Embedding frevvo: A Getting Started Guide

This Tutorial will walk you through step-by-step instructions to embed frevvo within your own web application. We'll describe one specific path for integration that's most commonly used by our customers. Obviously, there could be differences depending on the outer (embedding) application and its design. This Tutorial will also use the Java API – embedding with the .NET API is very similar.

Prerequisites

Step 1. Create a tenant with Delegating Security Manager

You'll need an in-house installation of frevvo. If you prefer to use a Cloud account, you'll need to contact frevvo. After you have completed the in-house installation, you'll need to create a tenant. For this tutorial, we'll use the the Delegating Security Manager. This security manager is explicitly designed for external applications that want to embed frevvo. Users and roles are defined outside frevvo (in your application) and your application is responsible for authentication and for passing credentials to frevvo.

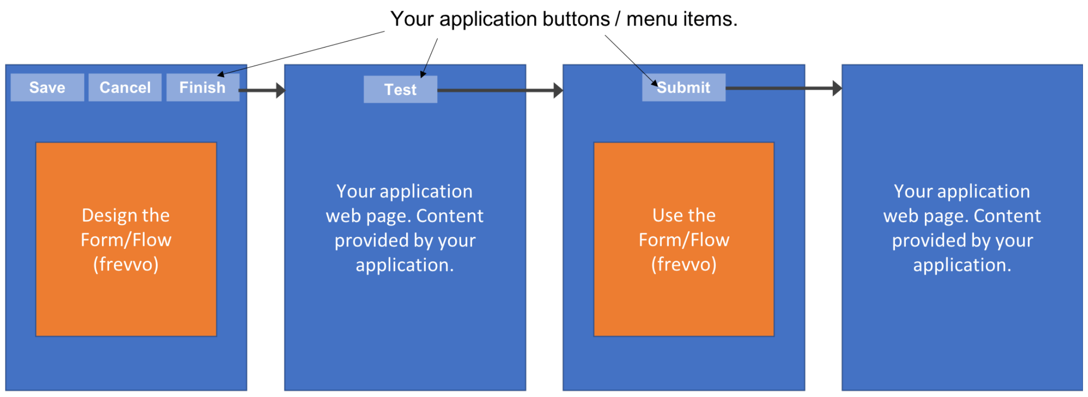
Create a tenant for this integration:

1. Browse this URL - **http://localhost:8082/frevvo/web/login -** Change the <localhost:8082> to the server:port where you installed Live Forms, if necessary
2. Login to Live Forms as user **admin@d**, password "admin".
3. Click the "***Manage Tenants***" link on the page that displays.
4. There is a single tenant named d. This is the default tenant.
5. Click the ⊕ icon to create a tenant and fill in the form. In the Security Manager Class pick list, choose Delegating Security Manager. Your new tenant will be created.

For the purposes of this tutorial, we'll assume you've created a tenant called dsm with Delegating Security Manager set, the admin user id is "admin" with password "admin". Of course, you should pick a more secure password.

We'll also assume that your application is accessed using a URL like http://localhost/oem/...

The integration described here looks like the image below.


Your application buttons / menu items.
Save
Cancel
Finish
Test
Submit
Design the Form/Flow (frevvo)
Your application web page. Content provided by your application.
Use the Form/Flow (frevvo)
Your application web page. Content provided by your application.

Content in blue comes from your web application.
Content in orange comes from frevvo.

Step 2: Some basic concepts

frevvo is organized as follows:

- Tenants have users: designers and non-designers.
    - Designers can create and edit forms and flows. Non-designers can use/administer them, view submissions etc. but cannot create or edit them.
- A designer user organizes forms and flows in Applications. You can have any number of Applications in a user. The Application is the basic downloadable unit in frevvo. While you can download individual forms and flows, we recommend sticking to Applications since they are self-contained.
- Each Application can hold an arbitrary number of schemas (XSDs), forms and flows. In frevvo, the form and flow designs are referred to as FormType and FlowType respectively. Individual instances are referred to as Form and Flow respectively. You'll notice that methods in the API use these terms e.g. FormTypeEntry refers to the FormType and FormEntry refers to an individual Form (an instance of the FormType).
- A FormType or FlowType can refer to multiple DocumentTypes. **Note: if you aren't planning to use XML Schemas, you can ignore this part.** As you might expect, an individual Form (an instance of the FormType) will have one or more Documents (instances of the corresponding DocumentTypes). You can read more in this article.
    - A form that's designed top-down – starting with a blank form and dragging and dropping controls into the canvas – will have a single DocumentType in it (sometimes referred to as the from-scratch DocumentType).
    - A form that's designed bottom-up – from one or more XML Schemas (XSD) – will have multiple DocumentTypes. There will always be the from-scratch DocumentType since you can always drag/drop controls. In addition, there will be a DocumentType per XSD root element that you add to the form.

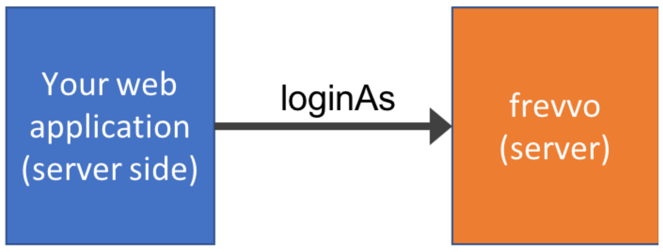
For OEM integrations, we strongly recommend that you have a 1-1 relationship between an Application and a form or flow (technically a FormType or FlowType). Specifically, you should always create a new Application and in that Application have a single form or flow. You can then download and manage the frevvo Application easily in the context of your own system.

Scenario I: Design a blank form and Test it

Step 1: Authenticate to frevvo from your application

In order to automatically integrate with Live Forms, your web application will first need to establish a session using the Live Forms API. This is done by using the **com.frevvo.forms.client.FormsService** class and by providing proper credentials to authenticate.

In most cases, you will want to use the loginAs() method available in V4.1.1 and higher. loginAs() allows you to login to frevvo as any tenant user provided you pass in the tenant admin's user and password credentials. This is convenient when you want login to frevvo using the same user that is logged into your application without having to know their password.



Establishes a login session between your web application (server) and frevvo (server).

The following snippet shows how to login as a designer user in the tenant with user id "designer".

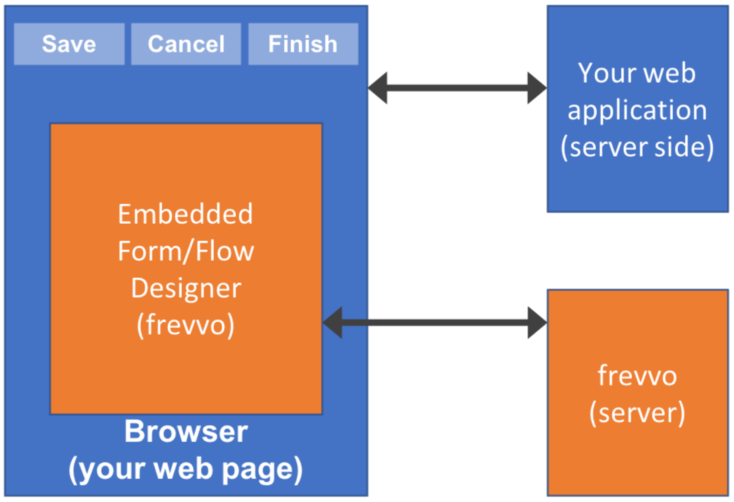
```
. . .
FormsService fs = new FormsService(protocol, host, port, null);
Map<String, String> customParams = new HashMap<String, String>(1);
customParams.put("autoLogin", "true");

// If you want to login as a designer user (who can create forms), pass in this role. Otherwise, pass in null.
List<String> roles = Collections.singletonList("frevvo.Designer");

// AutoLoginUserInfo alui = fs.loginAs(userId, tenantAdminUserId, tenantAdminPassword, virtual, roles, firstName, lastName, email,
customParams);
AutoLoginUserInfo alui = fs.loginAs("designer", "admin@dsm", "admin", false, roles, "", "Last Name", "Email Address", customParams);
if (alui == null)
throw new ServiceException("Login failed.<br/>");
. . .
```

- protocol is one of http or https
- host is the frevvo server host name
- port is the port on which frevvo is running
- userId is the id of the user you wish to login as.
- tenantAdminUserId is the user id of the tenant admin user specified using the syntax adminUserId@tenantName
- tenantAdminPassword is the password of the tenant admin user
- virtual: if set to true, does not actually create the user. For now, leave it set to false so the user is actually created.
- roles: You can provide a list of roles. For now, set the role frevvo.Designer. This creates the user and sets permissions so he/she can design forms. Of course, you can pass a list with as many roles as you want. You can also pass in null if you do not want to specify any roles.
- customParams: For now, set the autoLogin parameter with value true.

Since we chose the Delegating Security Manager, the user will be automatically created if the specified userId does not exist. Now, you are logged in from your application to frevvo as user designer.

Step 2: Create a new blank form for editing and render it

Now that we're logged in as a designer user, let's create a new form and display the frevvo designer. Our goal is to render the frevvo designer embedded inside your web application in the browser. The end user accesses your application page (e.g. http://localhost/oem/newform). The resulting page has your outer page content + the embedded frevvo form designer.


Save
Cancel
Finish
Embedded Form/Flow Designer (frevvo)
Browser (your web page)
Your web application (server side)
frevvo (server)

The following snippet shows how.

```
. . .
// FormService fs (created in above snippet)
if (fs == null)
throw new ServiceException("Not logged in.");

ApplicationEntry appEntry = new ApplicationEntry();
String appName = "Test App with blank form";
appEntry.setTitle(new PlainTextConstruct(appName));
// Insert it on Frevvo
ApplicationFeed feed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);
assert (feed != null);
appEntry = feed.insert(appEntry);

// Create form
FormTypeEntry ftEntry = new FormTypeEntry();
ftEntry.setTitle(new PlainTextConstruct("Blank Test Form"));
ftEntry.setVisibility(Visibility.PUBLICTENANT);
// Insert it into Frevvo application
ftEntry = appEntry.getFormTypeFeed().insert(ftEntry);
assert (ftEntry != null);

FormEntry fEntry = createEditingInstance(ftEntry, null, req, fs);
assert (fEntry != null);
String hRef = fEntry.getFormEditLink(params).getHref();
...
```

The code is self-explanatory. We first create a new Application in the designer user, then we create a new FormType and insert it in the Application, and finally we create an editing instance of the FormType. Now, we have a FormEntry that represents the editing form instance on the frevvo server and a link (URL) to that editing form instance. You should save the FormTypeEntry and FormEntry in your server-side code since you'll need them later. frevvo's API provides a serializable class for this purpose that you'll need to use called EntryHolder. Simply create a new EntryHolder (fEntry) as shown below:

```
EntryHolder typeHolder = new EntryHolder (ftEntry);
EntryHolder instanceHolder = new EntryHolder (fEntry);
// Stash the holders somewhere
```

Lastly, render the form designer embedded in your application's web page using an iframe.

```
<html>
<body style='background-color: #bbffff;'>
<Your application's web page HTML>

<iframe id="12345" src="hRef">


<Your application's web page HTML>
</body>
</html>
```

This will render your web page with the internal iframe coming from the frevvo server. The internal iframe will display the frevvo form designer embedded in your page.

Step 3: Save, Finish, and Cancel

It's also a very common requirement to hide the frevvo designer toolbar buttons and replace them with buttons or menu items consistent with your existing web page design to Save, Finish, and Cancel the form. Let's take a look at these – they are all very similar. First, add the buttons to your web page. Clicking each button simply links to a URL in your application.

```
<html>
<body style='background-color: #bbffff;'>
<Your application's web page HTML>

<SAVE button>
<FINISH button>
<CANCEL button>
<iframe id="12345" src="hRef">


<Your application's web page HTML>
</body>
</html>
```

Save button

Assume that clicking this button links to an endpoint in your web application e.g. http://localhost/oem/saveform.

```
// FormService fs (created above)
if (fs == null)
throw new ServiceException("Not logged in.");
// Saved above in instanceHolder
FormEntry fEntry = getFormEntry(...);
fEntry = fEntry.getSelf();
fEntry.submitEditingInstance(true, req.getHeader("Authorization"), null);

// Saved earlier in typeHolder
FormTypeEntry ftEntry = getFormTypeEntry(...);
assert (ftEntry != null);
fEntry = createEditingInstance(ftEntry, null, req, fs);
assert (fEntry != null);
EntryHolder instanceHolder = new EntryHolder (fEntry);
// Stash it away.
```

Re-render your web page as described above with the embedded iframe. Unfortunately, you will have to refresh the page – under the covers, frevvo has saved and committed the editing form, cleaned up, created a brand new instance with a different ID and URL. The old instance is no longer valid so we have to refresh the page using the URL for the newly created instance.

Finish button

Assume that clicking this button links to an endpoint in your web application e.g. http://localhost/oem/finishform.

```
// FormService fs (created above)
if (fs == null)
throw new ServiceException("Not logged in.");
// Saved above in instanceHolder
FormEntry fEntry = getFormEntry(...);
fEntry = fEntry.getSelf();
fEntry.submitEditingInstance(true, req.getHeader("Authorization"), null);
```

It's identical to the Save button except you don't create and re-render a new instance. The browser will now display a page rendered by your web application.

Cancel button

Assume that clicking this button links to an endpoint in your web application e.g. http://localhost/oem/cancelform.

```
// FormService fs (created above)
if (fs == null)
throw new ServiceException("Not logged in.");
// Saved above in instanceHolder
FormEntry fEntry = getFormEntry(...);
fEntry = fEntry.getSelf();
fEntry.submitEditingInstance(false, req.getHeader("Authorization"), null);
```

It's identical to the Finish button except for the false instead of true parameter on the last line.

Step 4: Test the form you just designed.

Now, assume you just clicked the Finish button and you've returned a web page with a Test button.

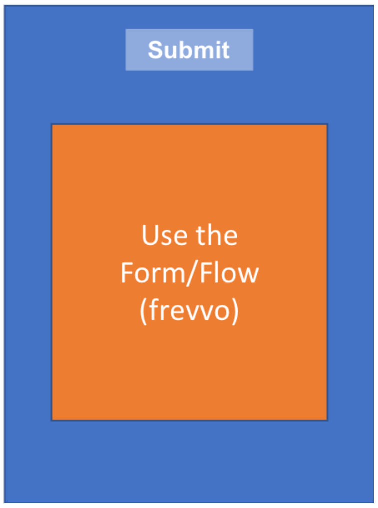
```
<html>
<body style='background-color: #bbffff;'>
<Your application's web page HTML>

<TEST button>

<Your application's web page HTML>
</body>
</html>
```

Test button

Clicking it will take you to an endpoint in your web application e.g. http://localhost/oem/testform.

```
if (fs == null)
throw new ServiceException("Not logged in.");
String formtypeCompositeId = null;
ApplicationFeed feed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);
assert (feed != null);

// Get the FormTypeEntry so you can create an instance of it. Earlier, we used the entry stashed away in a holder.
// Here's another way to construct a FormTypeEntry for an existing form (the one we just created above).

// Find the application we created by name (you can also find it by ID). Find the form we just created in this application.
String appName = "Test App with blank form";
for (ApplicationEntry appEntry : feed.getEntries()) {
if (appEntry.getTitle().getPlainText().contains(appName)) {
FormTypeFeed ftFeed = appEntry.getFormTypeFeed();
for (FormTypeEntry ftEntry : ftFeed.getEntries()) {
// There will be just one form.
formtypeCompositeId = ftEntry.getId();
break;
}
}
}

// Get the formType entry.
URL formTypeURL = fs.getEntryURL(FormTypeEntry.class, formtypeCompositeId);
assert (formTypeURL != null);
final FormTypeEntry ftEntry = (FormTypeEntry) fs.getEntry(formTypeURL, FormTypeEntry.class);
assert (ftEntry != null);

// Params (optional). There are many parameters - let's set a few.
Map<String, Object> params = new HashMap<String, Object>(1);
// If set, any form action will be ignored and the document set (including attachments) will be returned when the form is submitted.
// Otherwise, normal form action processing is performed (default). We set it so we can retrieve the data later (see below).
params.put(FormTypeEntry.FORMTYPE_FORMACTIONDOCS_PARAMETER, "xml");
// Control form resizing. You can set this to
// FormTypeEntry.VALUE_RESIZE_TRUE  to turn auto-resizing on.
// FormTypeEntry.VALUE_RESIZE_HORIZONTAL to turn auto-resizing on horizontally only.
// FormTypeEntry.VALUE_RESIZE_VERTICAL  to turn auto-resizing on vertically only.
// Set to FormTypeEntry.VALUE_RESIZE_FALSE
params.put(FormTypeEntry.EMBED_RESIZE_PARAMETER, FormTypeEntry.VALUE_RESIZE_VERTICAL);
// Set the initial width (defaults to the width set in the form or 600px otherwise) for the iframe.
params.put(FormTypeEntry.EMBED_WIDTH_PARAMETER, "99%");
// Set FormTypeEntry.EMBED_SCROLLING_PARAMETER
// to "auto" (default) if you want to automatically show scrollbars if required.
// to "no" if you don't want scrollbars.
// to "yes" if you always want scrollbars.
params.put(FormTypeEntry.EMBED_SCROLLING_PARAMETER, "no");

// Create a new instance of the form, construct its entry and stash it away.
URL fUrl = ftEntry.createFormInstance(params, null, null, null);
assert (fUrl != null);
String formId = FormsService.SCHEME_KIND_FLOW.equals(ftEntry.getKind()) ? Helper.extractFlowId(fUrl) : Helper.extractFormId(fUrl);
assert (formId != null);

FormEntry fEntry;
FormFeed fFeed = ftEntry.getFormFeed();
assert (fFeed != null);
for (FormEntry fE : fFeed.getEntries()) {
if (fE.getId().startsWith(formId))
fEntry = fE;
break;
}
assert (fEntry != null);
EntryHolder instanceHolder = new EntryHolder (fEntry);
// Stash it away. In this example, we'll stick it in the HTTP session.
session.setAttribute("frevvo.forms.form.use.entry", instanceHolder);

// Display your page with embedded form. You have to pass the instanceHolder saved above to the JSP. For example, above we saved it
in the HTTP session.
resp.sendRedirect("useform.jsp");
```

We find the formType, retrieve its FormTypeEntry, create a new instance of the form, retrieve its FormEntry and save it. Lastly, display the form embedded in your application's web page using an iframe. Let's look at an example using JSP.



```
<%@ page import="java.util.*,com.frevvo.forms.client.*,com.frevvo.forms.client.util.*,com.frevvo.ping.api.*" %>
<html>
<head>
</head>
<body>
<%
EntryHolder holder = (EntryHolder) session.getAttribute("frevvo.forms.form.use.entry");
if (holder != null){
FormEntry fEntry = holder.getEntry(fs, FormEntry.class);
if (fEntry != null) {
formUrl = fEntry.getFormUseEmbedLink().getHref();
}
}
%>

<div id="wrapper">
<a href=".../oem/submitForm">Submit</a>
<script src="<%= formUrl %>" type="text/javascript"/>
</div>
</body>
</html>
```

Retrieve the holder from the session, extract the FormEntry and get the embed link for using the form. Use it in a <script> tag as shown above. That's it. frevvo does the rest – the form instance will appear embedded in the browser.

Step 5: Submit the form and retrieve submission data

Now, click the Submit button. It goes to an endpoint in your web application e.g. http://localhost/oem/submitform. You can programmatically submit the form from your web application and retrieve the data and documents that were sent if you set the FormTypeEntry.FORMTYPE_FORMACTIONDOCS_PARAMETER to a non-null value as shown in the sample code above. This is the recommended approach.

```
if (fs == null)
 throw new ServiceException("Not logged in.");


// Saved above in instanceHolder
FormEntry fEntry = getFormEntry(...);
fEntry = fEntry.getSelf();

FormData data = fEntry.submitInstance(req.getHeader("Authorization"), null);
if (data != null) {
 for(Part part : data) {
 // Got part with content-type: " + part.getContentType();
 // Content is: part.getInputStream(), "utf-8");
 }
}
```

Step 6: Download the Application

In most cases, OEM partners who embed frevvo's software prefer to save the form definitions (metadata) in their own systems. That enables them to fully control storage, access, permissions etc. frevvo's downloadable unit is the Application. If you remember, before we created a form for editing we first created an Application and added it to the designer user and then placed the form in this Application. You can download the Application and save it yourself.

```
if (fs == null)
 throw new ServiceException("Not logged in.");

ApplicationFeed feed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);
assert (feed != null);

// Find the application we created by name (you can also find it by ID). Find the form we just created in this application.
String appName = "Test App with blank form";
ApplicationEntry appEntry = null;
for (ApplicationEntry entry : feed.getEntries()) {
 if (entry.getTitle().getPlainText().contains(appName)) {
  appEntry = entry;
  break;
 }
}

 // download

MediaContent mc = (MediaContent) appEntry.getContent();
assert (mc != null);
MediaSource ms = fs.getMedia(mc);
assert (ms != null);
assert ("application/zip".equals(ms.getContentType()));
assert (ms.getInputStream() != null);
byte[] buf = new byte[8196];
int len;
while ((len = ms.getInputStream().read(buf)) > 0)
// Process the bytes and save the application zip file.
```

Step 7: Delete the Application

Finally, you can delete the application from frevvo so that it is only saved in your own system. Of course, once deleted you cannot instantiate forms from this application. Many OEM partners will use one physical instance of the frevvo server for design and a separate physical instance for filling forms. If so, you can design forms, test them, download and delete from your design-time system and deploy them to your run-time system using your own deployment rules.

```
if (fs == null)
 throw new ServiceException("Not logged in.");

ApplicationFeed feed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);
assert (feed != null);

// Find the application we created by name (you can also find it by ID). Find the form we just created in this application.
String appName = "Test App with blank form";
ApplicationEntry appEntry = null;
for (ApplicationEntry entry : feed.getEntries()) {
 if (entry.getTitle().getPlainText().contains(appName)) {
  appEntry.delete();
  try {
   appEntry = appEntry.getSelf();
   assert (false);
  } catch (ResourceNotFoundException e) {
   assert (true); // Exception is expected.
  }
 }
}
```

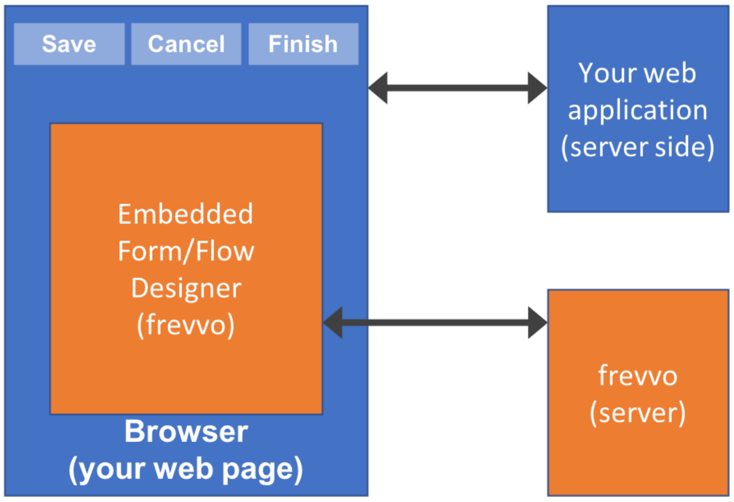
Scenario II: Design a form using an XML schema

Step 1: Authenticate to frevvo from your application

This is identical to Scenario I above.

Step 2: Create a form for editing from a schema

Now that we're logged in as a designer user, let's create a new form and display the frevvo designer. This time, we have an XML schema that we want to use as the metadata for the form's controls. As before, our goal is to render the frevvo designer embedded inside your web application in the browser. The end user accesses your application page (e.g. http://localhost/oem/newformschema). The resulting page has your outer page content + the embedded frevvo form designer.


Save
Cancel
Finish
Embedded Form/Flow Designer (frevvo)
Browser (your web page)
Your web application (server side)
frevvo (server)

The following snippet shows how.

```
. . .
// FormService fs (created in above snippet)
if (fs == null)
throw new ServiceException("Not logged in.");

ApplicationEntry appEntry = new ApplicationEntry();
String appName = "Test App with XSD form";
appEntry.setTitle(new PlainTextConstruct(appName));
// Insert it on Frevvo
ApplicationFeed feed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);
assert (feed != null);
appEntry = feed.insert(appEntry);

// Create form
FormTypeEntry ftEntry = new FormTypeEntry();
ftEntry.setTitle(new PlainTextConstruct("Test Form XSD"));
ftEntry.setVisibility(Visibility.PUBLICTENANT);
// Insert it into Frevvo application
ftEntry = appEntry.getFormTypeFeed().insert(ftEntry);
assert (ftEntry != null);

// add document type
InputStream is = ... // Input stream for XML Schema (must be a zip file).
try {
 // upload schema zip
 SchemaFeed schemaFeed = appEntry.getSchemaFeed();
 SchemaEntry schemaEntry = schemaFeed.insert(Helper.toSchemaMediaSource(is, "application/zip", "root.xsd", "Test Schema"));
 schemaEntry.setSystem(true);
 schemaEntry = schemaEntry.update();

 // add DocumentTypes to form
 DocumentTypeFeed dtFeed = ftEntry.getDocumentTypeFeed();
 DocumentTypeFeed dtFeedSchema = schemaEntry.getDocumentTypeFeed();
 for (DocumentTypeEntry dtEntry : dtFeedSchema.getEntries()) {
  dtFeed.insert(dtEntry);
 }
} finally {
 is.close();
}

FormEntry fEntry = createEditingInstance(ftEntry, null, req, fs);
assert (fEntry != null);
String hRef = fEntry.getFormEditLink(params).getHref();
...
```

The code is similar to the previous example with the difference being the green snippet above where we first add the XSD to the application and then add the root elements from the XSD to the document types of the form. The XSD must be provided as a zip file even if it is entirely contained in a single file. However, the system supports zips with multiple XSD files that import each other.

To insert the XSD into the Application entry, we use the method:

```
Helper.toSchemaMediaSource(is, "application/zip", "root.xsd", "Test Schema");
```

The parameters are the InputStream, the content type (always application/zip), the name of the root XSD in the zip file (if there's a single file, simply provide the name of that file) and finally the name you want to assign to this XSD (can be anything).

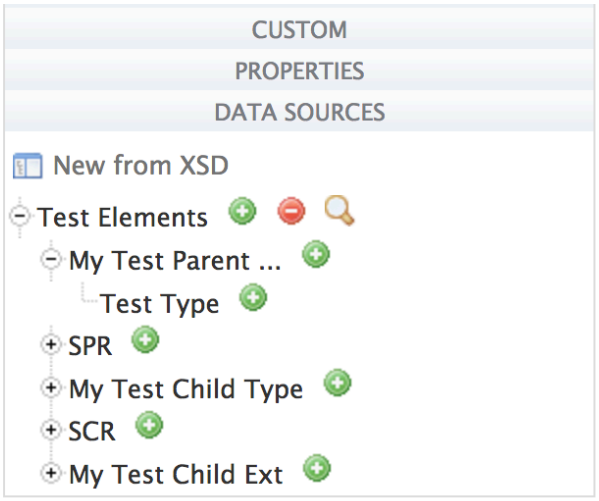
Finally, as before, we create an editing instance of the FormType. Now, we have a FormEntry that represents the editing form instance on the frevvo server and a link (URL) to that editing form instance. You should save the FormTypeEntry and FormEntry in your server-side code since you'll need them later. frevvo's API provides a serializable class for this purpose that you'll need to use called EntryHolder. Simply create a new EntryHolder (fEntry) as shown below:

```
EntryHolder typeHolder = new EntryHolder (ftEntry);
EntryHolder instanceHolder = new EntryHolder (fEntry);
// Stash the holders somewhere
```

Lastly, render the form designer embedded in your application's web page using an iframe.

```
<html>
<body style='background-color: #bbffff;'>
<Your application's web page HTML>

<iframe id="12345" src="hRef">


<Your application's web page HTML>
</body>
</html>
```

This will render your web page with the internal iframe coming from the frevvo server. The internal iframe will display the frevvo form designer embedded. The Data Sources panel of the form designer will display the available document types. The designer can expand and add the desired segment(s) from the XSD.



Steps 3–7: Save/Finish, Test, Submit the form. Download and Delete the Application.

These steps are identical to to Scenario I above.

Scenario III: List existing forms, use and submit them and view submissions

Step 1: Authenticate to frevvo from your application

This is identical to Scenario I above.

Step 2: List existing applications / forms

Step 3: Upload an application

As discussed above, most OEM partners prefer to save form definitions (metadata) in their own systems. When it's time to use a particular form, you will first try to find it (as described above). If it's not found (usually because it's being used for the first time), you'll need to upload your copy of the application that you downloaded above. Here's how:

```
if (fs == null)
 throw new ServiceException("Not logged in.");

ApplicationFeed appFeed = fs.getFeed(fs.getFeedURL(ApplicationFeed.class), ApplicationFeed.class);

// upload
MediaStreamSource mediaSource = new MediaStreamSource(is, "application/zip");
ApplicationEntry appEntry = appFeed.insert(mediaSource);
assert (appEntry != null);
```

You can browse this ApplicationEntry to find your specific form(s) as required.

Step 4: Use a form and retrieve submission data.

This is already described in Steps 4 and 5 in Scenario I.

Step 5: Use a form and initialize it with data before rendering it.

You can create a form and initialize it with an XML document. The code is already described in Step 4 from Scenario I. The only difference is in the createInstance() method of the FormTypeEntry.

```
if (fs == null)
 throw new ServiceException("Not logged in.");

// Get the FormTypeEntry as described earlier.

// Params
Map<String, Object> params = new HashMap<String, Object>(1);
// Set params as described earlier.

// Get the init document.
List<MediaSource> mss = new ArrayList<MediaSource>();
InputStream fis = ...; // The XML document.
MediaStreamSource ms = new MediaStreamSource(fis, "application/xml");
ms.setName("Document");
mss.add(ms);
// You can add as many initial documents as you want to the ArrayList.

// Create a new instance of the form, construct its entry and stash it away. The only difference is that we pass in the init document(s).
URL fUrl = ftEntry.createFormInstance(params, mss, null, null);

// Process and render as described earlier.
```

Step 6: Use a form without logging in.

Sometimes, you don't want to render the form embedded in a browser window. For example, perhaps you want to obtain a URL to the formType and email it so it can be used – potentially by a different person – in a different browser. It's still not a public form and cannot be instantiated by anyone.

```
FormTypeEntry ft = ...
String url = ft.getFormTypePopupLink(null).getHref();
// This URL can be opened in a different browser.
```

However, note that this URL is only valid and will only work as long as the original FormsService object is valid i.e. as long as the login session established by the FormsService has not been destroyed. You can also programmatically create the form instance as described above, get the URL to the instance and open it in a different browser. This is useful if you want to, for example, initialize the instance with data.

If you want to allow completely anonymous access, you must set the visibility of the FormType to PUBLIC. Public forms can be instantiated by everyone without authentication e.g. a Contact form.

```
FormTypeEntry ftEntry = ...
ftEntry.setVisibility(Visibility.PUBLIC);
```