# 

**Hybrid - Windows tablet - Open PDF from server in a new screen**

Table of Contents

[1 Introduction 4](#_Toc5802793)

[1.1 Definitions 4](#_Toc5802794)

[2 The approach used for opening PDF files in Windows hybrid 4](#_Toc5802795)

[3 The UXP widget used for opening PDF files 4](#_Toc5802796)

[4 The PDF Viewer screen 6](#_Toc5802797)

[5 Internationalization for the PDF Viewer Screen 6](#_Toc5802798)

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Version** | **Date** | **Details** |
| Victor Ionescu | 1.0 | 23.07.2015 | First version |
| Arun Vamadevan | 2.0 | 09.04.2019 | Retargeting for Windows10 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Comments:**

|  |
| --- |
|  |
|  |
|  |

# Introduction

The purpose of this document is to provide a clear and comprehensive guide on how to open a PDF from server in a new screen.

In the hybrid applications, if we need to display static or dynamic PDF content, it’s enough to call window.open(‘<URL of the PDF>’) and the content is displayed properly into a new window that is managed by the native application. This new window is provided by Cordova’s InAppBrowser plugin.

Unfortunately, this approach does not work for Windows hybrid apps, because we cannot use Cordova in these apps (technical constraints). Calling window.open(‘<URL of the PDF>’) in a windows hybrid app would open the external Internet Explorer, which would be ok, but doing so the cookies are not passed to the new instance of Internet Explorer. If the generation of the PDF needs session state on the server, we would get an error, because the JSESSIONID cookie is not transmitted.

This document is intended for developers and testers. It can also be used as a starting point for a future User Guide.

## Definitions

* Server application – the web application generated by UXP. This is deployed to a server and responds to HTTP requests that are launched by the mobile (native) application.
* Native windows application – the native part of a hybrid windows application. The shell is the native windows wrapper that calls the Connect server application. The shell uses a WebView, which is an embedded browser, to display the Connect application.

# The approach used for opening PDF files in Windows hybrid

We chose to make a new screen in the native app, this screen contains a PDF viewer component that allows us to save or print the PDF file.

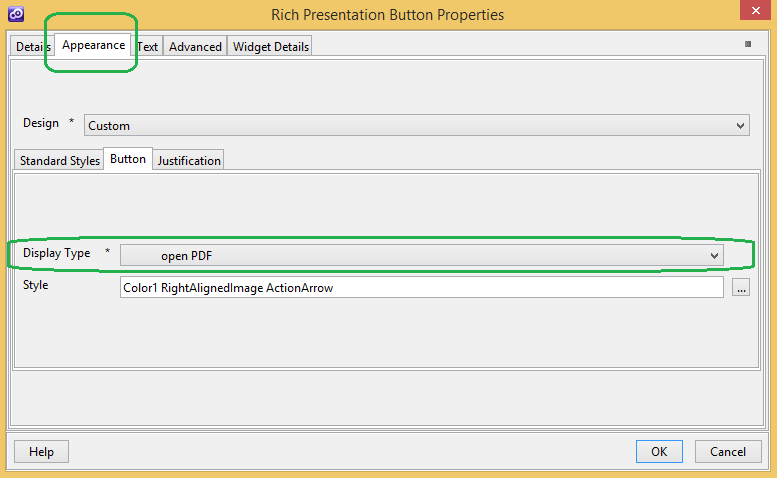
The flow is as follows:

* In the UXP project, we add a widget that contains special javascript code for Windows. This uses the native-javascript bridge provided by Windows Universal apps API.
* The native app receives the notification from the javascript call. The notification message contains the URL of the PDF file on the server.
* The native app downloads the PDF file from the server, by passing all the available cookies. The cookies get stored on the native app upon each page request. This functionality existed before this enhancement, as part of the offline-online navigation.
* The native app opens the PDF file in the PDF viewer screen (PDFViewerPage.xaml).
* When closing the PDF viewer screen, the PDF file is deleted.

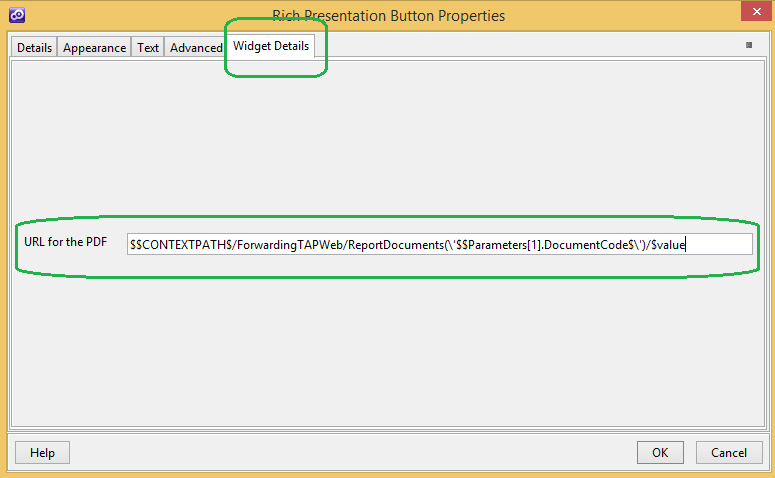
# The UXP widget used for opening PDF files

A widget is needed for communicating with the Windows native app. The widget contains code as well as the widget’s declaration that needs to be copied in your project’s widgets.xml:

Note that this widget is a button widget; you can attach the widget to any button in the presentation editor:



The only thing that is needed is the relative URL of the PDF file (or servlet) in the server app. This has to be entered in the “Widget Details” screen:



Technical note: the widget finds out if the containing app is a Windows hybrid app. If that is the case, the widget invokes

window.external.notify("OPEN\_IN\_APP\_PDF\_NOTIFY"+"$$ITEM.pdfURL$");

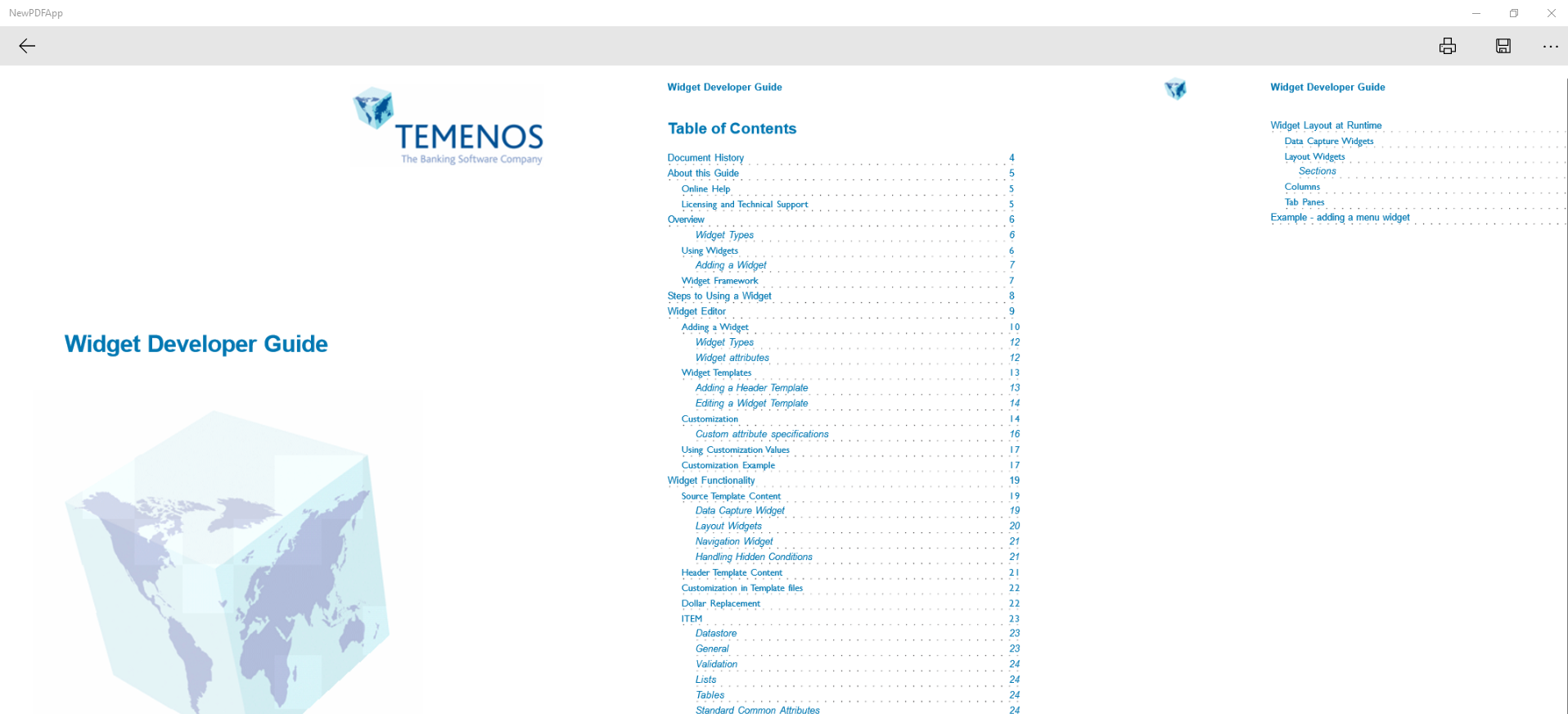
The line above notifies the Windows native app by using the javascript/native bridge. The native app receives the notification and displays the PDF.

If the containing app is not a Windows hybrid app (for ex: android, iOS, web), then window.open() is called for displaying the PDF.

# The PDF Viewer screen

Windows store app does not have a native control to render PDF. So we created a custom control to show PDF files. This control contains a SemanticZoom component with Image as the data template. With the help of newly introduced API - Windows.Data.Pdf, we could render PDF page as Image. PdfPage has a method called RenderToStreamAsync() which will actually do the work.

We will then bind this image in the ItemTemplate of SemanticZoom. We also have the functionality to take print of the PDF file or save them in local machine for future reference.



# Internationalization for the PDF Viewer Screen

The PDF Viewer screen contains a few labels that can be internationalized. This can be achieved by using the language maps editor on UXP.

The message IDs and the default text for the labels are:

filesave\_success= File saved successfully

pdf\_documents= PDF Documents

print= Print

reports= Reports