        xml:base="http://localhost:8082">
    <id>_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david</id>
    <title type="text">Address Book</title>
    <summary type="text">David - July 2011</summary>
    <updated>2011-07-22T11:23:25.326</updated>
    <category scheme="http://schemas.frevvo.com/fdata/2008#kind" term="FORM"/>
    <link type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/formtype/_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK
99v93jw!david"/>
    <link rel="self" type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/formtype/_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK
99v93jw!david"/>
    <link rel="edit" type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/formtype/_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK
99v93jw!david"/>
```

This example shows how to use the **controltypes** feed to get a list of all the control types that belong to the currently logged-in user, regardless of which forms they're used in.

1. After logging in, edit the Live Forms URL in the browser address bar to insert **api/controltypes** immediately after the tenant name - which in this example is *monahan.com*.
2. So in this example, we change the URL *from:* http: //localhost:8082/frevvo/web/tn/monahan.com/**user/david/app**
3. *to*: http: //localhost:8082/frevvo/web/tn/monahan.com/**api/controltypes**, and press **Enter**.

The browser returns the page below. Scroll though the page to see all the controls in all the user's forms.

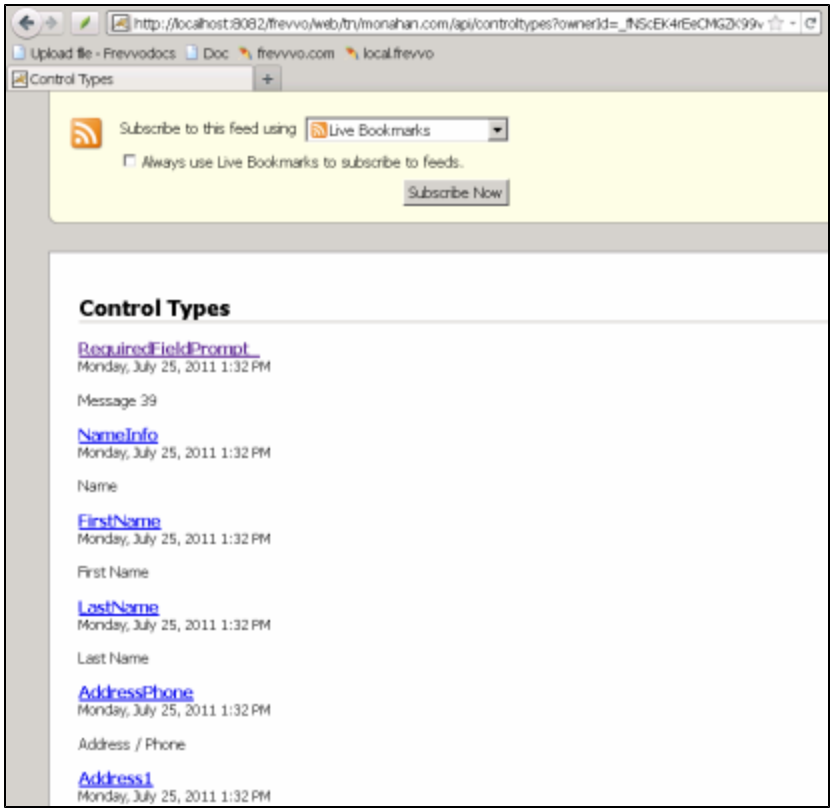


This example shows how to use the **controltypes?ownerId={formTypeId}** feed to get a list of the control types in a specific form - in this case the Address Book form in the example above.

1. After logging in, edit the Live Forms URL in the browser address bar to insert **api/controltypes?ownerId={formTypeId}** immediately after the tenant name - which in this example is *monahan.com*.
2. So in this example, we change the URL *from:* http: //localhost:8082/frevvo/web/tn/monahan.com/**user/david/app**
3. *to*: http: //localhost:8082/frevvo/web/tn/monahan.com/**api/controltypes?ownerId=_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david**, and press **Enter**.

> Note that the value of the **ownerId=** parameter is the form type definition for the *Address Book* form shown in the XML example above, including the ID (david) of the user who created the form.

The browser returns the page below.



If you click on the RequiredFieldPrompt link and either open or save the XML file, you see the XML for the control, which is shown below.

```
<entry xmlns="http://www.w3.org/2005/Atom"
        xmlns:fd="http://schemas.frevvo.com/fdata/2008"
        xml:base="http://localhost:8082">

<id>_bjLLYbIIEeClr-dcvjFaZg!_fNScEK4rEeCMGZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david</id>
    <title type="text">RequiredFieldPrompt_</title>
    <summary type="text">Message 39</summary>
    <category scheme="http://schemas.frevvo.com/fdata/2008#controltype"
            term="OutputControlType"/>
    <category scheme="http://schemas.frevvo.com/fdata/2008#displaytype"
term="Message"/>
    <updated>2011-07-25T13:52:22.017-04:00</updated>
    <fd:required value="true"/>
    <fd:readonly value="false"/>
    <link rel="parent" type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/controltype/_fQoD8a4rEeCMGZK99v93jw!_fNScEK4rEeCM
GZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david"
            title="This entry's parent control type"/>
    <link type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/controltype/_bjLLYbIIEeClr-dcvjFaZg!_fNScEK4rEeCM
GZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david"
            title="This entry"/>
    <link rel="self" type="application/atom+xml"

href="/frevvo/web/tn/monahan.com/api/controltype/_bjLLYbIIEeClr-dcvjFaZg!_fNScEK4rEeCM
GZK99v93jw!_W0jPga4rEeCMGZK99v93jw!david"
            title="This entry"/>
</entry>
```

Logging In and Out of the Forms Server

When interacting with Live Forms, an application first needs to establish a session using the Data API. See the C# code snipped below for an example.

```
FormsService service = new FormsService("http://localhost:8082", "yourappname");
    service.Login("myuser", "mypassword");

    // interact with frevvo

    service.Logout();
```

Since there is some overhead associated with logging in and out, you need to keep the FormsService instance around for as long as you need to interact with Live Forms before logging out. Logging out ensures that Live Forms will release any unneeded resources and the user count will be decremented, as the number of users is affected by your Live Forms license.

Calling the API from a JSP Page

An important thing to note about JSP pages is that the FormsService instance is state-full and the same instance needs to be used throughout the session - this is especially the case when rendering form URLs in the browser since an API key is automatically appended and bound to the FormsService session; as soon as you logout(), the API key becomes invalid. Typically, in a web application context, the FormsService instance is stored in the HTTP session that the user has with your web application.

```
<%@ page import="java.util.*,com.frevvo.forms.client.*,com.frevvo.forms.client.util.*"
%>
FormsService service = (FormsService) session.getAttribute ( "frevvo.forms.service" );
if (service == null) {
        service = new FormsService("http", "localhost", 8082, null);
        service.loginAs (request.getParameter( "username" ),
            tenantAdminUserId, tenantAdminPassword, true, null,
            request.getParameter( "firstname" ),
            request.getParameter( "lastname" ),
            request.getParameter( "email" ), null);
        }
session.setAttribute ("frevvo.forms.service", service);
```

ResetTask.jsp

The reset task API makes it possible to restart a completed submitted workflow. For example, imagine a sales person has reviewed a product order form on their task list and signed it and submitted it. This is the final form workflow step and the form is submitted to a document management system (ECM). All the form data as well as the PDF order form image is stored in the ECM. While the order is processed someone notices that a product is out of stock and wants to return the order form back to the sales person's form workflow task list. The reset task API feature makes this possible. The demo.war contains a api/resetTask.jsp that demonstrates how to use this API feature:

1. Copy <frevvo-home>/ext/demo.war to <frevvo-home>/tomcat/webapps. It will auto-extract.
2. Edit frevvo/tomcat/webapps/demo/api/resetTask.jsp
    a. Set the host:port **to your frevvo server**
    b. Set the username and password and tenant to the designer user how is the owner of the workflow you want to reset
3. Enable the flow's save property so the flow data is stored in frevvo's submission repository. This is critial as frevvo needs to access this data in order to reset the workflow back to a task list.
4. Restart the frevvo server.
5. Use the flow and complete/submit it.
    a. Save the frevvo.form.id POST parameter. It will be a GUID such as _udowHBy0EeCJpdjaAfsaSA

To reset the completed/submitted flow back to a task list, send a POST from your system to the resetTask.jsp. You must pass the flowing Url parameters to resetTask.jsp:

- frevvo.form.id - The GUID sent in the worflow submission POST to your back end system (ECM in the example above).
- frevvo.reset.activity.name - The actually name of the activity you want to reset the task to. See workflow activity properties in the flow designer. Note that is is simpler to have no spaces in the activity name.

Here is an example Url if the name of the activity that the sales person performed was "SalesReview" and the frevvo.form.id sent in the post to your back end system was _2U71ABwbEeC6avHeCMEeBw

```
http://localhost:8082/demo/api/resetTask.jsp?frevvo.form.id=_2U71ABwbEeC6avHeCMEeBw&fr
evvo.reset.activity.name=SalesReview
```

Here is sample code for resetTask.jsp. You should modify this as needed.

```
<%@ page import="java.util.*,com.frevvo.forms.client.*,com.frevvo.forms.client.util.*"
%>
<html>
   <head>
   </head>
   <body>
<%
FormsService service = new FormsService("http", "localhost", 8082, null);
  service.login ("designer@nancy.com", "designer");

String formId = request.getParameter("frevvo.form.id");
TaskEntry entry = service.getEntry(service.getEntryURL(TaskEntry.class, formId),
TaskEntry.class);
String stepName = request.getParameter("frevvo.reset.activity.name");
entry.setResetToStep(stepName);
entry.update();

service.logout();
%>
Your task was successfully reset.
   </body>
</html>
<pre>
```