Relativity®

Fest 2019 Workshop

Relativity Forms API Training

October 21, 2019 - Version 1.0

Contents

1 Course Overview

Relativity has deployed a new web engine to display forms for dynamic objects that can be toggled on or off on a per object type basis. This new Relativity Forms engine has a first-class JavaScript API that enables developers to customize their pages faster and easier than ever before. We're encouraging everyone to migrate to this new technology. This workshop will hit on the advantages and provide a working example of how to create or convert customizations with the new Relativity Forms API.

1.1 What You'll Learn

- Relativity Forms life cycle pipelines
- Writing Page Interaction and Console Event Handlers for use in Relativity Forms
- (And in the appendix, a brief introduction to Relativity Forms API testing)

1.2 Required Development Software

This workshop and associated development resources require a development environment that includes the following items:

- Relativity 10.3.107.60 DevVM
- Visual Studio 2017
- Relativity Visual Studio Templates
- .NET Framework 4.6.2
- Node.js 10.x

We have provided the required software for you for this workshop.

2 Getting Started

2.1 Workshop Projects

1. Open a Windows PowerShell by clicking the right-most icon (a blue icon with a white prompt on it) in the task bar.

2. Once PowerShell is open, navigate to **E:\relativity-forms-workshop-fest-2019** and pull the latest workshop content by entering the following commands:
   - e:
   - cd relativity-forms-workshop-fest-2019
   - git pull
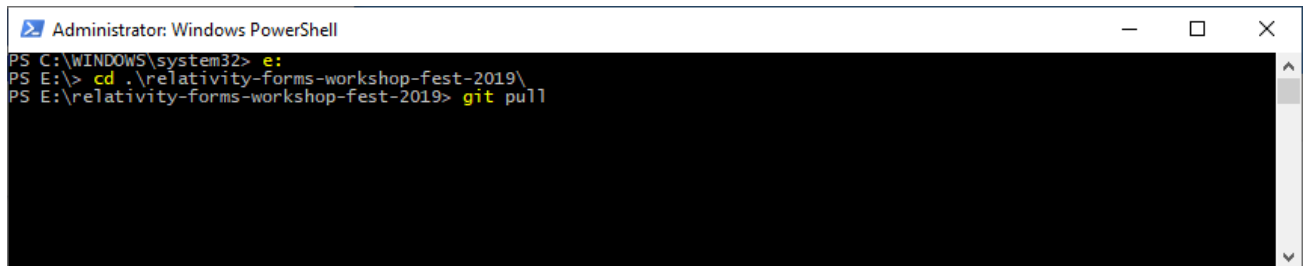
3. Make sure you have a folder shortcut on your Desktop with the name "**relativity-forms-workshop Shortcut**"
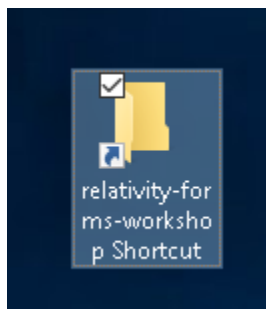
4. Double click on the folder to open it. Make sure you see the files as shown in the below screenshot.

E:\relativity-forms-workshop-fest-2019

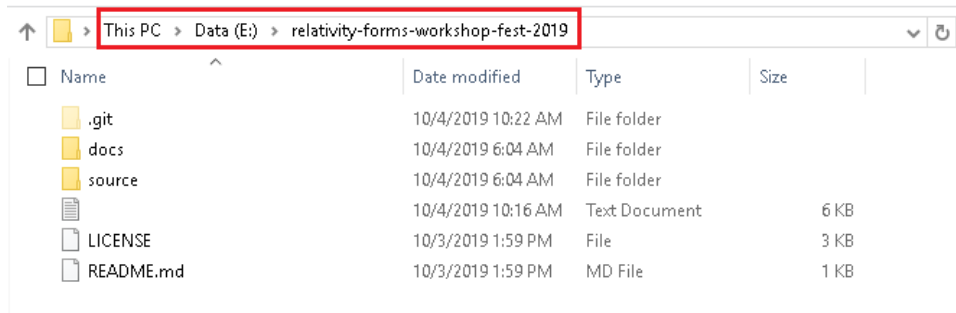5. Go to the **docs/workshop** folder. This folder contains the updated workbook PDF for this workshop. Double click this file to open it: **Fest 2019 Workshop – Relativity Forms API.pdf**
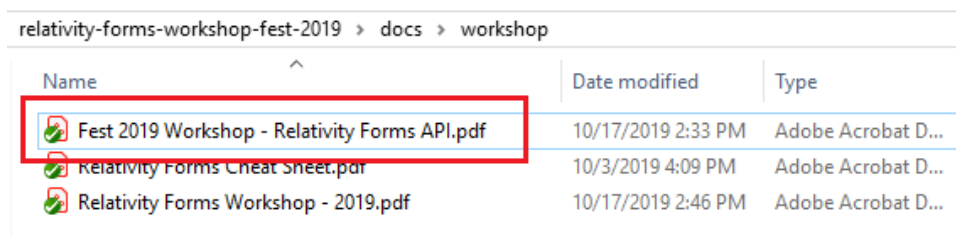
E:\relativity-forms-workshop-fest-2019\docs\workshop
E:\relativity-forms-workshop-fest-2019\docs\workshop\Fest 2019 Workshop – Relativity Forms API.pdf

6. In Explorer, return to **E:\relativity-forms-workshop-fest-2019** and then go to the **source/RelativityFormsWorkshop** folder. This folder contains the Visual Studio solution for the .NET portion of this workshop. Double click this file to open it in Visual Studio 2017.
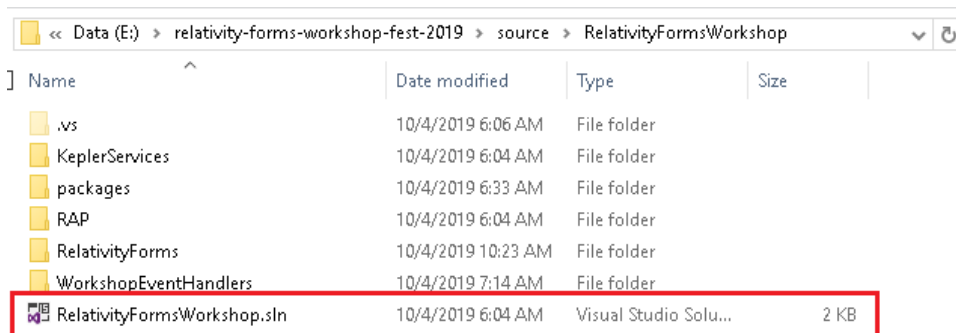
E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop
E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityFormsWorkshop.sln

7. In the same open Explorer window, open the **RelativityForms** folder in Visual Studio Code by right-clicking the folder and selecting the "Open with Code" option from the context menu.
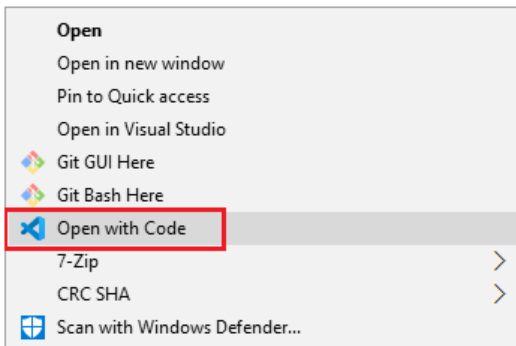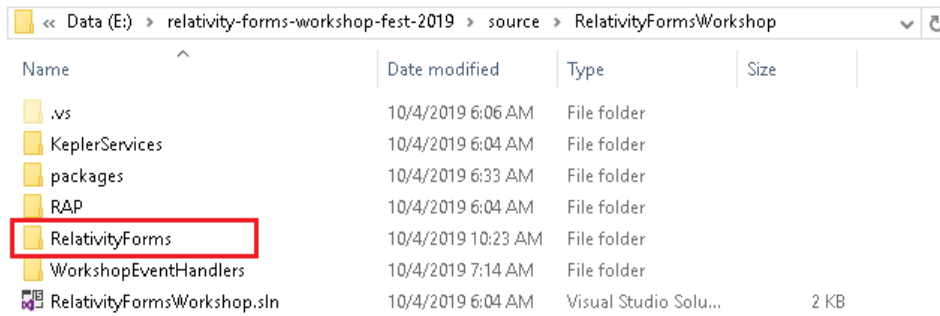
8. Information: The two sub-folders of **RelativityFormsWorkshop** which contain code that we will be modifying later in the workshop are:

- **WorkshopEventHandlers** - contains the code for the .NET event handlers, which we will modify and rebuild in later parts of this workshop.
  E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\WorkshopEventHandlers

- **RelativityForms** – contains the JavaScript code which we will use in later exercises in this workshop. It also contains a preview development template for writing and testing Relativity Forms event handler JavaScript. Due to time constraints, we will not be using this in the Workshop exercises today; however, a suppelement to Exercise 2, which does use the development template for basic testing and code quality checks is in this document's appendix.
  E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms

2.2 DevVm SetUp

1. Open the **Hyper-V Manager** application by clicking the first blue icon in the taskbar.
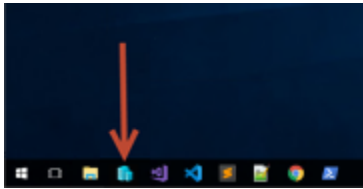
2. You will see a **Relativity 10.3.107.60 DevVM** listed in the Hyper-V Manager **Virtual Machines** section.
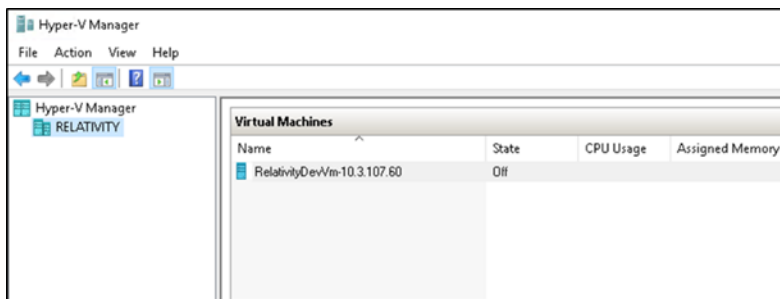
3. If the DevVm is off, right click on the **DevVM** and select the **Start** option.
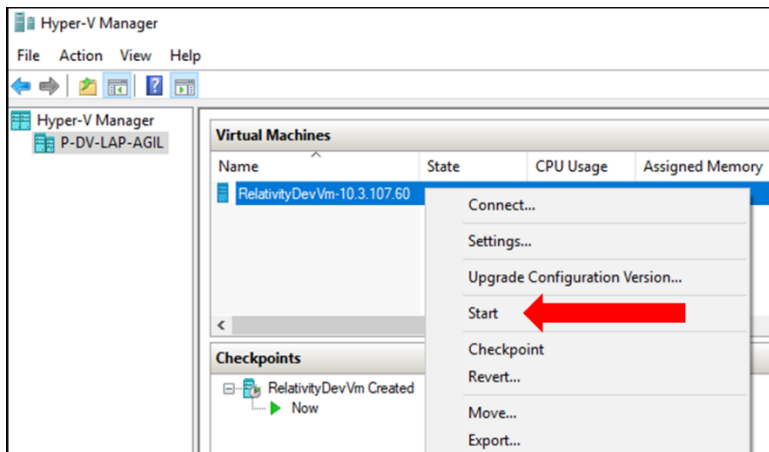
4. Right click on the **DevVM** and select the **Connect** option.
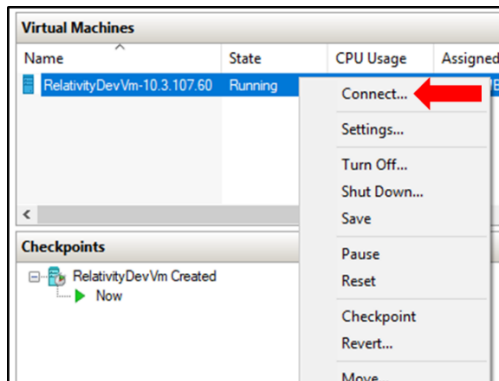
5. If you get a Display configuration pop-up, click the **Connect** button.
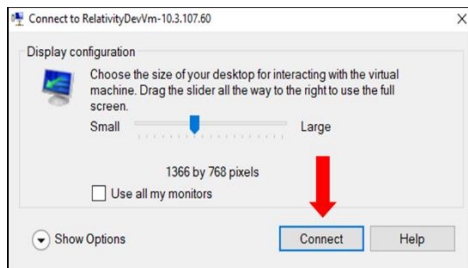
6. If faced with a sign in message, select the **Action** Menu item and then select **Ctrl+Alt+Delete** option. Otherwise, you are already logged-in and may skip to step 8.
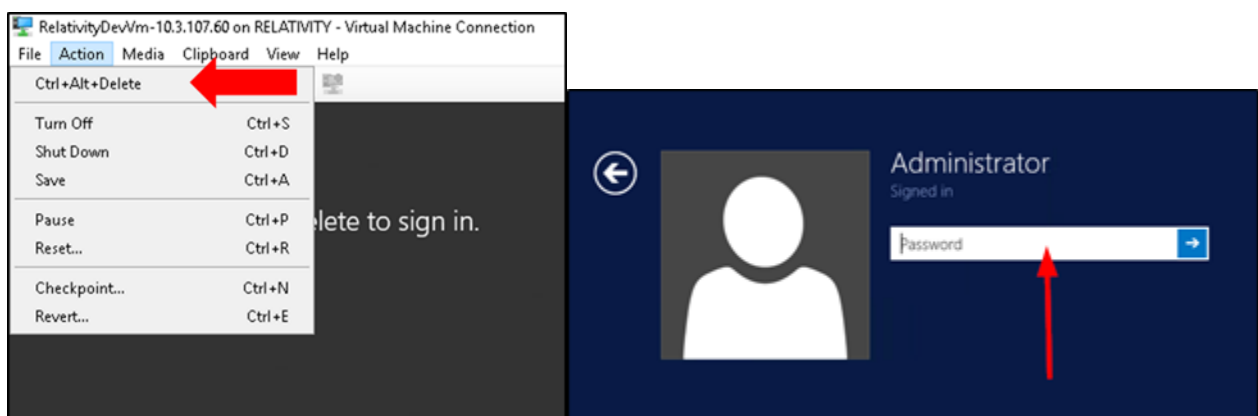
7. Use the below DevVM credentials highlighted in yellow to login to the VM.
   DevVM:
   Windows Admin login: Administrator
   Windows Admin password: Test1234!

8. On the DevVm, open Chrome. Navigate to relativity.com. If you are able to reach this address then your DevVm has internet set up and skip to step 9. If not follow the following instructions:
   a. **On your DevVm** *(not the Workshop Machine)*, right click on the Network icon in the bottom right corner and click *Open Network and Sharing Center*
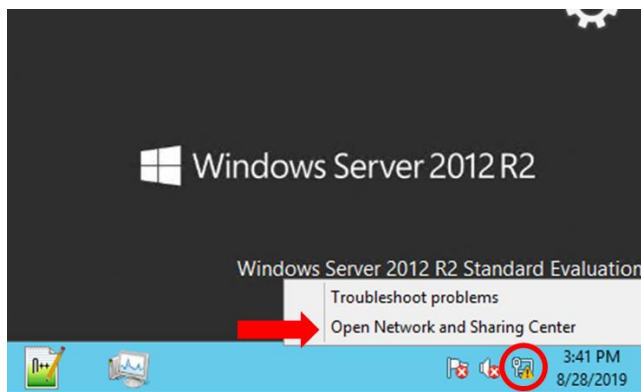
   b. Click on *Change adapter settings*
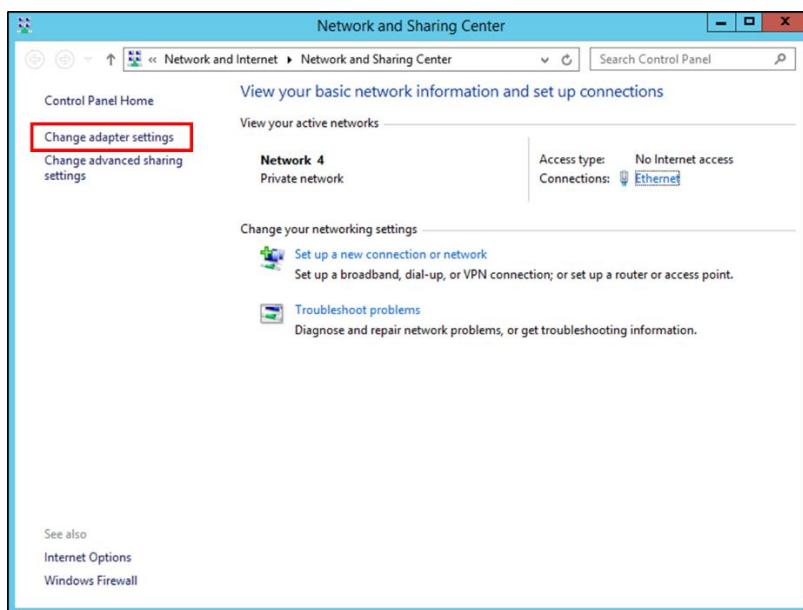
c. Select and right click on Ethernet and choose *Disable*
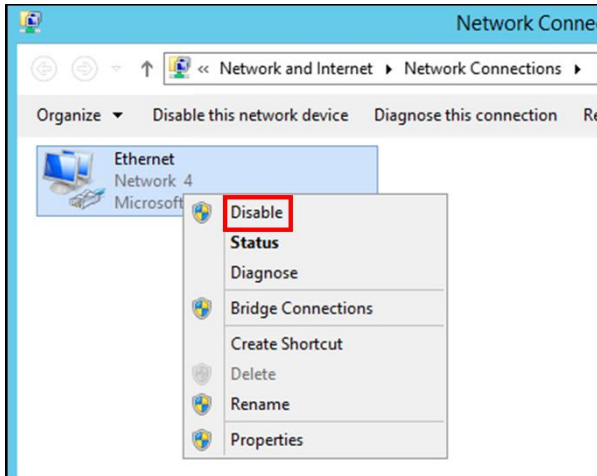
d. Next on your Workshop Machine *(not the DevVm)* right click on the Network icon in the bottom right corner and click *Open Network and Sharing Center*
e. Click *Change adapter settings*
f. Right Click on **Ethernet ##** (likely Ethernet 38) and select *Properties*
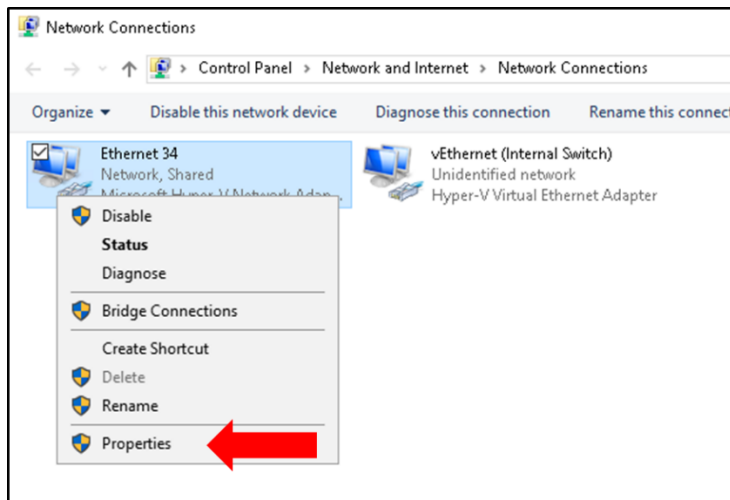
g. If it is checked, under the Sharing tab, **deselect** *Allow other network users to connect through this computer's Internet connection* and click Ok. Otherwise proceed to step h.
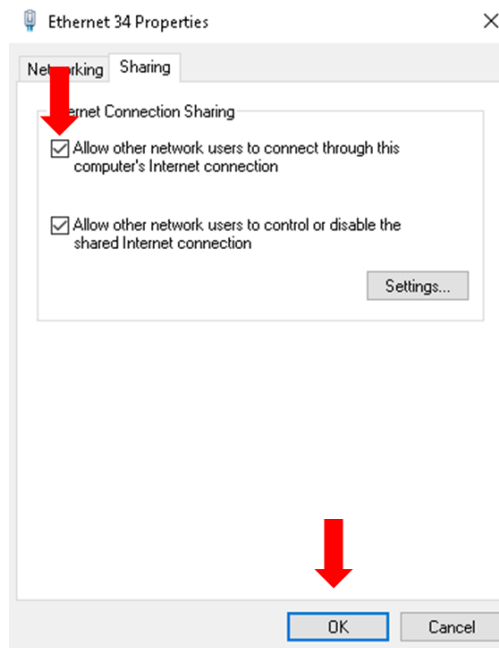
h. Now go through the same process to **reselect** *Allow other network users to connect through this computer's Internet connection* and click Ok. If a message pops up click Yes.

i. Go back to the DevVm *(Not the Workshop Machine)*, re enable Ethernet
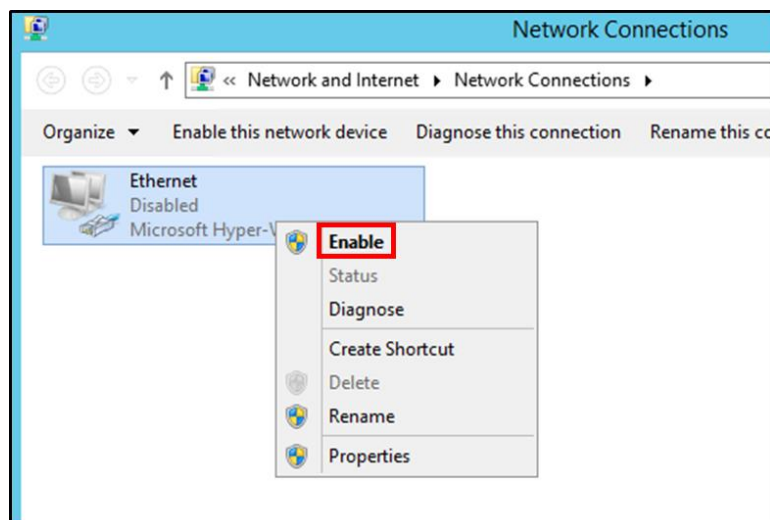
j. Now go to Chrome and navigate to relativity.com to test that your DevVm has internet.

9. On the DevVM, you should see the Desktop with few applications pinned to the taskbar, which you will be using throughout the workshop.

10. Click on the **Services** program icon ![gear icon] in the taskbar as shown in the below screenshot.
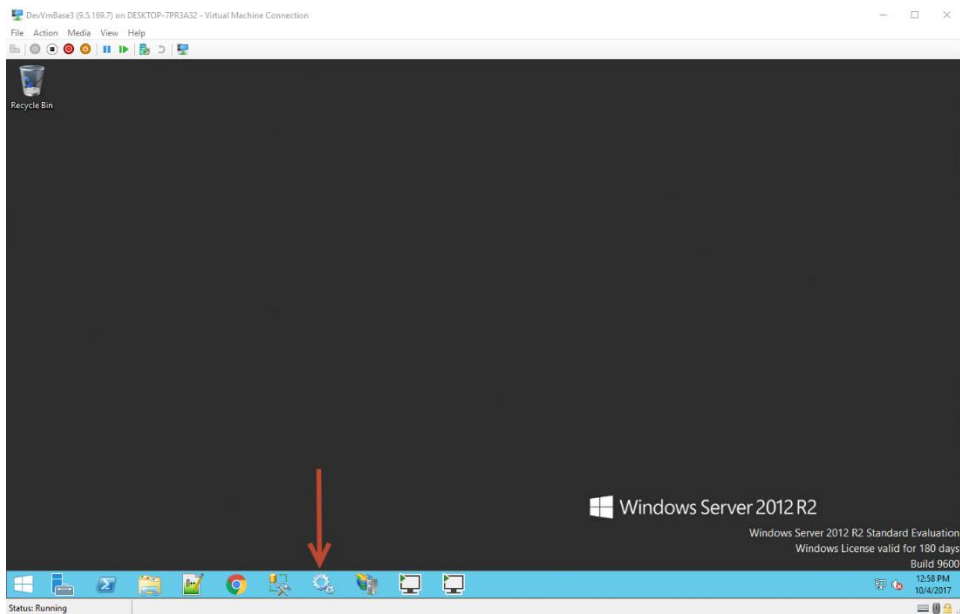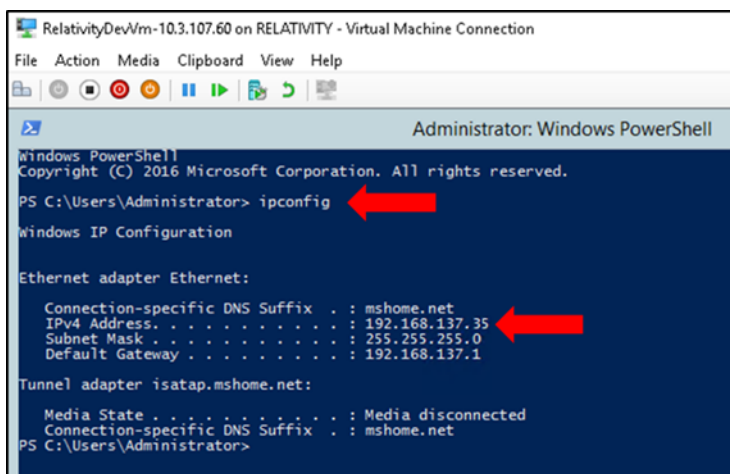
11. Make sure all the **services** listed below are **running**. If they are not already running, right click on the service and select the **Start** option,

    - kCura Service Host Manager
    - kCura EDDS Web Processing Manager
    - kCura EDDS Agent Manager
    - Service Bus Gateway
    - Service Bus Message Broker
    - Service Bus Resource Provider
    - Service Bus VSS

12. Open PowerShell on your DevVM and get the IP address of the VM.

Or open Hyper-V on your Workshop machine *(Not the DevVM)* and check the Networking tab on the bottom for the IP Address:
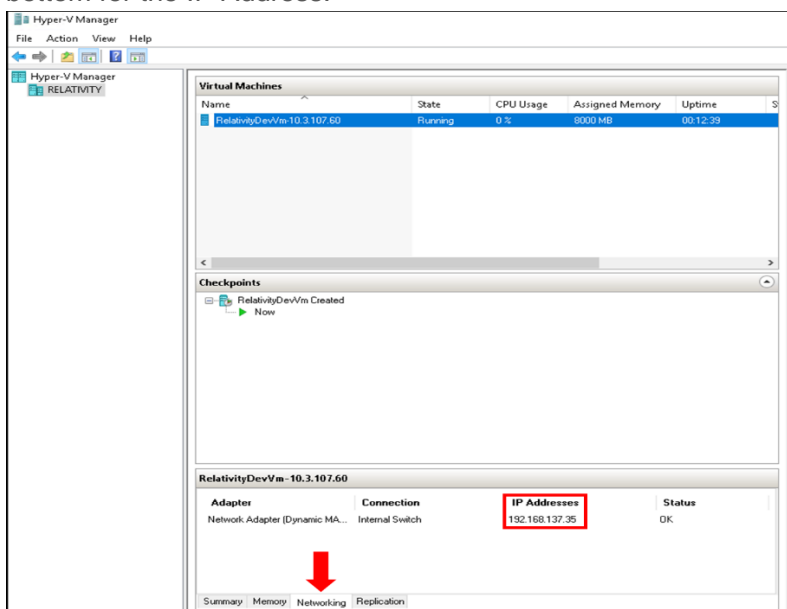
13. Open **Chrome** on your Workshop machine *(Not the DevVM)* and go to **http://{DevVmIPAddress}/Relativity** e.g (**http:// 192.168.137.35/Relativity**). Use the Relativity admin credentials below to login. Verify that you can login to Relativity.

   a. If the workspaces tab doesn't load, give it a few mins and then try refreshing the tab.

Relativity:

Admin login: relativity.admin@relativity.com

Admin password: Test1234!
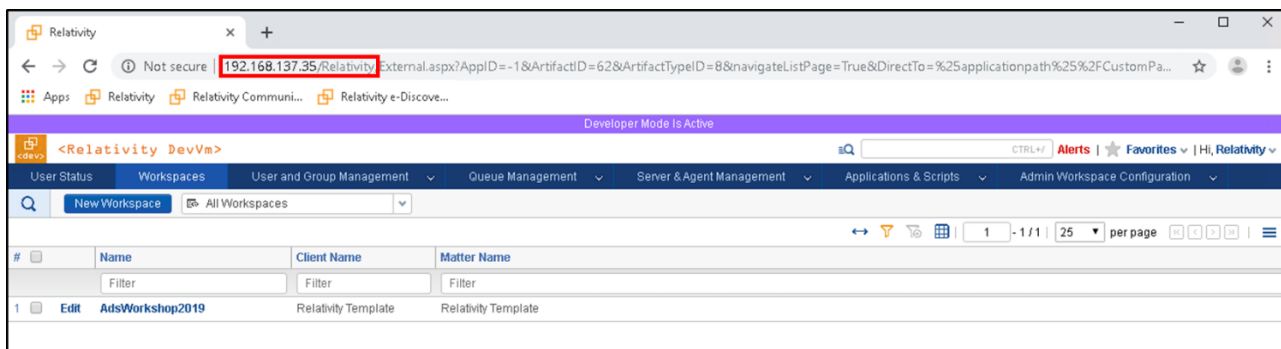
3 The Relativity Application

3.1 Purpose and Contents

The Relativity Application we will be starting with is called **Relativity Forms Workshop Fest 2019**. Its purpose is to track people's enthusiasm and whether they can contain their enthusiasm. The application is comprised of the following:

- o Object Type: Person
  - ▪ 2 Fields:
    - Current Enthusiasm
    - Contain Enthusiasm
    - Maximum Enthusiasm
  - ▪ 1 Tab, 1 View, and 1 Layout
  - ▪ 2 Event Handlers:
    - PageInteractionEventHandler
    - ConsoleEventHandler
- o A Kepler (ADS application-deployed) Service: PeopleAnalyzer

The PageInteractionEventHandler controls the visibility of the Maximum Enthusiasm Field, hiding or showing it based upon the value of the Contain Enthusiasm Field at the time the page loads, and when the value of Contain Enthusiasm is changed in Edit mode.

The ConsoleEventHandler contains a button which was intended to analyze the enthusiasm of the current Person; however, the team was mid-implementation when they decided that it was time to migrate this application to Relativity Forms.

3.2 Exercise 1: Enabling Relativity Forms

For this exercise, we will create two or more instances of the Person Object Type, use its event handlers, enable Relativity Forms on the Person Object Type, and experience the effect of doing so. *Please note that the type's* **Maximum Enthusiasm** *Field is required, in order to save the form, the Field will need to be given a value even if the Field becomes hidden.*

Using the Classic Application

1. Log in to Relativity
2. From the Workspace list, go to the Workspace "**Relativity Forms Workshop 2019**"
3. Click the "**Relativity Forms Workshop Fest 2019**" Tab to see a list of **Person** objects on the **People** Tab.

4. Save this list as a Favorite by clicking the gray star beside "Favorites" in the upper-right corner of Relativity.

5. Click "**New Person**" button
6. Notice how, on the Edit page, the **Maximum Enthusiasm** Field is not visible, as the true value ("Ok, I will") for **Contain Enthusiasm** is not selected.
7. Switch the value of **Contain Enthusiasm** between the true and false values to see the **Maximum Enthusiasm** Field be shown or hidden.
8. Complete the form, supplying a value for all four Fields in the form:
    a. **Name:** Calm Carla
    b. **Contain Enthusiasm:** "Ok, I will." (true)
    c. **Current Enthusiasm:** 100
    d. **Maximum Enthusiasm:** 400
9. Click "**Save and New**"
10. Complete this second form, supplying a value for all four Fields in the form:
    a. **Name:** Excited Ed
    b. **Contain Enthusiasm:** "No, I cannot." (false)
    c. **Current Enthusiasm:** 2000
    d. **Maximum Enthusiasm:** 2500
        Note that because this Field is required, it will need to be given a value before hiding it, in order to save the form
11. Click "**Save and New**"
12. Complete this third form, supplying a value for all four Fields in the form:
    a. **Name:** Agitated Alex
    b. **Contain Enthusiasm:** "Ok, I will." (true)
    c. **Current Enthusiasm:** 9001
    d. **Maximum Enthusiasm:** 9000
13. Click "**Save**" to return to View this new Person in View mode.

Unlocking the Application, and enabling Relativity Forms on the Person Object Type

1. Navigate to the **Relativity Applications** list by clicking on the "**Application Admin**" tab.
2. Filter the list by entering "**Forms**" in the Name Field filter, then click the link in the name column for the "**Relativity Forms Workshop Fest 2019**" application.

3. In the View form, click **"Unlock Application"** and accept the resulting confirmation dialog by clicking **"OK"**.

4. Navigate to the **Object Type** list in this Workspace by entering **"Object Type"** in the quick search input, and selecting the **"Object Type"** option beneath "WORKSPACE TABS"

5. Filter the Object Type list to only Person by entering **"Person"** into the Name Field filter, then click the Person row's **"Edit"** link.
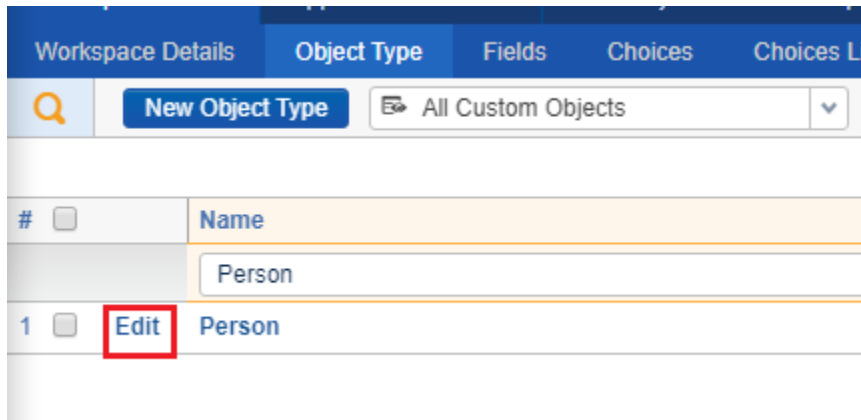
6. In the edit form, select **"Yes"** value for the **"Use Relativity Forms"** Field, and then press the **"Save"** button.

7. Note that after saving, the existing Console EventHandler and Page Interaction EventHandler are still attached to the Person type:

8. Return to the People list again by using your previously Favorited link, or by clicking the **"Relativity Forms Workshop Fest 2019"** Tab.
9. Click the link in the **"Name"** column of one of your Person objects, ideally one whose **"Contain Enthusiasm"** is the "No, I cannot." (false) value.
10. Note that the **"Maximum Enthusiasm"** Field is still showing, and that no Console is appearing in View mode.
11. Click **"Edit"**
12. Note that **"Maximum Enthusiasm"** is still showing as the page loads, and when user interaction changes the **"Contain Enthusiasm"** value.

(End of Exercise 1)

3.3 What Relativity Forms Enablement Does

After finishing the first exercise, having enabled Relativity Forms on Person, you might have noticed slight UI differences, and that the URL for the layouts have changed. Where Classic URLs for object instances look a bit like these:

- View:
  /Relativity/**Case/Mask/View.aspx**?AppID=1023671&ArtifactID=1038365&ArtifactTypeID=1000042
- Edit/Add:
  /Relativity/**Case/Mask/EditField.aspx**?AppID=1023671&ArtifactID=1038365&ArtifactTypeID=1000042

The URLs in Forms look more like this:

- /Relativity/**External.aspx**?AppID=1023671&ArtifactTypeID=1000042&ArtifactID=1038365&DirectTo=%25ApplicationPath%25%2FCustomPages%2F**FDFEEECF-449D-4C86-8C10-B062F58020C5**%2Findex.html%23view%2F1023671%2F1000042%2F1038365%2F1038364&SelectedTab=null

Relativity Forms is using a front-end application in a Custom Page rather than the Classic ASP.NET webforms pages. However, the far more important effect you're most likely to have noticed is that the

Page Interaction and Console EventHandlers, which are still attached to the Person type, are no longer firing.

While Classic UI for object layouts executes the PopulateScriptBlocks function on Page Interaction EventHandlers to put JavaScript into the page, and fires Console EventHandlers; Relativity Forms instead relies on the Page Interaction EventHandler's "**ScriptFileNames**" property to return an Array of one or more JavaScript file names which provide customization. Because the Page Interaction EventHandler in this application does not specify any ScriptFileNames, the apparent effect is that the EventHandlers are ignored. For information on how ScriptFileNames works, see "How Resource File Hosting Works" in this document's appendix.

4 Customizing Objects with Relativity Forms

4.3 Exercise 2: Customizing Relativity Forms

For this exercise, we will be updating our PageInteractionEventHandler to specify ScriptFileNames, using a template project for creating and testing a very basic Relativity Forms JavaScript file, uploading our new and altered files to Relativity, and examining our results.

Verify Assembly References

1. Back in section 2, we opened our existing application's .NET solution in a Visual Studio. In the Solution Explorer, expand the WorkshopEventHandlers project, and the References node beneath it. If you do NOT see warning icons over the kCura or Relativity references, you can skip to the "Updating the PageInteractionEventHandler" section, below. Otherwise, we must restore these references, continuing with step 2.
2. Right click the "References" node and select "Manage NuGet packages..." from the context menu.
3. In the NuGet Package Manager window, you should see a yellow bar at the top of the window informing you that some packages are missing and prompting you to restore the packages from your online source. Do this by clicking the "Restore" button.
4. Close the NuGet window. The icons near the references shown in the Solution Explorer should refresh on their own. This is cosmetic, but if you don't see this refresh occur, you can force it by clicking the refresh button (a blue circular arrow icon) in the middle of the top of the Solution Explorer.

Updating the PageInteractionEventHandler

As a developer, you have two options for how you would like to tell Relativity Forms about your ScriptFileNames. The first is to create a new PageInteractionEventHandler, implement a mostly empty PopulateScriptBlocks which returns an empty Response, and override its ScriptFileNames property. The second option, which we will follow, is to simply override an existing PageInteractionEventHandler's ScriptFileNames property. Modifying an existing handler like this yields a quicker development turn-around time, keeps the existing handler compatible with Classic, and reduces the work you must do in Relativity itself to get the script files associated with the object type.

1. In Visual Studio's Solution Explorer, select WorkshopPageInteractionEH.cs, which you can find under the WorkshopEventHandlers project.
2. Override the ScriptFileNames property by adding the following line to the class:

```
public override string[] ScriptFileNames => new string[] {
"relativityFormsConversion.js" };
```

3. Save the file.
4. Clean and rebuild the solution.

Relativity Forms JavaScript

In the interest of time saving, this portion of the exercise is provided for you in the following file:

> E:\relativity-forms-workshop-fest-
> 2019\source\RelativityFormsWorkshop\RelativityForms\src\relativityFormsConversion.js

Open this file in Visual Studio code and examine its content. Particularly the comments within.

A supplement to this portion of exercise 2 is available in the appendix in the section titled, "Project Template for Relativity Forms JavaScript." That supplemental exercise introduces a preview to a project template which contains front-end automation for unit testing and checking the code quality of your Relativity Forms JavaScript.

Uploading our Customizations and Viewing them in Relativity Forms

Our task now is to upload our altered EventHandler assembly and our JavaScript file to Relativity as resource files associated to our application.

Uploading the DLL

1. Log into Relativity and navigate to the Resource Files list in the Admin workspace.
2. Filter the list down to only files related to our application by entering "Workshop" into the "Application" column's Field filter; or simply use the "All Relativity Forms Resource Files" View, which is already present in the Relativity instance.
3. Click the "Edit" link in the row for the file with the name "WorkshopEventHandlers.dll".
4. Clear the file in the Edit form, and then click "Choose File" to browse to and select "WorkshopEventHandlers.dll" from the folder path "E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\WorkshopEventHandlers\bin\Debug" which was created at the end of the first part of this exercise.
5. Save the file.

Adding the JavaScript File

1. In the Resource Files list, click the "New Resource File" button.
2. In the Application Field, select our Application ("Relativity Forms Workshop Fest 2019").
3. Click "Choose File" to browse to and select "relativityFormsConversion.js" from the folder path "E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms\src".
4. Save the file.
5. For later time savings, click "Edit" after saving this file, and favorite the Edit form for the relativityFormsConversion.js resource file.

Results and Summary

Now that you have:

1. Altered the application's PageInteractionEventHandler so that it returns an Array of strings for its ScriptFileNames
2. Created the JavaScript file referenced by the ScriptFileNames, correcting its linting and unit test errors in the process

3. Uploaded the altered EventHandler and new JavaScript file as resource files for your application

it is time to see the results.

1. Open your browser's developer tools panel (we're using Chrome, and the shortcut for this is F12) to its Network tab. Be sure that "disable cache" is checked. **For the remainder of this workshop, keep the developer tools panel open**, as it will ensure that your updated JavaScript is used rather than cached values.
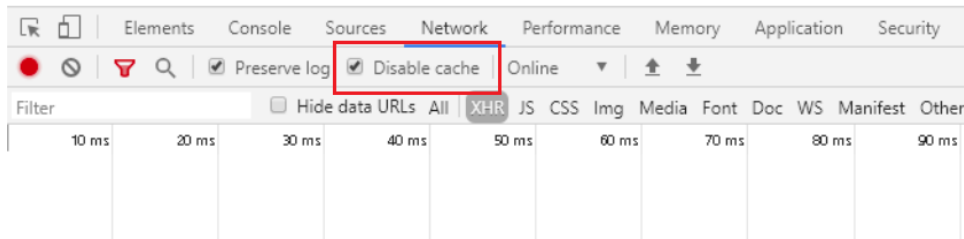
2. Using your Favorite link in Relativity, navigate to your People list.
3. Click a "Name" link for one of the Person objects.
4. After the page loads, note that an empty console now appears in the View layout.
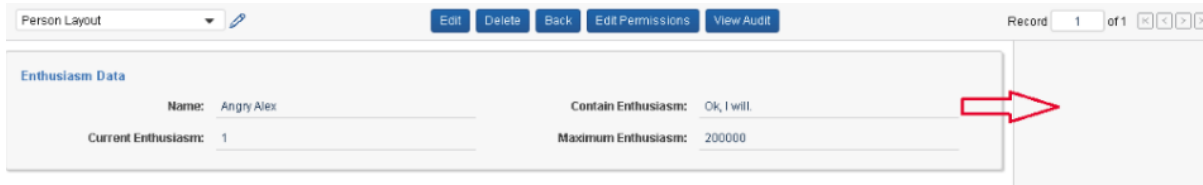
5. Select "XHR" in the developer tools Network tab and enter "relativityFormsConve" in the filter text input.
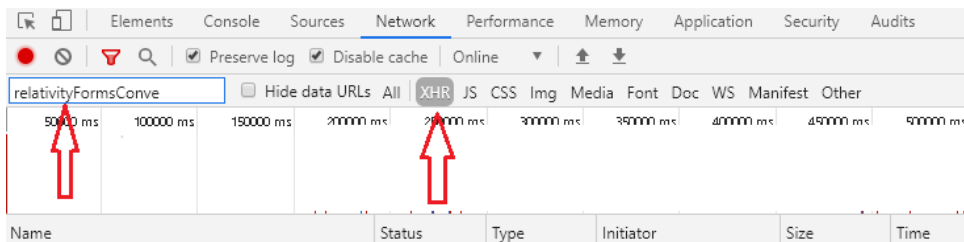
6. You will see that an entry appears for relativityFormsConversion.js in the Network tab.
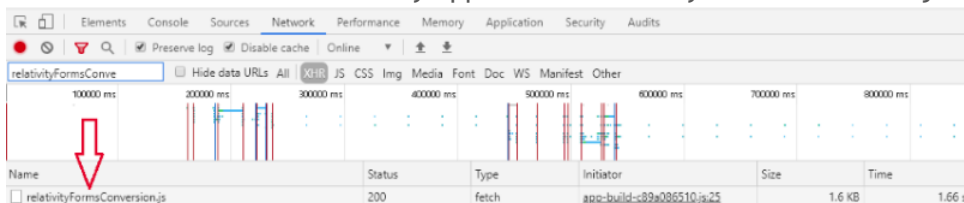
(End of Exercise 2)

5 Developing Relativity Forms Event Handlers

5.1 Page Interaction

After finishing the second exercise, you've seen the basic customization working inside of Relativity Forms; however, the Person object is still missing the Page Interaction functionality to show and hide the "Maximum Enthusiasm" Field, as well as the Console EventHandler's mark up and modest functionality. Our next two exercises will fill these gaps and make our Relativity Forms conversion a bit more capable than the application with which we started.

Relativity Forms executes your customizations at specific points along four main workflows (known in Relativity Forms as "Pipelines") in the page lifecycle. These pipelines are:

- Load
- Change
- Submit
- Delete

There are also a few Item List-related events which don't fit neatly within these four pipelines. You can read more about the pipelines and the Item List-related events here: Relativity Platform documentation for the Relativity Forms API

Already in the second exercise you have added a handler for the CREATE_CONSOLE event. The CREATE_CONSOLE event is fired as a part of the Load pipeline, as does its related event UPDATE_CONSOLE. Supplying Relativity Forms handlers for those two events replaces the Classic .NET ConsoleEventHandler. (Adding useful logic to CREATE_CONSOLE will be covered later in this section.)

Much of what a Classic PageInteractionEventHandler does applies in Relativity Forms to the Load and Change pipelines. In our example application, we are using the value of the "Contain Enthusiasm" Field to drive whether to show or hide the "Maximum Enthusiasm" Field when the page loads (Load Pipeline, HYDRATE_LAYOUT_COMPLETE event) and when the user clicks the "Contain Enthusiasm" Field (Change Pipeline, PAGE_INTERACTION event).

Reading Field Values

In Classic, this behavior requires DOM traversal to find the Field's parent element, and to read or manipulate it. On top of not being robust and being potentially fragile, implementation is not convenient. Different Field types require different approaches for reading values, and even for a single Field type, different approaches must be taken for reading a value for some display types.

Among the features provided to Relativity Forms JavaScript is the convenienceApi, which is a collection of helpful objects, functions, and properties to make event handler implementation easier. Without requiring DOM traversal, the convenienceApi's fieldHelper allows you to read and manipulate your Layout's Fields, addressing them directly by your choice of GUID, ArtifactID, or Name. Regardless of Field type, display type, or form mode, the fieldHelper syntax for reading a Field value is the same, and the value is returned in a data type appropriate to the Field type, without having to be cast or parsed.

Handling Page Interaction

In Classic, in order to respond to a Field's change, you need to manually wire-up the appropriate event listener to the Field's input or other DOM elements surrounding them. Relativity Forms takes the burden of listener attachment off of the developer by wiring all Fields up to the Change pipeline. Handling these event in Relativity Forms entails implementing a handler for the PAGE_INTERACTION event and reading information from the "**modelData**" and "**event**" parameters to determine the details of the change (change events convey the Field involved in the "**event.payload.fieldId**"). Note also that PAGE_INTERACTION events are issued by the both the Change and Submit pipelines. To determine which pipeline caused the event, check the "**event**" parameter's "**type**" property (Values of **event.type** include "change" and "submit").

Because change events presently only identify their issuing Field by ArtifactId, Relativity Forms provides the "this" binding's "**fieldGuidToFieldIdMap**". This Map helps you write handlers which will work in any Workspace.

5.2 Exercise 3: Reading and Manipulating Fields

Implementation: Recommended Steps for Time-Saving

This section is JavaScript-heavy and could be time-consuming for developers more used to C# than JavaScript. In the interest of time-savings, for this exercise, you may simply replace the contents of relativityFormsConversion.js with the contents of the annotated solution for this exercise.

1. Replace the contents of relativityFormsConversion.js with the contents of **exercise3.js.txt** from the same folder and save the file.
2. Review the file's contents, paying close attention to the comments.

Implementation: More Detailed Steps

Controlling Visibility via convenienceApi.fieldHelper.setIsHidden

1. Within your relativityFormsConversion.js immediately invoked function expression (IIFE), add a function called "updateMaximumEnthusiasmVisibility". This function will accept an optional boolean parameter called "showValue" and the function will not return a value.

   ```
   function updateMaximumEnthusiasmVisibility(showValue) {};
   ```

2. Create a local variable called "promise" within updateMaximumEnthusiasmVisibility and, using convenienceApi.promiseFactory.resolve, assign it a promise which resolves to the "showValue" parameter.

   ```
   var field = convenienceApi.fieldHelper; // only aliased for convenience.
   var promise = convenienceApi.promiseFactory.resolve(showValue);
   ```

3. If the type of showValue is not boolean (`typeof showValue !== "boolean"`), reassign the variable **promise**. Use convenienceApi.fieldHelper.getValue to set **promise** to a read of the value for "Contain Enthusiasm" Field, specified by GUID.

```
if (typeof showValue !== "boolean") {
    // If showValue was not provided, look it up.
    promise = field.getValue(CONTAIN_ENTHUSIASM_GUID);
}
```

4. Finally, using convenienceApi.fieldHelper.setIsHidden, add logic which shows or hides the "Maximum Enthusiasm" Field based upon the value resolved from "promise".

```
promise.then(function (containEnthusiasm) {
    field.setIsHidden(
    MAXIMUM_ENTHUSIASM_GUID, !containEnthusiasm);
});
```

Making Use of updateMaximumEnthusiasmVisibility in Response to Page Interaction

1. Add a handler for the PAGE_INTERACTION event to your eventHandlers object.

```
eventHandlers[eventNames.PAGE_INTERACTION] = function(modelData, event){};
```

2. Within the handler, create a variable within the handler called "**containEnthusiasmId**" and assign the value appropriate value from the fieldGuidToFieldIdMap on the "this" binding:

```
        var containEnthusiasmId =
        this.fieldGuidToFieldIdMap.get(CONTAIN_ENTHUSIASM_GUID);
```

3. Next, check that the handler's "**event**" parameter's "**type**" property is "**change**".

```
        if (event.type === "change") {}
```

4. If so, and if **event.payload.fieldId** is equal to **containEnthusiasmId**, invoke your **updateMaximumEnthusiasmVisibility** function, passing **modelData[containEnthusiasmId]** as a parameter.

```
if (event.type === "change") {
  switch (event.payload && event.payload.fieldId) {
    case containEnthusiasmId: {
      updateMaximumEnthusiasmVisibility(modelData[containEnthusiasmId]);
    } break;
    default: break;
  }
}
```

Making Use of updateMaximumEnthusiasmVisibility on Load

1. Add a handler for the HYDRATE_LAYOUT_COMPLETE event to your evenHandlers object. All the handler needs to do is invoke the **updateMaximumEnthusiasmVisibility** function without passing a parameter:

```
eventHandlers[eventNames.HYDRATE_LAYOUT_COMPLETE] = function () {
  updateMaximumEnthusiasmVisibility();
};
```

The HYDRATE_LAYOUT_COMPLETE event happens after the view models for each Field have been resolved. This implementation is safe regardless of form mode, and in most cases, any visibility change which happens will occur before the user is even able to see the markup.

Upload the Updated JavaScript and See Your Results

As mentioned at the end of exercise 2, in the interest of time the workshop is not requiring or covering any unit test updates; however, if you want to run the existing qa or tdd commands against the JavaScript in its current form, this could reduce effort if there's any flaw inside of your code which would prevent the code from executing in Relativity without error.

1. In Relativity, using your previously favorited link to the Edit form for the relativityFormsConversion.js resource file.
2. Clear the file and click "Choose File" to browse to and select "relativityFormsConversion.js" from the folder path "E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms\src".
3. Save the file.
4. Use your favorited link to return to the Person list.
5. Check your event handler functionality:
    a. Click a "Name" Field link of one of your Person objects with the "Contain Enthusiasm" value of "No, I cannot." Verify that the "Maximum Enthusiasm" field is hidden by the HYDRATE_LAYOUT_COMPLETE handler within the form in View mode.
    b. Click the "Edit" button for this object and verify that the "Maximum Enthusiasm" Field is hidden by the HYDRATE_LAYOUT_COMPLETE handler in Edit mode, as well.
    c. Click the "Contain Enthusiasm" Field back and forth between its values to verify that the PAGE_INTERACTION handler is working correctly.

(End of exercise 3)

5.3 Exercise 4: Console and Calling APIs

Your first conversion is nearly complete. All that remains for feature parity with the Classic event handlers is to add some markup to the console, and make the button alert the equivalent of a TODO message. We'll do this, by why stop at a TODO message when the obvious intent was to eventually have a service analyze the Person's level of enthusiasm?

In this exercise we'll take the functionality further. Using newly public functionality on the convenienceApi.relativityHttpClient, we will invoke a Kepler ADS-deployed service to analyze the Person's level of enthusiasm. We'll report the outcome to the user.

Implementation: Recommended Steps for Time-Saving

This section is JavaScript-heavy and could be time-consuming for developers more used to C# than JavaScript. In the interest of time-savings, for this exercise, you may simply replace the contents of relativityFormsConversion.js with the contents of the annotated solution for this exercise.

3. Replace the contents of relativityFormsConversion.js with the contents of **exercise4.js.txt** from the same folder.
4. Save the file.
5. Review the file's contents, paying close attention to the comments.

Implementation: More Detailed Steps

Background Information: Kepler?

Kepler is the name of ADS-deployed services, packaged with your application, and running along-side Relativity's own services on the Service Host. This functionality has only very recently (less than a week ago) been made available to application developers who are not Relativity employees, after several years of Relativity internal use and refinement. Relativity Forms has supported Kepler functionality on the convenienceApi.relativityHttpClient for the last several releases, but have kept its Kepler-oriented functions out of public documentation while the functionality remained internal. Public documentation for the Kepler functionality on the relativityHttpClient is still being finalized, so it is unlikely that it will be available in the Platform documentation by the time of the workshop; however, we will still use the keplerPost function in our implementation of this exercise.

Background Information: API in this Exercise

As mentioned very early within this workshop, the application we're migrating already contains a Kepler service called "PeopleAnalyzer". The service has only one method, **"ReportEnthusiasmAsync"**, which takes as input a ReportEnthusiasmRequest object called "request" and returns a Task<ReportEnthusiasmResponse>

```
public class ReportEnthusiasmRequest{
  public int artifactId { get; set; }
  public string name { get; set; }
  public int currentEnthusiasm { get; set; }
  public int? maximumEnthusiasm { get; set; }
  public bool containEnthusiasm { get; set; }
}



public class ReportEnthusiasmResponse{
  public string summary { get; set; }
}
```

This Kepler service method can be reached via the URL:

<Kepler_Base_Path>/KeplerLab.Services.Interfaces.IEnthusiasticPersonModule/PeopleAnalyzer/ReportEnthusiasmAsync

Background Information: relativityHttpClient.keplerPost

As described above, our service takes a request body, and so we need to POST to this method, and as such we're using keplerPost. The relativityHttpClient also has Kepler-specific functions for GET, PUT, and DELETE calls. Common among these functions is that the function itself is aware of the base path for Kepler APIs and will construct this portion of the URL using information it pulls from Relativity without you as a developer needing to care about it. The Kepler-specific functions also automatically set several required request options so that the developer doesn't need to construct them.

The **keplerPost** function takes three parameters:

- **url** – the URL for the API method (not inclusive of the Kepler base path)
- **payload** – a JSON serializable Array or object to send in the request body
- **requestOptions** – an optional object with any request option overrides or additional properties which the developer wishes to send. (we will not use this parameter in the workshop)

Adding Markup to a Console

Console manipulation is accomplished with the help of convenienceApi.console. We can create markup via functions on convenienceApi.console.generate, and we can append markup to the console area via convenienceApi.console.containersPromise. Code defined within CREATE_CONSOLE and UPDATE_CONSOLE handlers has the same access as the handlers themselves; meaning convenienceApi and its fieldHelper are accessible, which in turn provides access to read and manipulate Fields within the Layout.

1. Add a data access layer object as a variable called **"dal"** within your relativityFormsConversion.js IIFE.

```
var dal = {};
```

2. Add a property called "**reportEnthusiasmAsyncUrl**" to the **dal** object, the value set to the Kepler service path as follows:

```
var dal = {
  reportEnthusiasmAsyncUrl:
"KeplerLab.Services.Interfaces.IEnthusiasticPersonModule/PeopleAnalyzer/Re
portEnthusiasmAsync"
};
```

3. Add a function called "**reportEnthusiasmAsync**" to the **dal** object. The function should accept a number parameter called artifactId, and should not return a value. Initially, this function should simply alert a todo message the same way that the Classic ConsoleEventHandler button does.

```
reportEnthusiasmAsync: function reportEnthusiasmAsync(artifactId) {
  alert("Imagine that I am a report on the current person's enthusiasm");
}
```

4. Update the CREATE_CONSOLE handler to generate markup for
   o Console title:
```
var consl = convenienceApi.console;
var gen = consl.generate;
var title = gen.title({ innerText: "Contain Yourself!" });
```

   o Button:
```
var button = gen.button({ innerText: "Report Enthusiasm" });
```

   o And section:

```
var section = gen.section({}, [button]);
```

5. Add a click handler to the button, which invokes **dal.reportEnthusiasmAsync**.

```
button.addEventListener("click",
  dal.reportEnthusiasmAsync.bind(
    null, this.artifactId));
```

Note here that the "**this**" is the "this" binding available within every event handler.

6. Append the markup to the console's rootElement

```
consl.containersPromise.then(function(containers) {
  containers.rootElement.appendChild(title);
```

```
    containers.rootElement.appendChild(section);
  });
```

If you were to upload this script file now, you would find that the Person object now behaves in Relativity Forms as it does with the Classic ASP.NET event handlers with Relativity Forms disabled. Now, we'll take it the extra mile by calling our Kepler service.

Invoking the Kepler Service via relativityHttpClient.keplerPost

1. Within **dal.reportEnthusiasmAsync**, use [convenienceApi.fieldHelper.getValue](#) and [convenienceApi.promiseFactory.all](#) to obtain the values for all the Person's Fields, "Name", "Current Enthusiasm", "Contain Enthusiasm", and "Maximum Enthusiasm", and fire a "then" once all values have resolved. We don't have a GUID specified in our code for the "Name" Field, so when we get its value, we'll specify the "Name" Field by its name, rather than GUID. This is purely to show that we can.

```
reportEnthusiasmAsync: function reportEnthusiasmAsync(artifactId) {
  var field = convenienceApi.fieldHelper;
  convenienceApi.promiseFactory.all([
    field.getValue("Name"),
    field.getValue(CURRENT_ENTHUSIASM_GUID),
    field.getValue(CONTAIN_ENTHUSIASM_GUID),
    field.getValue(MAXIMUM_ENTHUSIASM_GUID),
  ]).then(function obtainReport(values) {
    // presently does nothing
  });
}
```

2. Within the then created in the previous step (called "obtainReport" above), construct a payload for the service call, and return the invocation of the service via keplerPost

```
  ]).then(function obtainReport(values) {
    var xhr = convenienceApi.relativityHttpClient;
    var url = dal.reportEnthusiasmAsyncUrl;
    var payload = {
      request: {
        artifactId: artifactId,
        name: values[0],
        currentEnthusiasm: values[1],
        containEnthusiasm: values[2],
        maximumEnthusiasm: values[3]
      }
    };
    return xhr.keplerPost(url, payload);
  });
```

3. Following the then from the previous step, add another then and alert the response summary to the user.

```
  return xhr.keplerPost(url, payload);
}).then(function reportSummary(response) {
  alert(response.summary);
}).catch(function reportApiIssue(err) {
  alert(err);
});
```

Upload the Updated JavaScript and See Your Results

1. In Relativity, using your previously favorited link to the Edit form for the relativityFormsConversion.js resource file.
2. Clear the file and click "Choose File" to browse to and select "relativityFormsConversion.js" from the folder path "E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms\src".
3. Save the file.
4. Use your favorited link to return to the Person list.
5. Check your event handler functionality:
    a. Visit each of your Person objects in View mode
    b. Click the console button.

    Note: The process running this application's service may not be warmed-up yet, which could result in service failures with the first several console button clicks.

(End of exercise 4)

6 Closing

This workshop has demonstrated only a small, but important, set of the capabilities of Relativity Forms, its JavaScript APIs, its advantages, and how you as a developer can migrate existing Object Types to Relativity Forms. It is our hope that this introduction to this technology interests you as much as it has excited us to build it and to open it up to the developer community.

Please share your feedback with us at RelativityFormsFeedback@relativity.com

6.2 Questions?

Hopefully, we will finish with a little time to spare for questions. Whether or not we end up with time, we value your feedback, and we want you to succeed and enjoy making use of this new technology. Feel free to contact us with questions in the Relativity DevHelp Community: https://devhelp.relativity.com/

6.2 References and Resources

- **Solutions to the Exercises**
  Did some of the you run out of time during some of the exercises? If so, solutions to each, as well as a completed Workshop solution is available at the following path:

  E:\relativity-forms-workshop-fest-2019\source\WorkshopSolutions

- **Platform Documentation:**
  https://platform.relativity.com/RelativityOne/

- **Relativity Forms:**
  https://platform.relativity.com/RelativityOne/Content/Relativity_Forms/Relativity_Forms_API.htm

- **Relativity Forms Cheat Sheet:**
  https://github.com/relativitydev/relativity-forms-workshop-fest-2019/raw/master/docs/workshop/Relativity%20Forms%20Cheat%20Sheet.pdf

- **This Workshop's Materials on GitHub:**
  https://github.com/relativitydev/relativity-forms-workshop-fest-2019

- **DevHelp:** https://devhelp.relativity.com/c/relativityui/relativityforms
  Category: RelativityUI
  Sub-Category: RelativityForms

- **Feedback:** RelativityFormsFeedback@relativity.com

7 Appendix

7.1 How Resource File Hosting Works

In Relativity 9.6, in support of ListPageInteractionEventHandlers, the overridable properties **ScriptFileNames** and **AdditionalHostedFileNames** were added to the PageInteractionEventHandler (links to 9.6 API documentation) class. Each of these properties specify the names of Resource Files which the application wants Relativity to make available to HTTP requests.

At the same time, an HTTP handler was added to Relativity which intercepts requests coming to paths beginning with **/Relativity/RelativityApplicationWebResourceFile/**. All PageInteractionEventHandlers for the application specified by the request path are checked for the appearance of the requested file name in the Arrays returned by ScriptFileNames and AdditionalHostedFileNames. If an entry exists, Relativity will serve the specified resource file. If no entry is found, even if a resource file by the given name exists, the request is rejected.

Relativity Forms makes use of this same hosting mechanism. When a Relativity Forms page is loaded, the page receives an aggregate list of all associated PageInteractionEventHandlers' ScriptFileNames. Relativity Forms makes a separate request for each file and registers the contents of each upon receipt.

What none of the PageInterationEventHandlers associated to an Object Type specify ScriptFileNames, the apparent effect is that the handlers are ignored.

7.2 Project Template for Relativity Forms JavaScript

A Deeper Supplement to Exercise 2

JavaScript in Classic PageInterationEventHandlers has been difficult to write and harder to test. Relativity Forms aims to make the development and testing of the JavaScript easier, and we're refining a project template with automated testing and code quality enforcement to further aid developers. In this alternative to the JavaScript portion of exercise 2, we'll use a preview of this template. Your task is to examine the template contents, run some simple commands, and update the basic script file into the file we used as a working customization of the Person object type in exercise 2.

Information about the Template

1. In the Visual Studio Code window which we opened near the beginning of this workshop, open the ReadMe.md. This file gives more detailed descriptions of the project template's contents, for later reference if needed.
2. The template's automation is started by commands you'll issue from a command prompt – in this workshop, we're using PowerShell – which are made available via the "scripts" section within the package.json file. You can run code quality checks ("linting") as well as unit tests, individually or in series, as a single run, or continually ("on a watch") in response to file changes.

    Single-run commands

    - **Lint alone:** `npm run lint`
    - **Test alone:** `npm run test`
    - **Lint and Test:** `npm run qa`

    Watch commands

    - **Tests only:** `npm run watch-tests`
    - **Lint and Test:** `npm run tdd`

    The files covered by these commands are the JavaScript files in the **"src/"** folder (the full path of this folder is E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms\src) and in any folder which you might add beneath it. Files with a ".js" extension are code files, which are expected to be written as es5. Those files with a ".spec.js" extension are unit test files, which are es6. The template contains several example code and test files with names which are prefixed with "will_fail_". These files are designed to show problems which the linting and unit test helpers will catch. While you might

want to look at them and experiment with them later, they may be ignored during this exercise. The two script files relevant to the exercise are:

- relativityFormsConversion.js
- relativityFormsConversion.spec.js

The file relativityFormsConversion.js will be our event handler JavaScript file, and relativityFormsConversion.spec.js is the unit test file for it.

The script file is nearly what we need as a basic event handler file, but there are a few errors which linting and unit tests will catch, and we will correct.

Alternative Exercise 2 – Relativity Forms JavaScript

1. In a PowerShell window, navigate to **E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms**

2. Replace relativityFormsConversion.js with _willFail_ex2.js by using the following command

   ```
   mv .\_willFail_ex2.js relativityFormsConversion.js
   ```

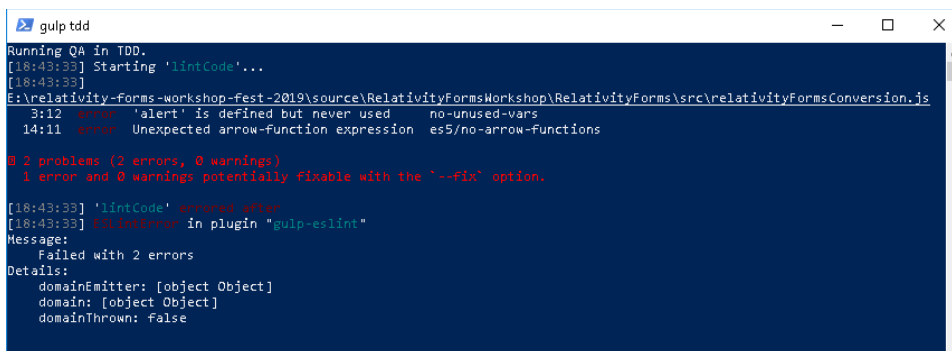3. Begin running linting and unit testing on a watch, using the command

   ```
   npm run tdd
   ```

   This will fail linting almost immediately. You should see the following errors:

   E:\relativity-forms-workshop-fest-2019\source\RelativityFormsWorkshop\RelativityForms\src\relativityFormsConversion.js

   3:12 error 'alert' is defined but never used   no-unused-vars

   14:11 error Unexpected arrow-function expression es5/no-arrow-functions

Correcting Errors: Writing and Testing JavaScript

Linting Issues

1. In Visual Studio Code, open relativityFormsConversion.js. The first error is an unused variable error, referring to the global hint in the file's third line.

2. Remove "alert, " from the third line, and save the file.
3. Look at the PowerShell window, which is still running the tdd task. Note that the linting has run again, the first error is gone, and the other linting error remains. It is an arrow-function, which is a syntax not available in es5, at line 14.
4. Because some of the browsers Relativity supports are incapable of many features of JavaScript made available in es6 and later, and because Relativity Forms does not transpile event handler code into es5, Relativity Forms requires that event handler code be written completely in es5 to begin with. The arrow function in this example was only in the file to expose you to the fact that linting will catch syntax which is not available in es5, so it can be entirely removed by replacing the eventHandlers variable definition (between lines 12 and 16) with this:

```
var eventHandlers = {};
```

then save the file.

Unit Test Issues

1. Look again at the PowerShell window. Linting will have happened again and succeeded, so unit tests will have run and failed.

   This skeleton file, and the unit tests for it were designed to demonstrate the two most basic features of the rdoEventHandlerTestHelpers library: ensuring that your event handler file creates an event handlers object, and surface area verification. This first failed run of tests is due to the file not creating an object. The error left behind reads:

   > An error was thrown in afterAll
   > Error: Expected 'undefined' to be 'object',
   > 'createEventHandlersForFileName(relativityFormsConversion.js, ...) did not return an object.
   > Is this file a well-formed immediately invoked function expression (iife)?
   > Be sure the file's first line is the beginning of the iife, and not a comment or empty line.'.

   As the error message attempts to hint, this particular failure is due to the fact that the first line of the event handler file is not the beginning of the IIFE, but is instead a comment.

2. In order to resolve this failure, remove the first line of the event handler file and any empty lines preceding the first parenthesis in the file (the beginning of the IIFE), then save the file.
3. After fixing the first unit test failure, tdd will rerun linting and unit tests, again yielding a failure in the unit tests. This time, the message reads:

   > relativityFormsConversion.js relativityFormsConversion.js event handlers has the expected surface area FAILED
   > Error: Expected 1 to be 0, 'The following functions are not present on the eventHandlers object: createConsole'.

   This is a failure of the test in relativityFormsConversion.spec.js at line 16. This test is checking to find out whether the object returned by event handler creation has a handler for each of the events named by the EXPECTED_SURFACE_AREA on line 17. Presently, that is only

"createConsole", which is the value of eventNames.CREATE_CONSOLE within the event handler JavaScript file.

4. In order to fix this failure, uncomment the CREATE_CONSOLE event handler stub, between lines 21 and 24, then save the file. This will fix the tests, and the mere existence of the implementation of CREATE_CONSOLE will result in the appearance of a console within View mode, once this file is used on the Person type.
5. Check the PowerShell window again, and you will see that the linting and the unit tests have passed with this last save. The relativityFormsConversion.js file is now ready to be used.

With the current development complete, we can (optionally) stop the template's running automation by breaking out of the tdd watch by pressing Ctrl+C twice, and/or closing the PowerShell window. **You are free to leave this running**, and in an actual development scenario, hopefully that is what a developer would do. However, due to time constraints, we will not update the unit tests from this point on in the workshop, and later additions to our event handler file may cause new linting or unit test errors.

Relativity

Relativity Forms Workshop - 37