ADVANCED FEATURES

Table of Contents

Designing

Accessibility Features in Twixl apps

Both iOS and Android support a number of Accessibility features. Below is an overview of what is supported in your Twixl apps (requires 15.3 or higher).

1. VoiceOver (iOS) & Talkback (Android)

From Apple's web site: "VoiceOver is a screen reader that interacts with objects in your apps so users can drive the interface even if they can't see it. Ensure that the user interface elements in your apps are accessible and useful."

From the Android web site: "Fully interact with what's on your screen through sound and touch. Use TalkBack to hear everything from notifications to app names to how much battery life you have left."

In your Twixl apps, nothing needs to be configured to support this. VoiceOver and Talkback are supported for the following:

- Reading PDF articles
- Reading HTML articles
- User Interface elements (buttons, alerts, ...)
- Text in browse grid cells
- The Settings screen
- The Paywall screen

NOTE: At this time, interactive InDesign content is not supported.

2. Dynamic Type (iOS) & Font size and display size (Android)

With this feature, people can choose their preferred text size and the OS will switch fonts automatically as needed.

In your Twixl apps, nothing needs to be configured to support this. Dynamic Type and Font&Display size are supported for the following:

- Reading PDF articles
- Reading HTML articles
- User Interface elements (buttons, alerts, ...)
- Text in browse grid cells
- The Settings screen
- The Paywall screen

NOTE: At this time, interactive InDesign content is not supported.

3. How to use?

Accessibility features are activated on a device level. Therefore, nothing has to be changed by the end-user in your app itself. Please refer to the Apple and Google help pages on how to activate and use the accessibility features on iOS and Android devices.

> For more general information on mobile accessibility guidelines and tips, check international initiatives as W3C.

Using custom URL Schemes in your app

Custom URL schemes are a powerful way to control navigation from just about anywhere in your app! This article provides a complete overview of what you can do with the different URL schemes.

1. Navigating to a collection or article

From within your article content, you can use special URL schemes to link to a collection or a particular article within a collection. These can also be used in InDesign content using a Hyperlink, a Web viewer or a Web Overlay Button.

You can use the following scheme:

```
tp-collection://[target_collection_name]/[target_article_name]
```

The syntax is as follows:

`tp-collection://[collection-name]`	Will link to the first article in a different collection.
`tp-collection://[collection-name]/[article-name]`	Will link to a specific article in a different collection.
`tp-collection://hamburger-menu`	Will open the Hamburger Menu (provided the Hamburger Menu is enabled).
`tp-collection://parent`	Will link to the parent collection of your current article (currently not supported in the Browser Client).
`tp-collection://root`	Will link to the root collection of your app.

2. Navigating to an article or page

The links below can also be used in your InDesign content, in a hyperlink, a web viewer or a web overlay. It can be used to go to another article in the same collection.

```
tp-pagelink://[article_name]
```

If you want to add a link to a particular page in an article in the same collection from within a browse page or article, you can do so creating an `HREF` like the one below:

```
tp-pagelink://[article_name]/[page_number]
```

An HTML example would be:

```
<a href="tp-pagelink://TOC/3">
```

For InDesign content the article name needs to be the name of the InDesign file. So, in case of the example above, the name of the InDesign document would be TOC.indd.

A number of other URL schemes for relative article and page navigation are also available:

`tp-next-article://`	Go to the next article.
`tp-previous-article://`	Go to the previous article.
`tp-first-article://`	Go to the first article of the publication.
`tp-last-article://`	Go to the last article of the publication.
`tp-next-page://`	Go to the next page.
`tp-previous-page://`	Go to the previous page.
`tp-first-page://`	Go to the first page of an article.
`tp-last-page://`	Go to the last page of an article.
`tp-article-top://`	Go to the top of a long-page article.

`tp-article-bottom://`	Go to the end of a long-page article.

3. Show/hide the toolbar (for InDesign content)

(Available in an InDesign Hyperlink, a Web viewer or a Web Overlay Button)

`tp-toolbar://hide`	Hides the toolbar with the Table of Contents icon, and optional sharing and bookmarking icons.
`tp-toolbar://show`	Displays the toolbar with the Table of Contents icon, and optional sharing and bookmarking icons.
`tp-toolbar://toggle`	Toggles the current view, whether visible or invisible.

4. Show Table of Contents (for InDesign content)

(Available in an InDesign Hyperlink, a Web viewer or a Web Overlay Button)

```
tp-toc://
```

When the `tp-toc://` url is triggered, it will show the Table of Contents dropdown.

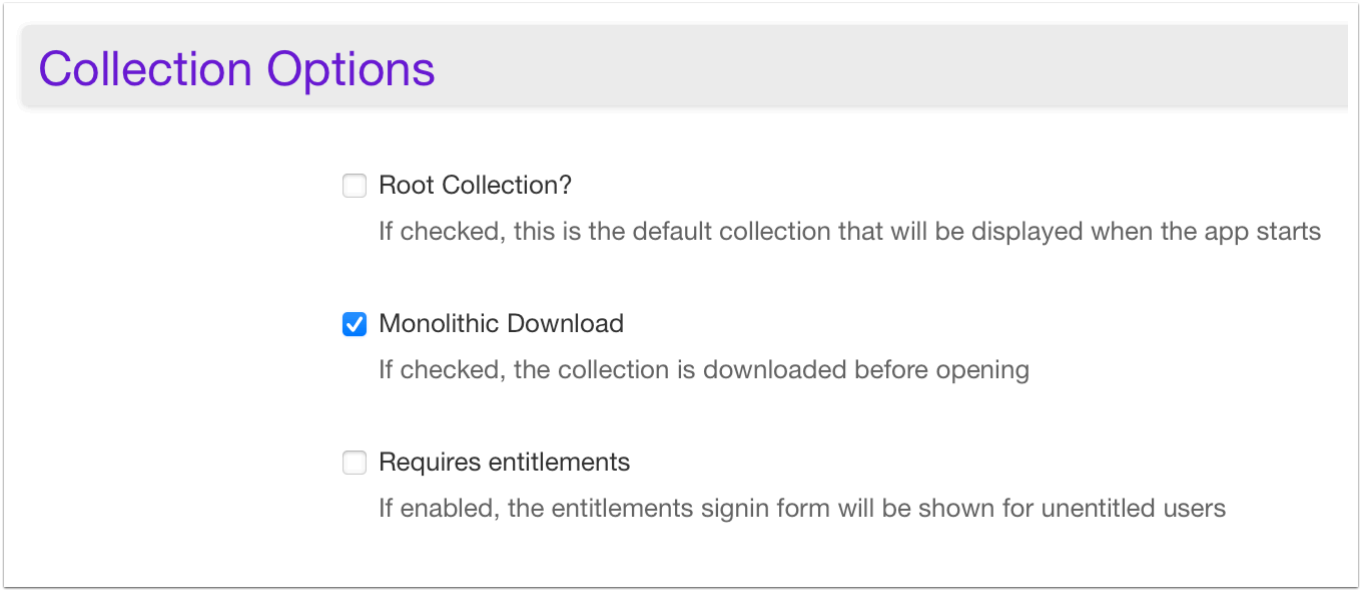
5. Downloads overview

(Can be used in e.g. an InDesign hyperlink, a Web Link content item, or an HTML article.)

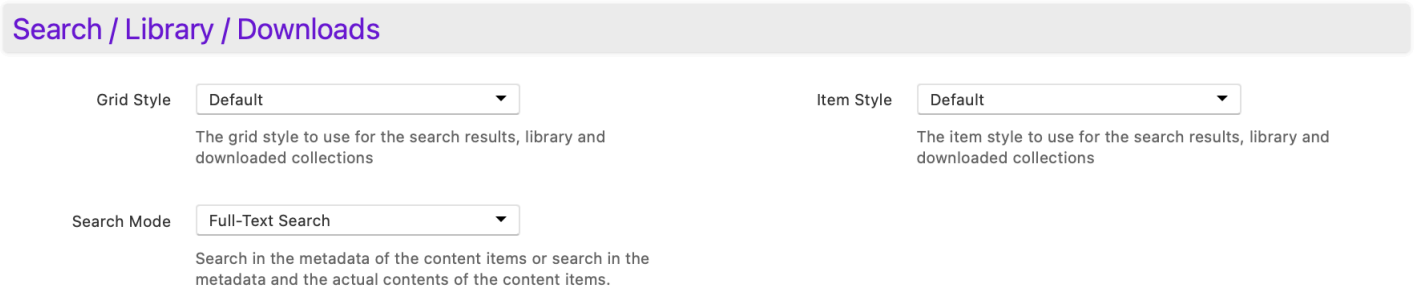
```
tp-downloads://
```

If you have an app where (some or all) collections are marked as **Monolithic Download**, this URL scheme will trigger the display of an overview of the collections that have been downloaded.

5.1. Watch a short 'How to' video...



- The grid & item styles that are used for this collection are defined in your app settings under 'Search / Downloads' (the same styles are used for both the Downloads collection and the Search results).
- Sorting is done by most recent publish date.



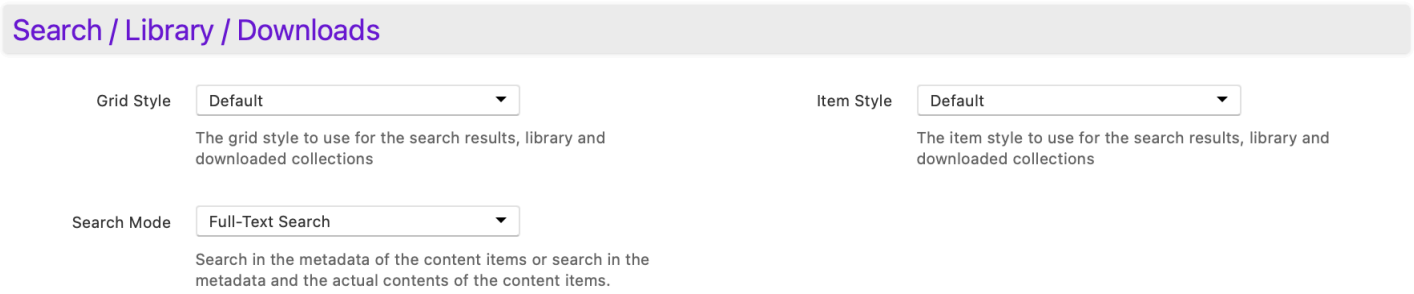
Users can also manage the content that has been downloaded: long-pressing the collection icon of the collection brings up a dialog that allows the user to delete the downloaded content. It can always be re-downloaded later.

6. Library overview

The Library feature allows the publisher to include a section where all the free collections for an app user are displayed, i.e. the free, the purchased and the entitled collections. This feature can be used in e.g. an InDesign hyperlink, a Weblink content item or an HTML article.

```
tp-library://
```

- The grid & item styles that are used for this collection are defined in your app settings under 'Search / Downloads' (the same styles are used for both the Downloads collection and the Search results).
- Sorting is done by most recent publish date.



The Library-feature differs from the Downloads feature as tp-library doesn't show all the offline available (=downloaded) collections. It only shows the for a user freely available collections and this difference should be considered before implementing this feature.

7. Searching

(Can be used in e.g. an InDesign hyperlink, a Web Link content item, or an HTML article.)

To trigger the search-dialog in your app, you use the following url scheme:

Custom URL Scheme	Function
`tp-search://`	Shows an empty dialog
`tp-search://[keyword]`	Executes a search-command with the specified keyword

Example (search for content with the word `twixl`):

```
tp-search://twixl
```

- The grid & item styles that are used for the search result are defined in your app settings under 'Search / Downloads' (the same styles are used for both the Downloads collection and the Search results - see above).

8. Paywall & subscriptions

```
tp-paywall://
```

Enables you to trigger the paywall, showing both purchases and different subscriptions that you have defined.

```
tp-subscriptions://
```

Lets you trigger the paywall, showing only the different subscriptions you have defined.

```
tp-restore-purchases://
```

Lets you trigger the 'Restore purchases' functionality.

9. Entitlements

Show the Entitlements sign-in form

```
tp-entitlements-signin://
```

Triggers the entitlements sign-in form.

Show the Entitlements register form

```
tp-entitlements-register://
```

Triggers the entitlements register form. This only works if the entitlements server provides a "register" action. Available in the kiosk info cell - infoCell.html

Get entitlement token

```
tp-entitlements-get-token://<callback_function>
```

Get the current entitlement token, passing it as the argument to the `callback_function`. If no callback function is specified, it will default to `twixlKioskOnGetEntitlementToken`.

Clear entitlement token

```
tp-entitlements-clear-token://
```

Clears the entitlements token (can be used for testing purposes).

10. Sending e-mail

Please refer to the article Advanced Mailto hyperlinks to learn how to use the `mailto:` URL scheme, to send e-mail from your app.

11. Making a phone call

```
callto:[number] or tel:[number]
```

Allows you to trigger a phone call.

Example:

```
callto:+32493252577

tel:+32493252577
```

12. Sharing on social media

To trigger the sharing sheet, you use the following url-scheme:

```
tp-share://
```

13. Going back in Browsing History

This special url scheme can be used in to go back in the browsing history in an app. The syntax is as follows:

```
tp-history-back://
```

A history will be saved when going though the different content items. This means that if you are in **Article1** and you scroll to **Article2**, if you then press the custom URL tp-history-back:// it will go back to **Article1**.

The history will be saved when:

- Scrolling horizontally to other content items.
- Scrolling horizontally/vertically to pages of the same content item.
- Going to another content item or collection with a custom url (tp-pagelink://article2, tp-next-article, tp-collection://collection2...)
- Going to different pages in the same content item with a custom url (tp-next-page://, tp-last-page://, tp-first-page://...)

14. Going back in Navigation History

This special url scheme can be used in all the same places as `tp-collection` links and goes back in the navigation stack for a specific number of steps. The syntax is as follows:

```
tp-navigate-up://[number-of-steps]
```

The `number-of-steps` indicates how many steps you want to go back.

Example:

Application *XYZ* has the following structure:

- **Root Collection** (browse mode)
  - **Languages Collection** (browse mode)
    - **English Collection** (browse mode)
      - **English Document 1** (detail mode)

Some specific uses and what the result will be (in this case):

- English document 1: `tp-navigate-up://1` takes you to the **English Collection**.
- English document 1: `tp-navigate-up://2` takes you to the **Languages Collection**.
- English document 1: `tp-navigate-up://3` takes you to the **Root Collection**.
- English document 1: `tp-navigate-up://200` also takes you to the **Root Collection**.

IMPORTANT NOTE:

`tp-navigate-up://` is supported only on iOS and Android. It cannot be used in the browser client.

15. Launching an app

It is possible to trigger launching an app from within a Twixl app using the following scheme:

Please note that the implementation is slightly different on iOS vs Android.

- **iOS:** the URL scheme is the same as the application identifier:

  e.g. if the app identifier is `com.twixlmedia.myapp` -> then the URL to launch the app would be: `com.twixlmedia.myapp://`

- **Android:** the URL scheme is the same as the app identifier without the dots, dashes, underscores and all lowercase:

  e.g. if the app identifier is `com.twixlmedia.myapp` -> the the URL to launch the app would be: `comtwixlmediamyapp://`

IMPORTANT NOTE:

This will only work if the app is already installed on the device.

16. Registering a test device for push notifications

`tp-register-test-device://`

Triggering this custom URL scheme will add the current device to the list of test devices for push notifications. Note that once push notifications for your app has been configured, you'll need to create a new build of your app before you can use this URL scheme. More details here.

17. Launching third-party apps

In this scenario, it depends on the url schemes supported by the app you want to launch. E.g. an app which can be opened using the url scheme `com.mycustomapp://` can be launched by simply creating a hyperlink as follows:

```
<a href="com.mycustomapp://">Launch My Custom App</a>
```

IMPORTANT NOTE:

This will only work if the app has already been installed on the device.

18. Opening a hyperlink in the device browser

```
tp-open-in-device-browser=1
```

Although in most cases, you want to keep readers in the embedded browser, sometimes you may want to open a link directly in Safari on iPad or in the default browser on Android. A good use case for this is if you want to link to a PDF that you want readers to be able to download.

You can do this by adding this extra parameter to your URL. Available for Hyperlinks.

Some examples:

The URLs below will open in the embedded app browser:

```
http://www.website.com/file.pdf?id=1
```

```
http://www.website.com/file2.pdf
```

And these will open in the device browser:

```
http://www.website.com/file.pdf?id=1&tp-open-in-device-browser=1
```

```
http://www.website.com/file2.pdf?tp-open-in-device-browser=1
```

19. Open in embedded web browser

```
tp-open-in-web-browser=1
```

Adding this url parameter to a hyperlink will open the link in the embedded web browser instead of opening inline.

20. Get the device info via JavaScript

Adding the following JavaScript code to your Content Item, Web Viewer, Web Overlay Button, ...

```
window.location.href = "tp-device-info://";
```

will execute the following JavaScript function:

```
function twixlOnGetDeviceInfo(deviceUDID, appVersion, appIdentifier, entitlementToken) {
}
```

This then gives you access to the Device UDID, the app version, the app identifier and the entitlement token. You can also customize the name of the function that gets called:

```
window.location.href = "tp-device-info://myCustomFunctionName";
function myCustomFunctionName(deviceUDID, appVersion, appIdentifier, entitlementToken) {
}
```

SAMPLE FILES:

For more info, see the sample files on this GitHub-page.

21. Custom Variables

Custom variables allow you to set or get one or more variables with the following properties:

- They are saved on the device
- They can be re-used in advanced scripting

A sample scenario is e.g. to store a user preference and change the content based on that variable.

```
tp-set-custom-vars://
```

```
tp-get-custom-vars://
```

Example:

```
tp-set-custom-vars://?name=&product=
```

MORE INFO ABOUT CUSTOM VARS:

Custom Variables rely on Advanced Scripting. For more info about Advanced Scripting and an example, see:

- Advanced Scripting: Introduction
- Advanced Scripting | Sample App 6: Custom Vars

Custom URL Schemes: Advanced Mailto hyperlinks

Using Hyperlinks in Twixl InDesign articles, you can launch the reader's mail client and create a new mail message. This article explains how to use the `mailto:` syntax (so you can prefill some fields for your reader) and tackles some difficulties for Android devices.

WARNING:

It appears iOS 14.6 introduced a new behaviour where support for rich content in mailto URLs has been removed, because of potential security issues. Please take this into account when trying to use mailto hyperlinks in your Twixl app.

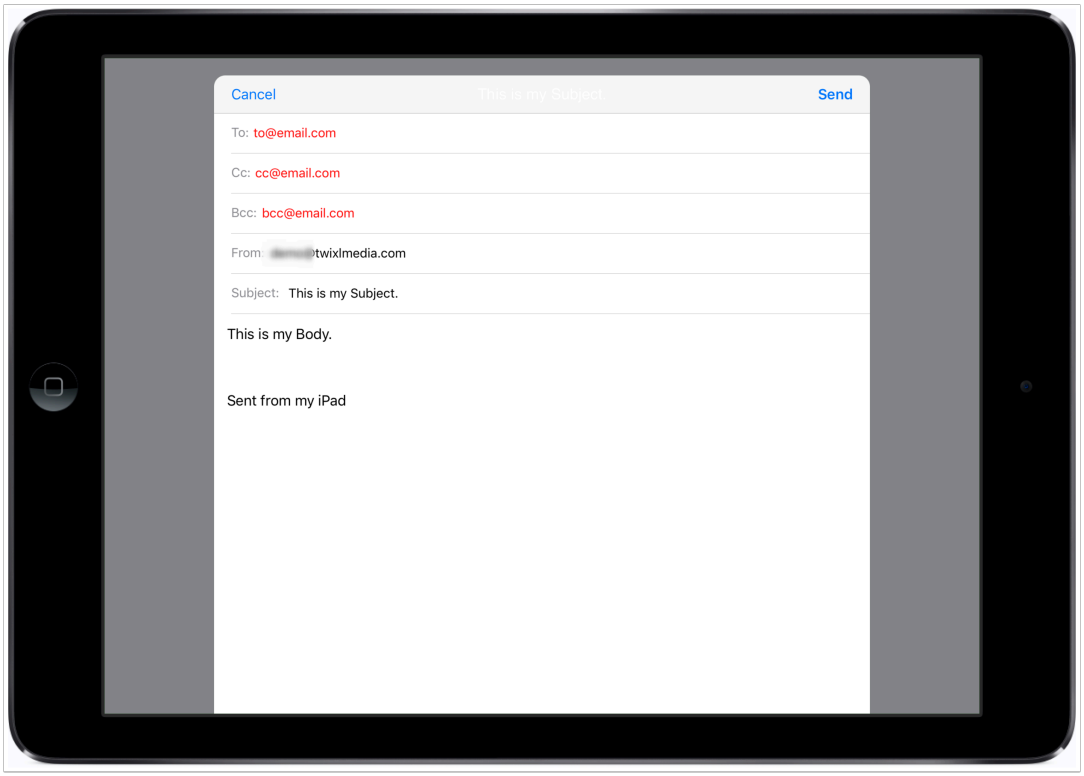
TIP:

For more general info about using Hyperlinks in InDesign, check this article: Hyperlinks

1. General Example

1.1. What?

Let's say you want to achieve the following result: A clickable hyperlink that will prepare a new email for your reader with a pre-defined addressee, subject and body.



Field	Value
To:	to@email.com
CC:	cc@email.com
BCC:	bcc@email.com
Subject:	This is the Subject.
Body:	This is the Body.

1.2. Syntax

The syntax for this example is as follows:

```
mailto:to@email.com?cc=cc@email.com&bcc=bcc@email.com&subject=This%20is%20my%20Subject.&body=This%20is%20my%20Body.
```

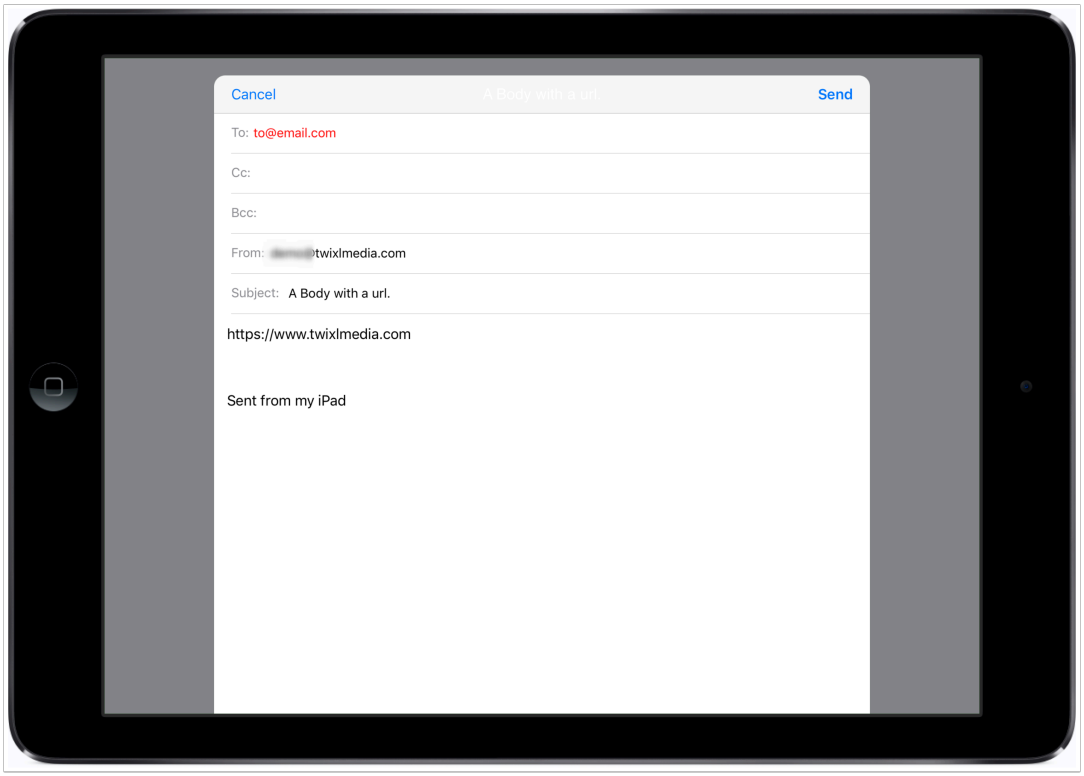
Syntax	Description
`mailto:to@email.com`	This is the startpoint, it opens the reader's mail-client with a new empty mail and

Syntax	Description
	the *To: Field* prefilled with the address `to@email.com`
`?cc=cc@email.com`	This syntax tells that you want to add an option to this url. In this case you want the *CC: Field* prefilled with the address `cc@email.com` . Extra options after the first option needs to be added with `&` as a prefix. The first option after mailto always needs the prefix `?` . **The following options can also be added**:
`&bcc=bcc@email.com`	Option to prefill the *BCC: Field*.
`&subject=This%20is%%20my%20Subject.`	Option to prefill the *Subject: Field*. Don't forget to replace each space by `%20` .
`&body=This%20is%20my%20Body.`	Option to prefill the *Body: Field*. Don't forget to replace each space by `%20` .

2. Example with a url in the predefined body.

Adding a url to the body can be tricky. If you do this with the standard syntax, it will not work on Android devices.

2.1. What?



Field	Value
To:	to@email.com
CC:	
BCC:	
Subject:	A Body with a url.
Body:	https://www.twixlmedia.com

2.2. Syntax

The correct syntax for this example is as follows:

```
mailto:to@email.com?subject=A%20Body%20with%20a%20url.&body=https%3A%2F%2Fwww.
twixlmedia.com
```

So, the tricky part is the url. You can't use a standard url like `https://www.twixlmedia.com` . You will need to **encode** that url first. You can encode a url by using one of these free online tools:

- https://meyerweb.com/eric/tools/dencoder/
- https://www.urlencoder.org
- http://www.url-encode-decode.com

The result for `https://www.twixlmedia.com` will be `https%3A%2F%2Fwww.twixlmedia.com`. It's this encoded url that you need to embed in the syntax of your `mailto:` url.

REMEMBER:

Always test your setup before you publish! More info about *Preview on device* can be found here.

Custom URL Schemes: How to use in HTML articles for the Browser Client

We support the vast majority of Custom URL Schemes for apps in the Browser client, but you'll need to make changes to your code if you want them to work for the Browser Client. This article shows you how to.

Suppose you have this basic HTML-article:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Browser Client Test</title>
</head>
<body>

    <h1 id="title">show collection</h1>

    <script>

        function isBrowserClient() {
            try {
                if (window.name.slice(0, 4) == "twx-") {
                    return true;
                } else if (window.location.hostname.indexOf("twixlmedia.com") > 0) {
                    return true;
                }
            } catch (exception) {
                return false;
            }
        }

        function goToURL(url) {
            if (isBrowserClient()) {
                twxHandleURL(url);
            } else {
                window.location.href = url;
            }
        }
```

```
        var title = document.getElementById("title");
        title.addEventListener('click', function() {
            goToURL("tp-collection://hidden-collection");
        });

    </script>

</body>
</html>
```

When you want to support the *Custom URL Schemes* in a HTML document, you will need to use *JavaScript* to load the URL. We've made two extra functions in the sample below to help with this:

```
function isBrowserClient() {
    try {
        if (window.name.slice(0, 4) == "twx-") {
            return true;
        } else if (window.location.hostname.indexOf("twixlmedia.com") > 0) {
            return true;
        }
    } catch (exception) {
        return false;
    }
}
```

This function checks if the HTML is loaded via the Browser Client or if it's loaded in a mobile app.

```
function goToURL(url) {
    if (isBrowserClient()) {
        twxHandleURL(url);
    } else {
        window.location.href = url;
    }
}
```

This function is a helper to make loading a URL work in both the mobile apps and in the browser client. If it's the Browser Client, we need to use the function `twxHandleURL` (which is always available in the browser client). If it's a mobile app, you can just navigate to the new URL.

The last piece will depend on how you've constructed the HTML:

```
var title = document.getElementById("title");
title.addEventListener('click', function() {
    goToURL("tp-collection://hidden-collection");
```

```
});
```

In our example, we are using plain JavaScript to handle the click event on the `title` element. When the user taps or clicks on the `title` element, it will navigate to the *Collection* named `hidden-collection`.

Reminder: The Twixl Browser Client is a very easy, quick and cheap solution to have an internet extension of your app so users can share your app content with others that don't have your app or to simply view app content while using a computer. However, the Twixl Browser Client does not replace fully designed websites nor has the same specific functionalities as your app might have. In case you want all features on your website, we recommend to use website builders.

Online/offline content in In Design web viewers or web overlays

In a web viewer or web overlay, you can refer to a remote url or to a local WebResource, depending on whether a user is online or not.

With a remote url, the user needs to be online in order to see the contents. With a local WebResource, the contents is embedded in the InDesign article, so you don't have to be online in order to see the contents. Imagine you want to mix both options. As you never know upfront if the user's device is going to be online or not, you would like to have a check that displays a remote URL if the device is online. And if the device is offline, it would be nice if we could show a local WebResource instead. You will need to use some JavaScript to accomplish this though.

The sample file contains a publication with 3 different articles:

- The first article shows a web viewer containing a link to a remote website.
- The second article shows a local WebResource which is embedded in the publication
- The third article is a smart article. It will check if the device is online, and if it is, it will show a link to a remote web site. If the device is offline, it will show a local WebResource which is embedded in the publication.

The way we've implemented it in the third article is by adding an extra file called `check.html` in the WebResources folder for the publication. In that file, we've pasted the following HTML code:

```
<html>

<head>
  <title>Check If The Device is Online or Not</title>
</head>

<body>

  <script type="text/javascript">
    if (navigator.onLine) {
      window.location.href = "http://blog.twixlmedia.com/assets/are-we-online/online.
html";
    } else {
      window.location.href = "offline.html";
    }
  </script>

</body>
```

```
</html>
```

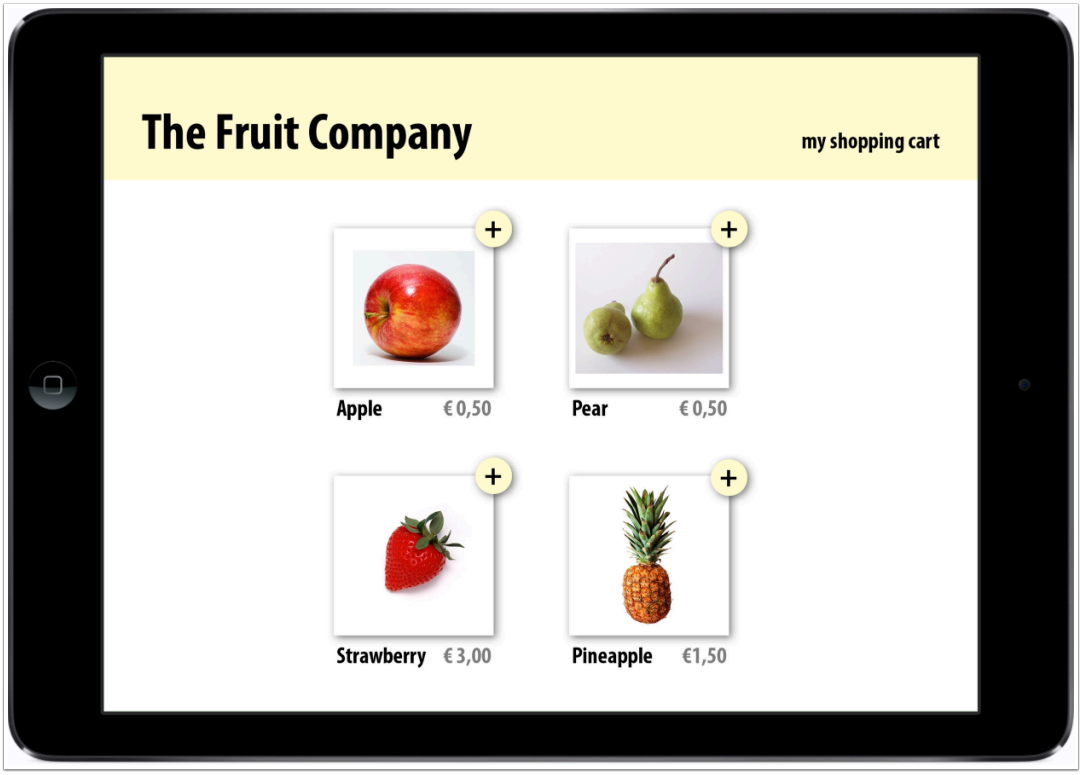
Whenever the page gets loaded, it will execute the script. With the call to `navigator.onLine` (case-sensitive), it will ask the web browser if there is an internet connection. If there is an internet connection, it will redirect to the HTTP url, if not, it will show the local WebResource. If you preview the sample publication on your device, load it first with a network connection. On the third article, it will tell you that an online page was loaded. If you then e.g. turn on Airplane mode in the system settings and open the publication again, the third article will automatically load the offline file. This technique is very handy if you have certain content that is only available online. Using this check, you can show a proper message to the user indicating that he/she needs to be online to view that content.

Download the sample files

How to integrate a form, survey or shopping basket in a Twixl app

InDesign Web Viewers are a very powerful and sometimes underestimated feature that allows you to add dynamic content in a Twixl publication. Any rectangle on an InDesign page can become a source of HTML content. This is not limited to just displaying a specific page of a web site, but can also be used to integrate things like a form, a survey or even a shopping basket that can use localStorage.

In order to explain how this works, we created a **Shopping basket** example.



You can check out the result in the Twixl Viewer by tapping this link on your device (make sure you have the Viewer Classic installed). If you open the Shopping Basket example in the Twixl Viewer, there will be three pages in the publication:

- the first page shows 4 types of fruit, and tapping the "+" icon will add the item to the shopping cart and display the overview of the basket
- the second page provides the same option, but is visualized a bit differently
- the third page shows an overview of the shopping cart, from where the order can be confirmed

How does this work?

The InDesign source publication can be downloaded here. The download consists of:

- an InDesign book called `LocalStorageShoppingCart` with three articles
- a `WebResources` folder with the HTML content

MORE INFO:

1. In **article 1**, each of the 4 types of fruit has a Web Overlay button assigned to it that triggers it to be added to the shopping cart.
2. In **article 2**, the same information is visualized differently using Web Viewers for each of the fruit types, and will allow you to "Add one" to the basket and update the information of the button on the fly.
3. **Article 3** is the overview of the shopping basket, where a user can confirm the order by entering his e-mail address.

The logic for all of this can be found in the contents of the **WebResources** folder, in a document `shopping_cart.html`, along with some JavaScript files.

Embedding a Twitter feed in HTML

When you want to embed content in a .html-article (e.g. a Twitter feed), you need to make to circumvent the `file:` protocol. Let's explain what the problem is and how to tackle that problem.

Don't!

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>Twitter</title>
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link href="css/normalize.css" rel="stylesheet" media="all">
 <link href="css/styles.css" rel="stylesheet" media="all">
</head>
<body>
 <div id="page">
 <div id="twitter-holder">
 <a class="twitter-timeline" data-lang="de" data-width="240" data-height="400" data-
dnt="true"  href="URL-TO-YOUR-TWITTER-FEED"><span id="console">TITLE FOR YOUR NIFTY
TWITTER FEED</span></a>
 </div>
 </div>
 <script src="//platform.twitter.com/widgets.js" charset="utf-8"></script>
</body>
</html>
```

The problem in this example is this piece of code: `<script src="//platform.twitter.com/widgets.js" charset="utf-8"></script>`

In mobile apps, we load the pages using the `file:` protocol. This means it will try to load the script as follows: `<script src="file://platform.twitter.com/widgets.js" charset="utf-8"></script>`. That url of course doesn't exist and that's the reason why you get an error.

Do!

```
<!DOCTYPE html>
```

```
<html>
<head>
 <meta charset="utf-8">
 <title>Twitter</title>
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link href="css/normalize.css" rel="stylesheet" media="all">
 <link href="css/styles.css" rel="stylesheet" media="all">
</head>
<body>
 <div id="page">
 <div id="twitter-holder">
 <a class="twitter-timeline" data-lang="de" data-width="240" data-height="400" data-
dnt="true"  href="https://twitter.com/spitexch?ref_src=twsrc%5Etfw"><span
id="console">Spitex Schweiz auf Twitter. Der Zugang zu diesen Inhalten setzt eine
aktive Internetverbindung voraus.</span></a>
 </div>
 </div>
 <script src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
</body>
</html>
```

The trick is to hardcode the `src`, so you force the mobile app to use the `https` protocol. The nice thing about this, is that this hardcoding will work in your mobile apps, but also in your Browser Client! So the end-result needs to be: `<script src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>`

How to display video in a Browse Grid cell?

If you want to display video inside a Browse Grid cell in article-based apps, read this KB-article!

In short

Basically, you need to wrap a video inside a HTML document using either an Inline Web Viewer or an Embedded Viewer and add an option to the `<video>` tag.

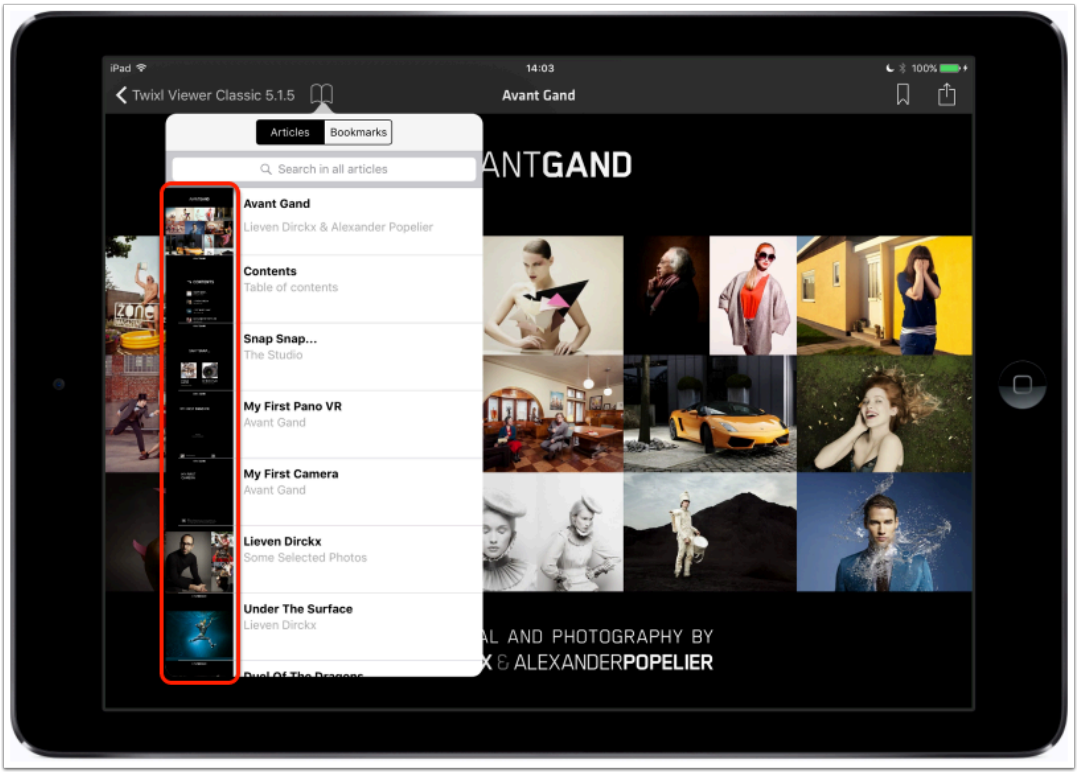
How to

1. Create a .html-file with a link to the video (example below).
2. Link to any video-source (e.g. Vimeo)
3. Add the `playsinline` option to the `<video>` tag. This disables the full screen on phones and this way, the video will be played inside the Browse Grid Cell.
4. Create an Inline Web Viewer or an Embedded Web Viewer. See Working with content items for more info.

Using custom thumbnails in the Table of Contents viewer

When using web viewers, panorama VR or image sequences on a page, these will not be rendered when the Twixl Publisher export creates the thumbnails that are used for the Table of Contents (TOC) overview.

There is a way to customize the thumbnails that are being displayed though…

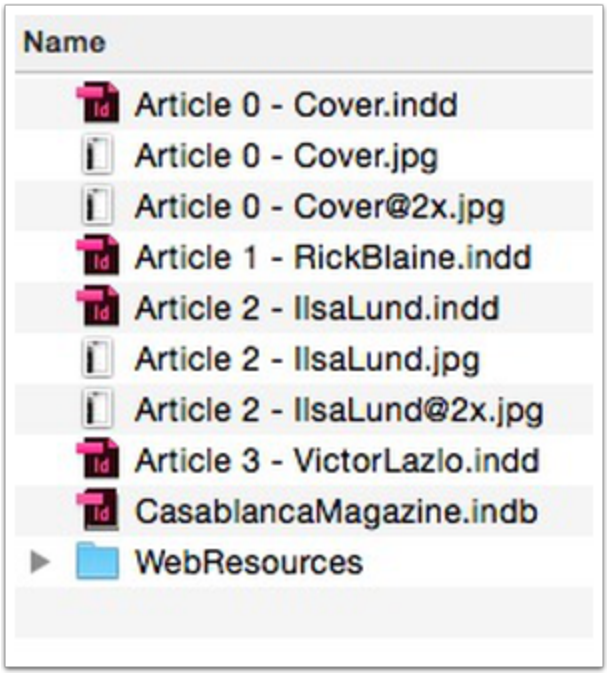


For InDesign-based content

1. The thumbnails that you want to customize need to be placed in the same folder as the InDesign files for the article. If you use a publication-based workflow, it is best practice to keep both the different articles and the publication (InDesign book) file within the same folder.
2. The thumbnail files need to have the same name as the article, but with .jpg as the extension instead of .indd, and have a size of **1024x1024** pixels (or a larger, preferably square, size).
3. The retina thumbnails should add a suffix `@2x` to the file name, e.g. `ArticleName@2x.jpg`, and have a size of **2048x2048** pixels (or a larger, preferably square, size).
4. For articles where no custom thumbnail has been defined, the Twixl Publisher plug-in will generate one on the fly during the Export process.

A sample folder structure should then look as follows:

- `Article1.indd`
- `Article1.jpg` (1024 × 1024 pixels)
- `Article1@2x.jpg` (2048 × 2048 pixels)
- `Article2.indd`
- `Article2.jpg` (1024 × 1024 pixels)
- `Article2@2x.jpg` (2048 × 2048 pixels)
- `MyPublication.indb`



For HTML-articles

The Twixl Distribution Platform will generate thumbnails. Here are the rules you need to take in account:

- If there is an `index.jpg` file in the `.zip` file on the same level as the `index.html` file, that image will be used as a thumbnail
- If there is an `index.jpeg` file in the `.zip` file on the same level as the `index.html` file, that image will be used as a thumbnail
- If there is an `index.png` file in the `.zip` file on the same level as the `index.html` file, that image will be used as a thumbnail
- If no images are present, the Twixl Distribution Platform will look at the first image it can find in the HTML-file.

Resizing Web Content at 100% minus toolbar

If you have HTML Content (either in an InDesign-article, in an HTML-article our some kind of Web Viewer), this trick will help you to calculate the available vertical screen estate, without the toolbar!

What?

The toolbar is not the same on all devices, certainly not on Android devices (due to the vast multitude of available Android devices). The trick is to check the available size in the browser window (which in Twixl terminology is the size of the actual web viewer area, equal to the screen size minus the toolbar)

How?

You can easily do this in JavaScript as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Window Size</title>
</head>
<body>
    <pre><script>
 var windowWidth = window.innerWidth
 || document.documentElement.clientWidth
 || document.body.clientWidth;

 var windowHeight = window.innerHeight
 || document.documentElement.clientHeight
 || document.body.clientHeight;

 document.writeln("window width: " + windowWidth);
 document.writeln("window height: " + windowHeight);
    </script></pre>
```

```
</body>
</html>
```

Using the GPS and Google Maps in your Twixl app

In this KB-article, we'll explain how you can integrate Google Maps into content of your app. The following types of content are supported:

- HTML-articles
- Web Viewers
- Web Overlay Buttons

1. Create a new HTML web page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <title>Google maps - Twixl</title>
        <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

    <style type="text/css">
      html { height: 100%; }
      body { height: 100%; margin: 0; padding: 0; }
      #map_canvas { height: 100%; width: 100%; }
    </style>
 </head>

<body>
        <div id="map_canvas"></div>
</body>
</html>
Explanation:
<div id="map_canvas"></div>
The div map_canvas will be the placeholder for the map.
<style type="text/css">
      html { height: 100%; }
      body { height: 100%; margin: 0; padding: 0; }
      #map_canvas { height: 100%; width: 100%; }
</style>
```

The inline CSS above will make your HTML page scalable. So you can easily change width and height from your canvas (also in InDesign).

2. Add Google API

First you need to provide a link to the Google Maps API and add the initialize function to the header of your HTML page. The URL contained in the `script` tag is the location of a JavaScript file that loads all of the symbols and definitions you need for using the Google Maps API. This `script` tag is required.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=false"></script>

<script type="text/javascript">
 function initialize() {

 var mapOptions = {
 center: new google.maps.LatLng(51.01471, 3.651465),
 zoom: 11,
 mapTypeId: google.maps.MapTypeId.ROADMAP
 };

 var map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);
}
</script>
```

Also edit the body tag in your page so we call the initialize function on page load.

```
<body onload="initialize()">
```

Explanation:

```
var mapOptions = {
center: new google.maps.LatLng(51.01471, 3.651465),
zoom: 11,
mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

In the variable mapOptions you define the map options:

- Latitudes and Longitudes (center the map on a specific point)
- Zoom Levels (Where `zoom 0` corresponds to a map of the Earth fully zoomed out, and higher zoom levels zoom in at a higher resolution.)
- Map Types (`ROADMAP`, `SATELLITE`, `HYBRID` and `TERRAIN`)

```
var map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);
```

When you create a new map instance, you specify a `<div>` HTML element in the page as a container for the map. HTML nodes are children of the JavaScript document object, and we obtain a reference to this element via the `document.getElementById()` method.

3. Add pins to the map

```
var marker = new google.maps.Marker({

position: new google.maps.LatLng(51.01471,3.651465),

map: map,

});
```

To add a default pin to your map you need to add this javascript code in the initialize function after the map variable. Custom pin:

```
var image = 'http://website.com/pin_icon.png';

var marker = new google.maps.Marker({

position: new google.maps.LatLng(51.01471,3.651465),

map: map,

icon: image

});
```

Other options for the marker are:

- `clickable` (boolean)
- `draggable` (boolean)
- `flat` (boolean) -> removes shadow
- `title` (string) -> rollover text
- `zIndex` (number)

4. Events marker & map

```
google.maps.event.addListener(marker, 'click', function() {
 alert('demo alert');
 });
```

You also can add events to your markers and map with: `google.maps.event.addListener` In the event function you can place URL's, alerts, infoWindows, ...

Events:

- `click`
- `dblclick`
- `mouseup`
- `mousedown`
- `mouseover`
- `mouseout`

5. Current GPS location

In order to add the current GPS location, first we change the Google API url to:

```
<script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=true"></script>
```

Then add this after the map variable:

```
// Check for geolocation support
 if (navigator.geolocation) {
 // Get current position
 navigator.geolocation.getCurrentPosition(function (position) {
 // Success!
 var center = new google.maps.LatLng(position.coords.latitude,position.coords.
longitude);

 var image = 'http://i.stack.imgur.com/orZ4x.png';
 var marker = new google.maps.Marker({
 position: center,
 map: map,
 title:"Twixl Demo",
 icon: image,

 });

});
 }
 else {
 // No geolocation fallback:
 markOutLocation(51.01471, 3.651465);
 }
```

6. Overview code

The code below will display a pin with the current location

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
 <title>Google maps - Twixl</title>
 <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

 <style type="text/css">
 html { height: 100%; }
 body { height: 100%; margin: 0; padding: 0; }
 #map_canvas { height: 100%; width: 100%; }
 </style>

 <script src="http://maps.google.com/maps/api/js?sensor=true"></script>

 <script type="text/javascript">
 function initialize() {

 var mapOptions = {
 center: new google.maps.LatLng(51.01471, 3.651465),
 zoom: 11,
 mapTypeId: google.maps.MapTypeId.ROADMAP
 };

 var map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);

 // Check for geolocation support
 if (navigator.geolocation) {
 // Get current position
 navigator.geolocation.getCurrentPosition(function (position) {
 // Success!
 var center = new google.maps.LatLng(position.coords.latitude,position.coords.
longitude);

 var image = 'http://i.stack.imgur.com/orZ4x.png';
 var marker = new google.maps.Marker({
 position: center,
 map: map,
 title:"Twixl",
 icon: image,
 });
 });
```

```
 }
 else {
 // No geolocation fallback:
 markOutLocation(51.01471, 3.651465);
 }
 }
 </script>

 </head>

<body onload="initialize()">
 <div id="map_canvas"></div>
</body>
</html>
```

Advanced Scripting

Advanced Scripting: Introduction

With Advanced Scripting you can add conditions to a Content Item or a Collection

1. Advanced Scripting, what is it?

Using Advanced Scripting, you can link a number of properties available in the app (bundle ID, language, geolocation, ...) and decide whether the Content Item or Collection should be visible or not.

ENTITLEMENT PACK REQUIRED:

To make use of the Advanced Scripting functionality, you need to have the Entitlement Pack activated. This is a paid option. See Specs & Pricing for more info.

2. General info about Advanced Scripting

Advanced scripting is a collection of JavaScript functions that will be evaluated for each collection and/or content item in the context of an application.

It can be used to filter the content items shown in a collection based on a custom set of business rules.

Where can I activate Advanced Scripting?

ENTITLEMENT PACK REQUIRED:

To use the Advanced Scripting functionality, you need to have the Entitlement Pack activated. This is a paid option. See Specs & Pricing for more info.

Although Advanced Scripting is all about filtering Content Items and Collections, you define Advanced Scripting per Collection. As such you need to edit a Collection in order to adapt the Advanced Scripting to your specific needs.

The default script looks as follows:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

IMPORTANT:

Each of the 4 base functions that get executed in the advanced filter currently have a maximum execution time of 2 seconds.

TIP:

You can find a complete syntax and lifecycle overview here. This info also can be found on the Twixl Distribution Platform, below the Advanced Scripting editor window.

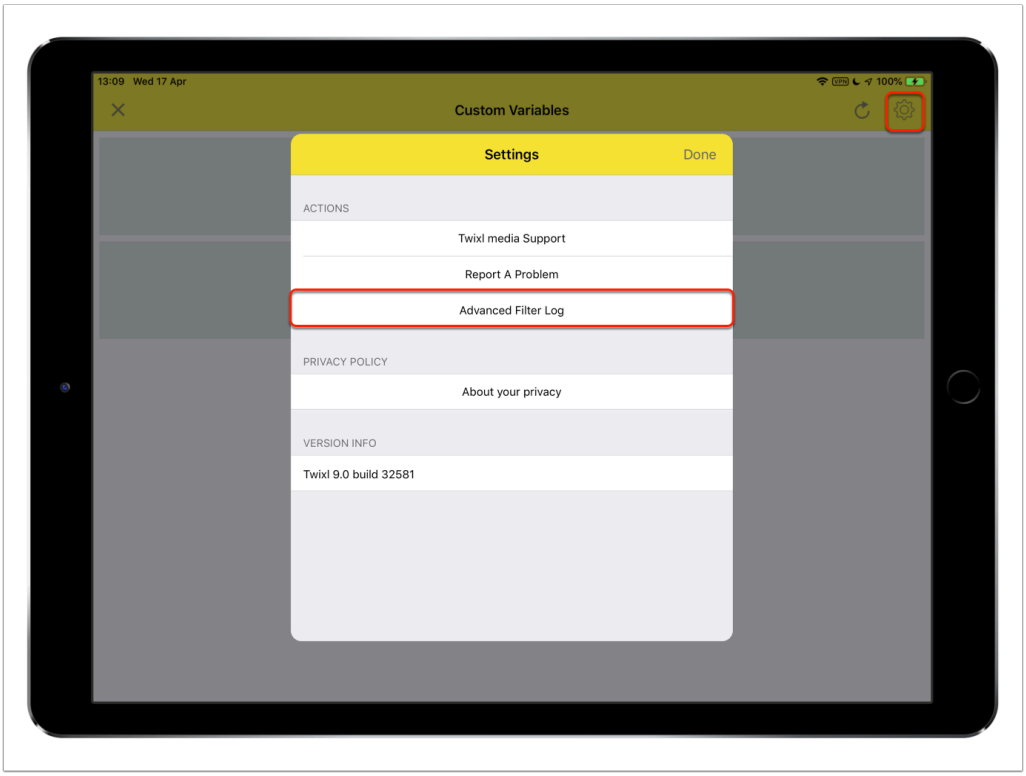
Debugging options

In a mobile article-based app

2.1. Article-based app without a Hamburger Menu

For article-based apps without a *Hamburger Menu*:

1. Tap the *Gear Menu* first
2. Then select *Advanced Filter Log*

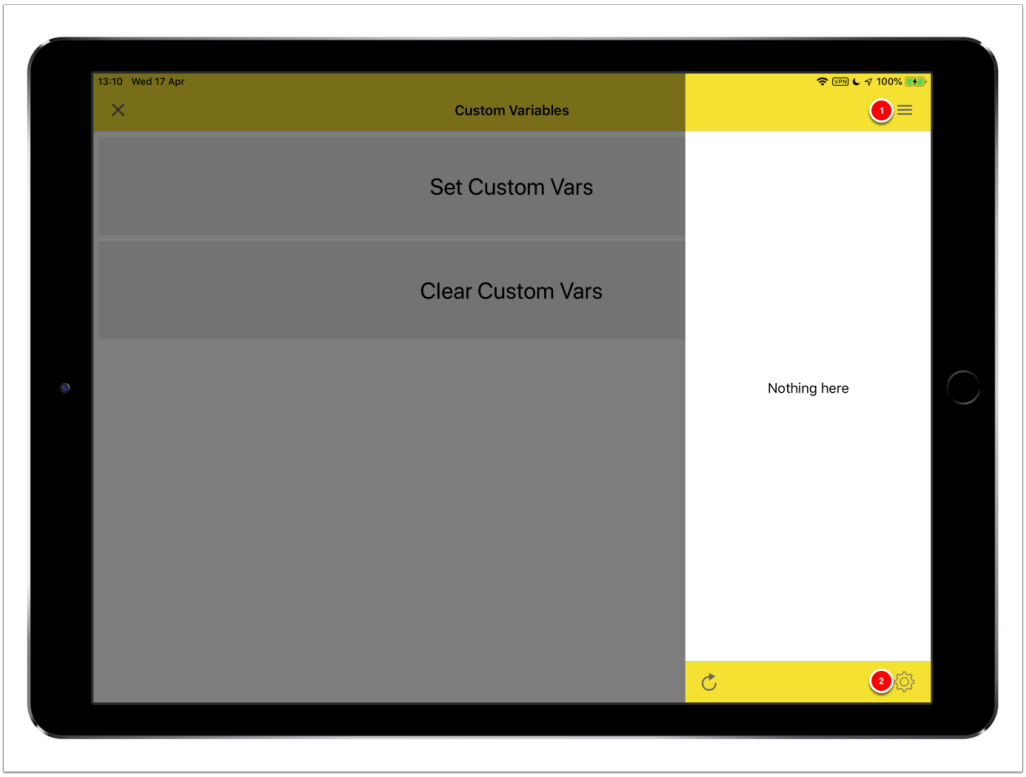


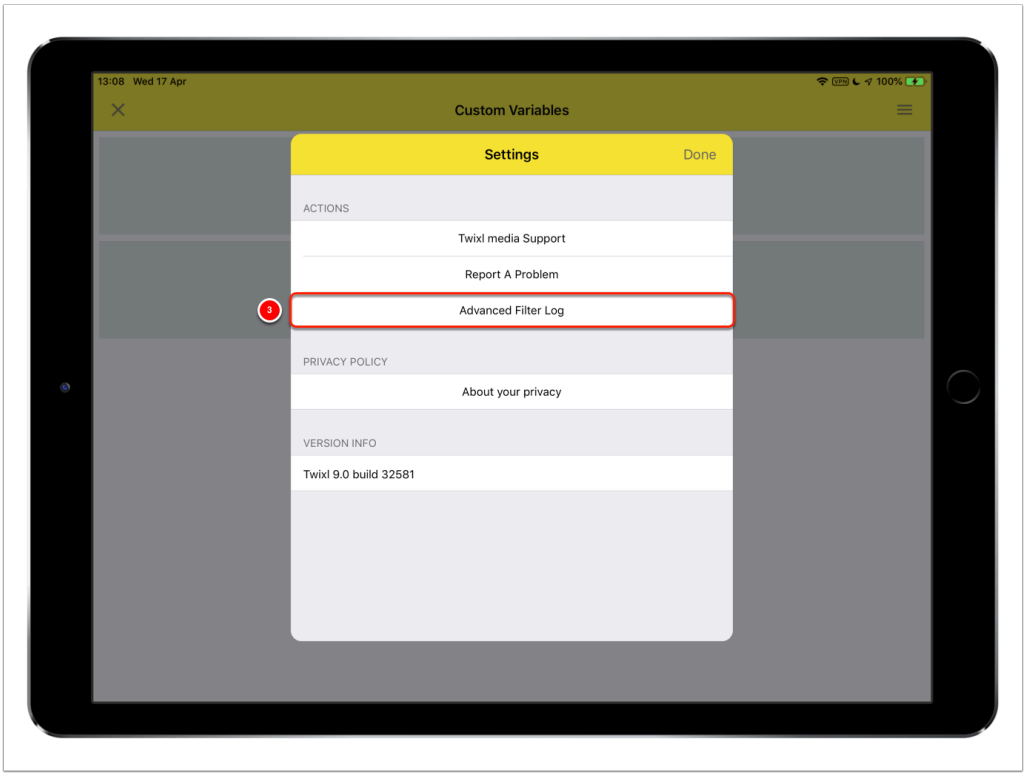
2.2. Article-based app with a Hamburger Menu

For article-based apps with a *Hamburger Menu*:

1. Activate the Hamburger Menu first
2. Select the *Gear Menu*

3. Then select *Advanced Filter Log*





In the Browser Client

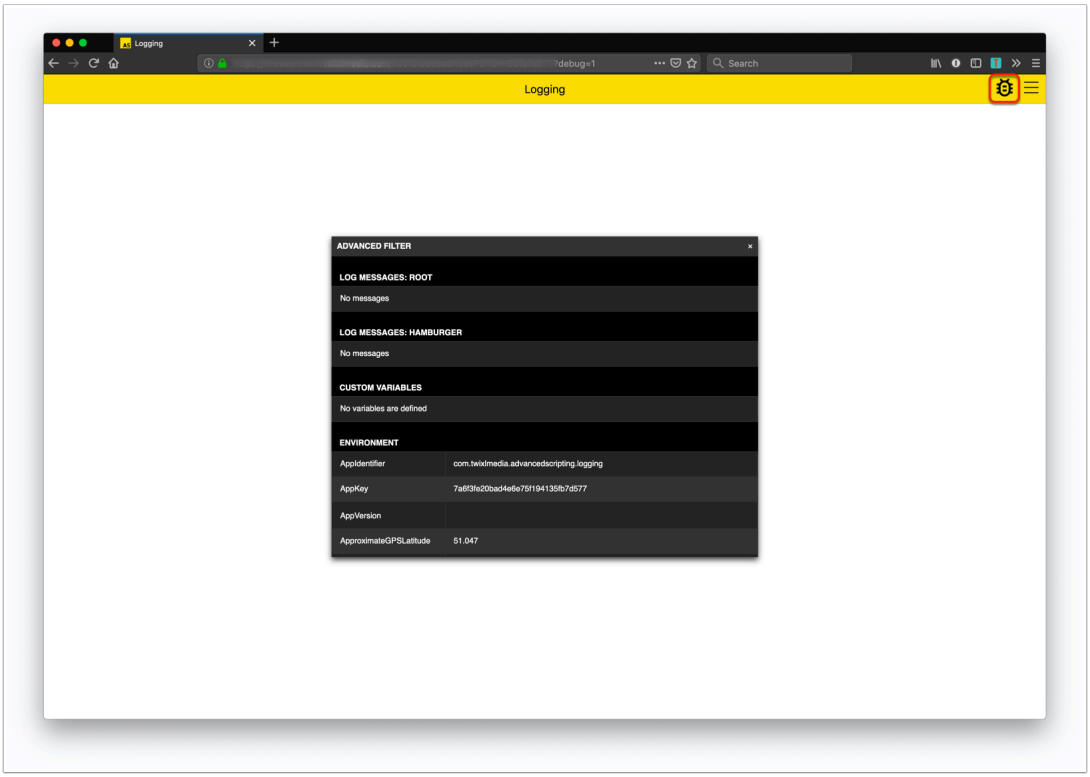
For The Browser Client, you need to add the suffix ?debug=1 and then you'll notice the debugger icon in the toolbar.

So this means, if your Browser Client url is:

```
https://browserclient.twixlmedia.com/<app>
```

Then you'll need to add ?debug=1 in the following way:

```
https://browserclient.twixlmedia.com/<app>?debug=1
```



3. Advanced Scripting Sample Apps

INFO:

We've created several Advanced Scripting Sample Apps for you, each with a specific case that could be useful. They are a good start to explore the immense list of possible scenarios for Advanced Scripting. This list can evolve, and new samples will be added along the way.

- Advanced Scripting | Sample App 1: Login / Logout Button
- Advanced Scripting | Sample App 2: Different content for phone vs tablet vs Browser Client
- Advanced Scripting | Sample App 3: Using twxlog
- Advanced Scripting | Sample App 4: Using JSON and XML
- Advanced Scripting | Sample App 5: Using twxhttp

- Advanced Scripting | Sample App 6: Custom Vars
- Advanced Scripting | Sample App 7: Multiple Languages
- Advanced Scripting | Sample App 8: Privacy Policy

Advanced Scripting | Sample App 1: Login / Logout Button

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

The test Entitlement-account you can use to test this behaviour:

- **Username:** `test`
- **Password:** `test`

Use Case

In this example, we show a Login button in the app when you are not entitled and a Logout button when you are entitled.

Instructions

Preparations

1. Create a new app
2. Setup Entitlements in the app (e.g. Print Subscribers)
3. Add a Content Item to the Root Collection with the following properties:
   - Type: Web Link
   - Name: `login`
   - Link to URL: `tp-entitlements-signin://`
4. Add a Content Item to the Root Collection with the following properties:
   - Type: Web Link
   - Name: `logout`
   - Link to URL: `tp-entitlements-clear-token://`
5. Add a Content Item to the Root Collection with the following properties:
   - Type: HTML Article
   - Name: `whatever`
6. Set up your Advanced Scripting (see below)

Advanced Scripting Settings

For the Root Collection

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (contentItem.Name == "login") {
        return !environment.IsEntitled();
    }
    if (contentItem.Name == "logout") {
        return environment.IsEntitled();
    }
    return true;
```

```
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

Related KB-articles

- Using the Entitlements option for article-based apps
- Custom URL Schemes for article-based apps
- Working with content items
- Working with collections

Advanced Scripting | Sample App 2: Different content for phone vs tablet vs Browser Client

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show different content on phone vs tablet vs the Browser Client

Instructions

Preparations

1. Create a new app
2. Add the following content items in the Root Collection:
   - Placeholder for tablet
     - Type: Placeholder
     - Name: `tablet`
     - Title: `Tablet`
   - Placeholder for phone
     - Type: Placeholder
     - Name: `phone`
     - Title: `Phone`
   - Placeholder for browser
     - Type: Placeholder
     - Name: `browser`
     - Title: `Browser`
   - Placeholder for the other content
     - Type: Placeholder
     - Name: `any`
     - Title: `Any`
3. Setup your Advanced Scripting (see below)

Advanced Scripting Settings

1. Exact naming

For the Root Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}
```

```
// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (contentItem.Name == "browser") {
        return environment.IsBrowser();
    } else if (contentItem.Name == "phone") {
        return environment.IsPhone();
    } else if (contentItem.Name == "tablet") {
        return environment.IsTablet();
    } else {
        return true;
    }
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

2. Non-exact naming

In reality, your Content Items will probably have other names (in order to differentiate) and then it's probably not a good idea to call them all phone, or tablet or browser. In reality, you'll probably work with a suffix, a prefix or a name containing one of the parameters mentioned above.

2.1. Using a prefix

If you want to add a prefix to your existing Name for your Content Item, then you need to use the following code:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (contentItem.Name.startsWith("browser")) {
        return environment.IsBrowser();
    } else if (contentItem.Name.startsWith("phone")) {
        return environment.IsPhone();
    } else if (contentItem.Name.startsWith("tablet")) {
```

```
        return environment.IsTablet();
    } else {
        return true;
    }
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

The name of your Content Item should then be something like:

- **browser**-contentitem01
- **phone**-contentitem01
- **tablet**-contentitem01

2.2. Using a suffix

If you want to add a suffix to your existing Name for your Content Item, then you need to use the following code:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (contentItem.Name.endsWith("browser")) {
        return environment.IsBrowser();
    } else if (contentItem.Name.endsWith("phone")) {
        return environment.IsPhone();
    } else if (contentItem.Name.endsWith("tablet")) {
        return environment.IsTablet();
    } else {
        return true;
    }
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

The name of your Content Items should then be something like:

- `contentitem01-browser`
- `contentitem01-phone`
- `contentitem02-tablet`

Related KB-articles

- The Browser Client
- Working with collections
- Working with content items

Advanced Scripting | Sample App 3: Using twxlog

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how to use the `twxlog` functions and also highlight the lifecycle of the Advanced Scripting. This is ideal to debug while setting up your Advanced Scripting scenario!

Instructions

Preparations

1. Create a new app
2. Enable the Hamburger Menu
3. Add a content item in the Root Collection with the following properties:
   - Type: Placeholder
   - Name: `item-root`
   - Title: `Root Collection Item`
4. Add a content item in the Hamburger Menu Collection with the following properties:
   - Type: Placeholder
   - Name: `item-hamburger`
   - Title: `Hamburger Item`
5. Add a new Collection with the name Sample Collection
6. Add a Content Item in the Sample Collection with the following properties:
   - Type: Placeholder
   - Name: `item-sample`
   - Title: `Sample Collection Item`
7. Set up your Advanced Scripting (see below)
8. Run the app in the Twixl app and then go to the *Gear menu* > **Advanced Filter Log** to see the log messages.
9. For the Browser Client, open the Root Collection, append `?debug=1` to the url and you'll see the debug icon in the toolbar.

Advanced Scripting Settings

For the Root Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
    twxlog.Info("setupFilter: " + collection.Name);
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    twxlog.Info("shouldShowCollection: " + collection.Name);
    return true;
}
```

```
// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    twxlog.Info("shouldShowItem: " + collection.Name + ", " + contentItem.Name);
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
    twxlog.Info("teardownFilter: " + collection.Name);
}
```

For the Hamburger Menu Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
    twxlog.Info("setupFilter: " + collection.Name);
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    twxlog.Info("shouldShowCollection: " + collection.Name);
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    twxlog.Info("shouldShowItem: " + collection.Name + ", " + contentItem.Name);
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
    twxlog.Info("teardownFilter: " + collection.Name);
}
```

For the Sample Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
    twxlog.Info("setupFilter: " + collection.Name);
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    twxlog.Info("shouldShowCollection: " + collection.Name);
    return true;
}
```

```
// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    twxlog.Info("shouldShowItem: " + collection.Name + ", " + contentItem.Name);
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
    twxlog.Info("teardownFilter: " + collection.Name);
}
```

Related KB-articles

- Hamburger Menu (> 5.5)
- The Browser Client
- Working with collections
- Working with content items

Advanced Scripting | Sample App 4: Using JSON and XML

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how to use the `twxxml` and `twxjson` functions. These allow you to parse XML and JSON data structures. There are often used in combination with the twxhttp module as most web services return either JSON or XML.

Some use cases:

1. Based on the Entitlement Token, you can issue for example a request to the Entitlement Server to get more information about the user and change the content based on that information. Your Entitlement Server can output XML or JSON so that the advanced script can easily parse this data.

2. In the `setupFilter`, you might want to issue an HTTP request to your ad system to find out which of your ads are still valid so that you can use that information to hide the invalid ads in your app. If your ad server returns XML or JSON, you can easily parse the response with the provided `twxxml` and `twxjson` modules.

Instructions

Preparations

1. Create a new app
2. Set up your Advanced Scripting (see below)
3. Run the app in the Twixl app and then go to the *Gear menu* > **Advanced Filter Log** to see the log messages.
4. For the browser client, open the root collection, append `?debug=1` to the url and you'll see the debug icon in the toolbar.

Advanced Scripting Settings

For the Root Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {

    // Get some json
    var json = '{"name": "Twixl media", "items": ["item 1", "item 2"]}';

    // Parse the json and log it
    var parsedJson = twxjson.Parse(json);
    twxlog.InfoDump(parsedJson);

    // Encode json
    var data = {"name": "Twixl", "items": ["item 1", "item 2"]};
    twxlog.Info(twxjson.Encode(data));

    // Get some xml
    var xml = '<?xml version="1.0" encoding="UTF-8" standalone="no"?><data><name>Twixl
Publisher</name><items count="2"><item>Item 1</item><item>Item 2</item></items></data>';

    // Parse the xml and log it
    var parsedXml = twxxml.ParseString(xml);
    twxlog.Info(parsedXml.String());
    twxlog.InfoDump(parsedXml.Element("name").Text());
    twxlog.InfoDump(parsedXml.Element("items").Attr("count", 0));
    twxlog.InfoDump(parsedXml.Element("items").Elements("item"));
```

```
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

Related KB-articles

- Working with collections

Advanced Scripting | Sample App 5: Using twxhttp

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how to use the `twxhttp` functions. These allow you to make requests to external websites to obtain information from there. Very handy to get info from internal company-tools, e.g. if your Twixl app is a sales tool!

Instructions

Preparations

1. Create a new app
2. Set up your Advanced Scripting (see below)
3. Run the app in the Twixl app and then go to the *Gear menu* > **Advanced Filter Log** to see the log messages.
4. For the Browser Client, open the root collection, append `?debug=1` to the url and you'll see the debug icon in the toolbar.

Advanced Scripting Settings

For the Root Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {

    // Get a HTTP client
    var client = twxhttp.NewClient();

    // Get the text from a url
    var textURL = "https://demo.twixlmedia.com/advanced-scripting/sample.txt";
    var textResponse = client.Get(textURL);
    twxlog.Info("Text output: " + textResponse.ToString());

    // Get JSON from a url
    var jsonURL = "https://demo.twixlmedia.com/advanced-scripting/sample.json";
    var jsonResponse = client.Get(jsonURL);
    twxlog.InfoDump(jsonResponse.ToJSON());

    // Using the status code
    var badURL = "https://demo.twixlmedia.com/advanced-scripting/sample.invalid";
    var badResponse = client.Get(badURL);
    twxlog.Info("Status code: " + badResponse.StatusCode());

    // HTTP Post
    var postURL = "https://demo.twixlmedia.com/advanced-scripting/post.php";
    var postResponse = client.PostForm(postURL, {"name": "Twixl media", "product":
"Twixl Publisher"});
    twxlog.Info("Post output: " + postResponse.ToString());

    // HTTP Raw Post
```

```
    var rawURL = "https://demo.twixlmedia.com/advanced-scripting/post_json.php";
    var rawResponse = client.PostRaw(rawURL, "application/json", '{"name": "Twixl
media", "product": "Twixl Publisher"}');
    twxlog.Info("Raw Post output: " + rawResponse.ToString());

}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

Related KB-articles

- Working with collections
- The Browser Client

Advanced Scripting | Sample App 6: Custom Vars

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how to use the `tp-set-custom-vars://` function.

Custom variables allow you to set one or more variables with the following properties:

- They are saved on the device
- They can be re-used in the advanced scripting

A sample scenario is e.g. to store a user preference and change the content based on that variable.

Instructions

Preparations

1. Create a new app
2. Add a content item with the following properties to the Root Collection:
   - Type: Web Link
   - Title: `Set custom vars`
   - URL: `tp-set-custom-vars://?name=Twixl&product=Publisher`
3. Add a content item with the following properties to the root collection:
   - Type: Web Link
   - Title: Clear custom vars
   - URL: `tp-set-custom-vars://?name=&product=`
4. Set up your Advanced Scripting (see below)
5. Run the app in the Twixl app and then go to the *Gear menu* > **Advanced Filter Log** to see the log messages.
6. For the Browser Client, open the Root Collection, append `?debug=1` to the url and you'll see the debug icon in the toolbar.
7. After tapping the Set Variables, check the debugger again to see that the values are saved

Advanced Scripting Settings

For the Root Collection

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {

    var customVars = environment.CustomVars();
    twxlog.InfoDump(customVars);

}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return true;
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    return true;
```

```
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

How to use Custom Variables in a HTML-article?

If you want to get a list of Custom Variables from within an HTML-article, you can use the following sample code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>tp-get-custom-vars</title>
</head>
<body>
    <script>

        function twixlOnGetCustomVars(customVariablesAsJSON) {
          console.log(customVariablesAsJSON);
        }

        window.location.href = "tp-get-custom-vars://twixlOnGetCustomVars";

    </script>
</body>
</html>
```

Related KB-articles

- Working with collections
- The Browser Client
- Working with content items

Advanced Scripting | Sample App 7: Multiple Languages

In the series Advanced Scripting Sample Apps we explain some use cases on how to use the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how you can ask the user to select a language and filter Content Items and Collections based on the chosen language.

Instructions

Preparations

1. Create a new app
2. Add the following contents item to the Root Collection with the following properties:
   - Button to select English
     - Type: Web link
     - Name: `select.en`
     - Title: `English`
     - Link to URL: `tp-set-custom-vars://?lang=en`
   - Button to select French
     - Type: Web link
     - Name: `select.fr`
     - Title: `French`
     - Link to URL: `tp-set-custom-vars://?lang=fr`
   - English Collection
     - Type: Collection and Link
     - Name: `en.collection1`
     - Title: `English Collection 1`
   - French Collection
     - Type: Collection and Link
     - Name: `fr.collection1`
     - Title: `French Collection 1`
3. Add a content item to the hamburger collection with the following properties:
   - Type: Web link
   - Name: `Choose Language`
   - Link to URL: `tp-set-custom-vars://`
4. Set up your Advanced Scripting (see below)

Advanced Scripting Settings

For the Root Collection

```
// Create a global variable in which we store the language
var language = "";

// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
```

```
    language = environment.CustomVar("lang").toLowerCase();
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    return collection.Name.startsWith(language);
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (language !== "") {
        return contentItem.Name.startsWith(language);
    }
    return contentItem.Name.startsWith("select.");
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

For all other collections and the hamburger menu where you want to filter by language

```
// Create a global variable in which we store the language
var language = "";

// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
    language = environment.CustomVar("lang").toLowerCase();
}

// Executed for once for every collection
function shouldShowCollection(collection, environment) {
    if (language === "") {
        return false;
    }
    return collection.Name.startsWith(language);
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    return true;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

Related KB-articles

- Working with collections
- Hamburger Menu
- Working with content items

Advanced Scripting | Sample App 8: Privacy Policy

In the series Advanced Scripting Sample Apps we explain some use cases for the Advanced Scripting. **For more general info, it's very important that you read this KB-article first.**

Preview with the Twixl App

You can preview this app by scanning the QR code with the Twixl app.

Use Case

In this example, we show how you can add a Privacy Policy to your app which users have to accept before they are granted access to your app.

Instructions

Preparations

1. Create a new app
2. Add a Cell Style for:
   - The Privacy Policy
   - The button to accept the Privacy Policy
   - The default content
3. Add the following contents items to the Root Collection with the following properties:
   - Privacy policy
     - Type: Embedded Web Viewer
     - Name: `privacy.policy`
     - Title: `Privacy Policy`
     - Content: zipped .html file with the privacy policy
   - Button to accept the Privacy Policy
     - Type: Web link
     - Name: `privacy.accept`
     - Title: `Accept & Continue`
     - Link to URL: `tp-set-custom-vars://?has_accepted=yes&version=20190125`
   - The actual content (just ensure that the name doesn't start with privacy).
4. Set up your Advanced Scripting (see below)

Advanced Scripting Settings

For the Root Collection

```
// Keep a variable to check if the user accepted the privacy policy
var hasAccepted = false;


// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
    hasAccepted = environment.CustomVar("has_accepted") == "yes";
}

// Executed once for every collection
// Root and hamburger are always shown
function shouldShowCollection(collection) {
    return hasAccepted;
```

```
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    if (contentItem.Name.startsWith("privacy.")) {
        return !hasAccepted;
    }
    return hasAccepted;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
}
```

Extra exercise 1 - Clearing the setting

Preparations

1. Add a the following content item to the Hamburger Menu Collection
   - Button to accept the Privacy Policy
     - Type: Web link
     - Name: `<default>`
     - Title: `Clear Privacy Policy`
     - Link to URL: `tp-set-custom-vars://`
2. Set up your Advanced Scripting (see below)

Advanced Scripting Settings

For the Hamburger Menu Collection:

```
// Executed once for every collection before the filtering of the items
function setupFilter(collection, environment) {
}

// Executed for every single content item
function shouldShowItem(contentItem, collection, environment) {
    var hasAccepted = environment.CustomVar("has_accepted") === "yes";
    return hasAccepted;
}

// Executed once for every collection after the filtering of the items
function teardownFilter(collection, environment) {
```

```
}
```

Extra exercise 2

Try to improve the code and add an extra check to find out which version of the privacy policy was accepted. If it's different than what has been accepted before, you should force the user to agree to it again...

Related KB-articles

- Working with collections
- Hamburger Menu
- Working with content items

Extra's

Scripting the Twixl Publisher plug-in

This article expects that you are familiar with Adobe's ExtendScript scripting language.

1. InDesign ExtendScript Labels

The way the Twixl Publisher plug-in stores the interactive properties is by attaching labels to the different elements.

Every object in Adobe InDesign can be provided with one or more labels via de ExtendScript function:

```
object.insertLabel(key, value)
```

To retrieve the label, you can use the extractLabel function using a specific key:

```
object.extractLabel(key)
```

2. Attaching properties

Here's an overview of the type of InDesign elements you can attach properties to, depending on the type of interactive element:

1. **Slide Show:** MultiStateObject
2. **Web Viewer:** Rectangle
3. **Web Overlay:** Rectangle, TextFrame, PageItem
4. **Scrollable Content:** Rectangle with one PageItem as its child element
5. **Full screen image:** Graphic
6. **Movie:** Movie
7. **Sound:** Sound
8. **Image Sequence:** Rectangle

3. Twixl Publisher interactive element keys

Every type of Twixl Publisher interactive element has its own key under which the properties are saved. Here's an overview:

Slide Show	`com.rovingbird.epublisher.mso`
Web Viewer	`com.rovingbird.epublisher.wv`
Web Overlay	`com.rovingbird.epublisher.wo`
Scrollable Content	`com.rovingbird.epublisher.sc`
Full screen image	`com.rovingbird.epublisher.image`
Movie	`com.rovingbird.epublisher.movie`
Sound	`com.rovingbird.epublisher.sound`
Image Sequence	`com.rovingbird.imagesequence`

4. JSON

Under these keys, all properties are saved in a JSON data structure. JSON (http://www.json.org) is a text based format which makes it very easy to store key/value structures.

```
var myProperties = {
    key1: "value",
    key2: "value",
    key3: "value"
};
var myJsonString = JSON.stringify(myProperties);
myObject.insertLabel("com.rovingbird.epublisher.mso", myJsonString);
```

To retrieve it later on, you can use the following code:

```
var myLabel = myObject.extractLabel("com.rovingbird.epublisher.mso");
var myProperties = JSON.parse(myLabel);
```

5. Property overview

5.1. Slide Show

```
{
    'msoShowScrollViewIndicator':            false,
    'msoAllowUserInteraction':               false,
    'msoShowScrollbars':                     false,
    'msoAllowFullScreen':                    false,
    'msoScrollViewIndicatorOpacity':         50,
    'msoScrollViewIndicatorBackgroundColor': '000000',
    'msoScrollViewIndicatorActiveColor':     'FFFFFF',
    'msoScrollViewIndicatorInactiveColor':   'AAAAAA',
    'msoFileFormat':                         'PNG',
    'msoFileFormatIdx':                      0,
    'msoFileFormatDesc':                     '',
    'msoTransitionStyle':                    'Push',
    'msoTransitionStyleIdx':                 0,
    'msoAllowAutoPlay':                      false,
    'msoInterval':                           0,
    'msoDelay':                              0,
    'msoTapPlayPause':                       false,
    'msoAllowLoop':                          false,
}
```

5.2. Web Viewer

```
{
    'wvUrl':                  '',
    'wvAllowUserInteraction': false,
    'wvTransparent':          false,
    'wvScaleToFit':           false,
    'wvShowScrollbars':       false,
    'wvOpenLinksInline':      true,
    'wvShowLoadingIndicator': false,
}
```

5.3. Web Overlay

```
{
```

```
    'woUrl':                    '',
    'woAllowUserInteraction': false,
    'woWidth':                  '',
    'woHeight':                 '',
    'woShowScrollbars':         false,
    'woShowLoadingIndicator': false,
    'woBackgroundColor':        '000000',
    'woBackgroundOpacity':      50,
    'woAnalyticsName':          '',
    'woScaleToFit':             false,
}
```

5.4. Scrollable Content

```
{
    'scAllowScrolling': false,
    'scShowScrollbars': true,
    'scEnablePaging':   false,
    'scEnableZooming':  false,
}
```

5.5. Movies

```
{
    'movieAutoStart':           false,
    'movieShowController':      false,
    'movieLoop':                false,
    'movieShowFullScreen':      false,
    'movieReturnToPosterFrame': false,
    'movieAnalyticsName':       '',
}
```

5.6. Sound

```
{
    'soundAutoStart':     false,
    'soundLoop':          false,
    'soundAnalyticsName': '',
}
```

5.7. Full Screen Images

```
{
    'imageAllowFullScreen': false,
    'imageAnalyticsName':   '',
}
```

5.8. Image Sequences

```
{
    'isqFolder':        '',
    'isqReverse':       false,
    'isqAnalyticsName': '',
}
```

6. Automating preflight, export and preview

Download an example (for .article and .publication files) to see how the preflight, export and preview processes can be scripted:

InDesign-Plugin Scripting Examples

Installing the Helper as a Windows Service

In some cases, it can be helpful to run the Helper as a Windows Service (e.g. when you are using InDesign Server to script some parts of your Twixl Publisher workflow). This KB-article explains how to do install the Helper as a Windows Service.

Run the following command from a windows command prompt with admin privileges:

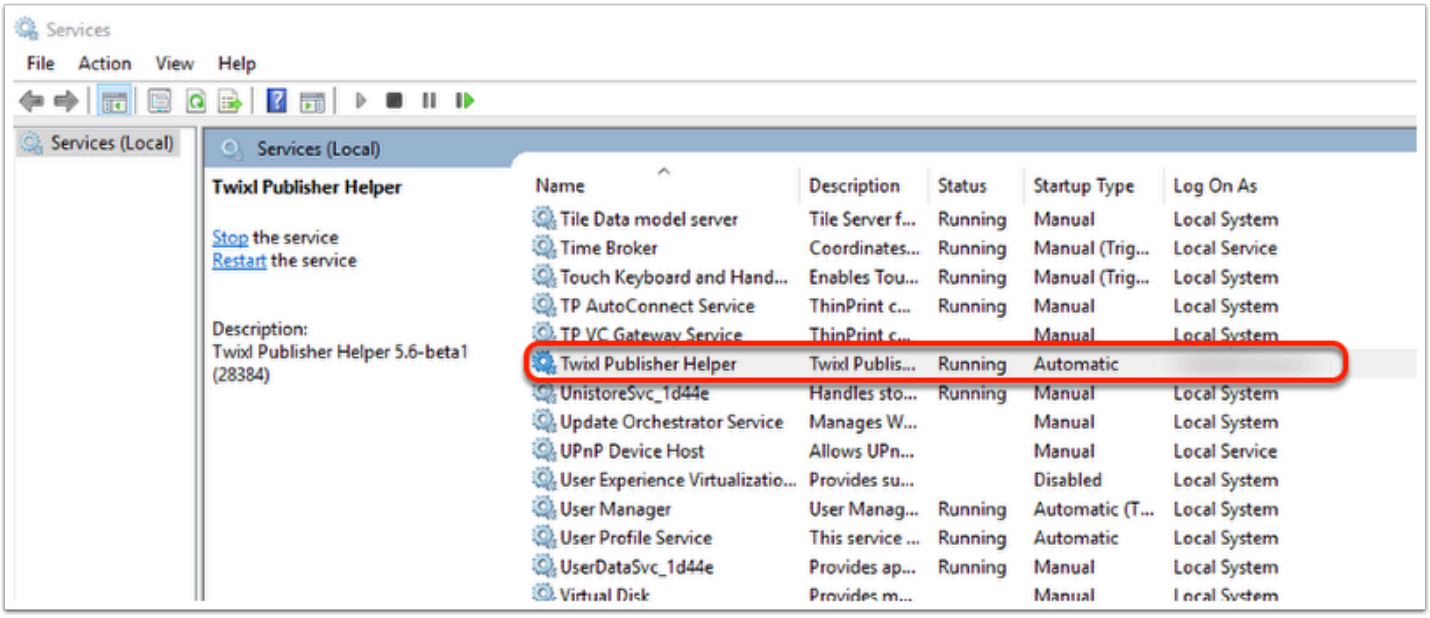
```
C:\Program Files (x86)\TwixlPublisher\TwixlPublisherHelper.exe --service install
```

This will install the *Twixl Publisher Helper* under the `LocalSystem` account. However, it will need to run under the same user account as Adobe InDesign Server to make it work properly.

To do so, go to computer management and select services. Then double click on the `Twixl Publisher Helper` service and change the user in `LoginAs` to the same user account as the one used by InDesign Server.

Once you have done so, you should be able to start it and it should be recognized by the InDesign plugin and InDesign Server, even when no user is logged in to the system.

However, one last thing needs to be taken into account before starting the service. You should check task manager to make sure no `TwixlPublisherHelper.exe` process is running before starting the service, otherwise, it won't work.



IMPORTANT NOTE:

It might be necessary to navigate to the path first (before entering the command):

```
cd Program Files (x86)\TwixlPublisher\TwixlPublisherHelper.exe

TwixlPublisherHelper.exe --service install
```

How to pass a Custom Entitlements Server token to the Browser Client?

In some cases, it might be interesting to pass the login from a Custom Entitlements Server directly to the Browser Client of your article-based apps. This KB-article explains possible reasons (scenarios), requirements and how to do this.

1. Why could this be useful?

Your article-based app is an in-house app and is linked to an internal tool (a CRM, Salesforce, Microsoft Sharepoint, ...). To make content available via the Browser Client, you provide urls in your internal company tool that link directly to the Browser Client without the need for the user to login again.

2. Requirements

What do you need?

- An article-based app with the Browser Client activated
- A Custom Entitlements Server
- Access to the entitlements tokens (a username, an email-address), depending in the specific setup of your Custom Entitlements Server.

ABOUT THE BUILT-IN ENTITLEMENT SCENARIOS:

When you are not using a Custom Entitlements Server, it's technically possible to use this Advanced Feature with one of the built-in Entitlements scenarios. But this will require you to use our API and the setup will be much more complex. Furthermore, most use cases are all about connecting the Browser Client with users from internal company tools. And in the vast majority of those cases, that's why a Custom Entitlements Server is needed anyway.

3. How to setup?

By using the query string paremeter called `twx-set-entitlement-token`, you can tell the Browser Client to use that specific Entitlement Token. It will set the token and force a

redirect to the same url without the entitlement token parameter. As such, you'll need to construct your url as following:

```
https://[browserclient-url]?twx-set-entitlement-token=[token]
```

3.1. Possible examples

```
https://browserclient.twixlmedia.com/app-key?twx-set-entitlement-token=user-1@company-a.com
```

- https://browserclient.twixlmedia.com/app-key : In this case, the url is a default Twixl URL without Custom Domain (see The Browser Client for more info).
- `?twx-set-entitlement-token=` : This query string parameter is necessary to call the next part of the url, the token.
- `user-1@company-a.com` : The token is in this case an email address. This could have been a username or something else (depending on the method used in your inhouse tool).

```
https://demoapp.company-a.com?twx-set-entitlement-token=user1
```

- https://demoapp.company-a.com : In this case, the url is a Custom Domain linking to your Browser Client (see The Browser Client for more info about using Custom Domains).
- `?twx-set-entitlement-token=` : This query string parameter is necessary to call the next part of the url, the token.
- users1 : The token is in this case a username. This could have been an access code, an email address or something else (depending on the method used in your internal company tool).