# Adding layers to PDF page using optional content

**Written by Apitron Documentation Team**

## Introduction

While working on exporting the PDF document, sometimes you need many versions of the same content to be on one page along with an option to show only one version at once.

A multilanguage report or manual are perfect examples of such documents. Instead of producing a separate file for each language you could create a single file which would contain all the necessary information. A user would be able to switch content versions with a single click by selecting the appropriate layer.

Another example of layered content structure is an engineering drawing or complex schema composed of different logically separated parts which could be made visible or invisible on demand.

All these things are made possible using PDF feature called *optional content* - see the section "*8.11 Optional Content"* of the PDF specification for the details. The Apitron PDF Kit .NET component provides an API for layers manipulation and creation. Using this product you can easily create layered content in your PDF documents.

In general, the creation of the multiple layers on PDF page looks as follows:

1. Create several `OptionalContentGroup` objects and register them as document resources – these objects represent layer identifiers in PDF.

2. Create the `OptionalContentConfiguration` object, set its properties controlling the behavior and visual layer structure shown in reader's UI. This object combines layers together and you can use it to define initially visible layers, locked layers, layers that should work as radio buttons etc. You can also define the visual tree structure – parent layer nodes and child nodes.

3. Create and initialize the `OptionalContentProperties` object required by the FixedDocument object – this object is used to define the default configuration to be used by the PDF reader to show layers, and to specify the list of layers (OptionalContentGroups resource ids) actually referenced in document's content (cause not all registered layer ids may be in use).

4. Use `ClippedContent` objects to define the layers and assign their `OptionalContentID` property to the one of the registered layer ids (Optional Content Group resource IDs). Put these objects on PDF page using Page.Content.AppendContent(…) method.

The code demonstrating these steps can be found in the next section.

## The code

```
class Program
{
    static void Main(string[] args)
    {
        using (Stream stream = File.Create("manual.pdf"))
        {
            // create our PDF document
            using (FixedDocument doc = new FixedDocument())
            {
                // turn on the layers panel when opened
                doc.PageMode = PageMode.UseOC;

                // register image resource
                doc.ResourceManager.RegisterResource(
                    new Apitron.PDF.Kit.FixedLayout.Resources.XObjects.Image(
                    "chair","../../data/chair.jpg"));

                // FIRST STEP: create layer definitions,
                // they should be registered as document resources
                OptionalContentGroup group0 = new OptionalContentGroup("group0","Page layers",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group0);

                OptionalContentGroup group1 = new OptionalContentGroup("group1", "Chair image",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group1);

                OptionalContentGroup group2 = new OptionalContentGroup("English", "English",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group2);

                OptionalContentGroup group3 = new OptionalContentGroup("Dansk", "Dansk",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group3);

                OptionalContentGroup group4 = new OptionalContentGroup("Deutch", "Deutch",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group4);

                OptionalContentGroup group5 = new OptionalContentGroup("Русский", "Русский",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group5);

                OptionalContentGroup group6 = new OptionalContentGroup("Nederlands", "Nederlands",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group6);

                OptionalContentGroup group7 = new OptionalContentGroup("Français", "Français",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group7);

                OptionalContentGroup group8 = new OptionalContentGroup("Italiano", "Italiano",
                    IntentName.View);
                doc.ResourceManager.RegisterResource(group8);

                // SECOND STEP:
                // create the configuration, it allows to combine the layers together in any order

                // Default configuration:
                OptionalContentConfiguration config = new OptionalContentConfiguration(
                    "configuration");
```

```
            // add groups to lists which define the rules controlling their visibility
            // ON groups
            config.OnGroups.Add(group0);
            config.OnGroups.Add(group1);
            config.OnGroups.Add(group2);

            // OFF groups
            config.OffGroups.Add(group3);
            config.OffGroups.Add(group4);
            config.OffGroups.Add(group5);
            config.OffGroups.Add(group6);
            config.OffGroups.Add(group7);
            config.OffGroups.Add(group8);

            // lock the image layer
            config.LockedGroups.Add(group1);

            // make other layers working as radio buttons
            // only one translation will be visible at time
            config.RadioButtonGroups.Add(new[] { group2, group3, group4, group5, group6, group7,
                group8 });

            // show only groups referenced by visible pages
            config.ListMode = ListMode.VisiblePages;
            // initialize the states for all content groups
            // for the default configuration it should be on
            config.BaseState = OptionalContentGroupState.On;
            // set the name of the presentation tree
            config.Order.Name = "Default config";
            // create a root node + sub elements
            config.Order.Entries.Add(group0);
            config.Order.Entries.Add(new OptionalContentGroupTree(group1, group2, group3,
                group4, group5, group6, group7, group8));

            // FINAL step:
            // assign the configuration properties to document
            // all configurations and groups should be specified
            doc.OCProperties = new OptionalContentProperties(config, new
                OptionalContentConfiguration[] {}, new[] { group0, group1, group2, group3,
                group4, group5, group6, group7, group8 });

            // create page and assing top layer id to its content
            // it will allow you to completely hide page's
            // content using the configuration we have created
            Page page = new Page();
            page.Content.OptionalContentID = "group0";

            // create image layer
            ClippedContent imageBlock = new ClippedContent(0, 0, 245, 300);
            // set the layer id
            imageBlock.OptionalContentID = "group1";
            imageBlock.AppendImage("chair", 0, 0, 245, 300);

            // put the layer on page
            page.Content.SaveGraphicsState();
            page.Content.Translate(0, 530);
            page.Content.AppendContent(imageBlock);
            page.Content.RestoreGraphicsState();

            // append text layers
            AppendTextLayers(page);
            // add the page to the document and save it
            doc.Pages.Add(page);

            doc.Save(stream);
        }
    }
    Process.Start("manual.pdf");
}
```

```
    static void AppendTextLayers(Page page)
    {
        page.Content.SaveGraphicsState();
        page.Content.Translate(250, 325);

        // evaluate each property of a resource dictionary and add text to the PDF page
        foreach (PropertyInfo info in typeof(strings).GetRuntimeProperties())
        {
            if (info.PropertyType == typeof(string))
            {
                ClippedContent textContent = new ClippedContent(0, 0, 300, 500);
                // assign layer id
                textContent.OptionalContentID = info.Name;
                textContent.Translate(0, 0);

                // preprocess parsed elements and set additional properties
                // for better visual appearance
                IEnumerable<ContentElement> elements =
                    ContentElement.FromMarkup((string)info.GetValue(null));

                foreach (Br lineBreak in elements.OfType<Br>())
                {
                    lineBreak.Height = 10;
                }

                foreach (Section subSection in elements.OfType<Section>())
                {
                    subSection.Font = new Apitron.PDF.Kit.Styles.Text.Font("HelveticaBold", 14);
                }

                // draw text
                textContent.AppendContentElement(new Section(elements), 300, 500);
                // put the text layer on page
                page.Content.AppendContent(textContent);
            }
        }

        page.Content.RestoreGraphicsState();
    }
}
```

You can see that we used ContentElements from FlowLayout API to prepare translated text blocks, more information about the Fixed and Flow layout API can be found by this link.

We exactly followed the algorithm described in the *Introduction* section:

1. Created layer identifier resources and registered them

2. Created default layers configuration in a form of tree view and configured layers to work as radio buttons, except the image layer which we marked as locked to demonstrate this feature

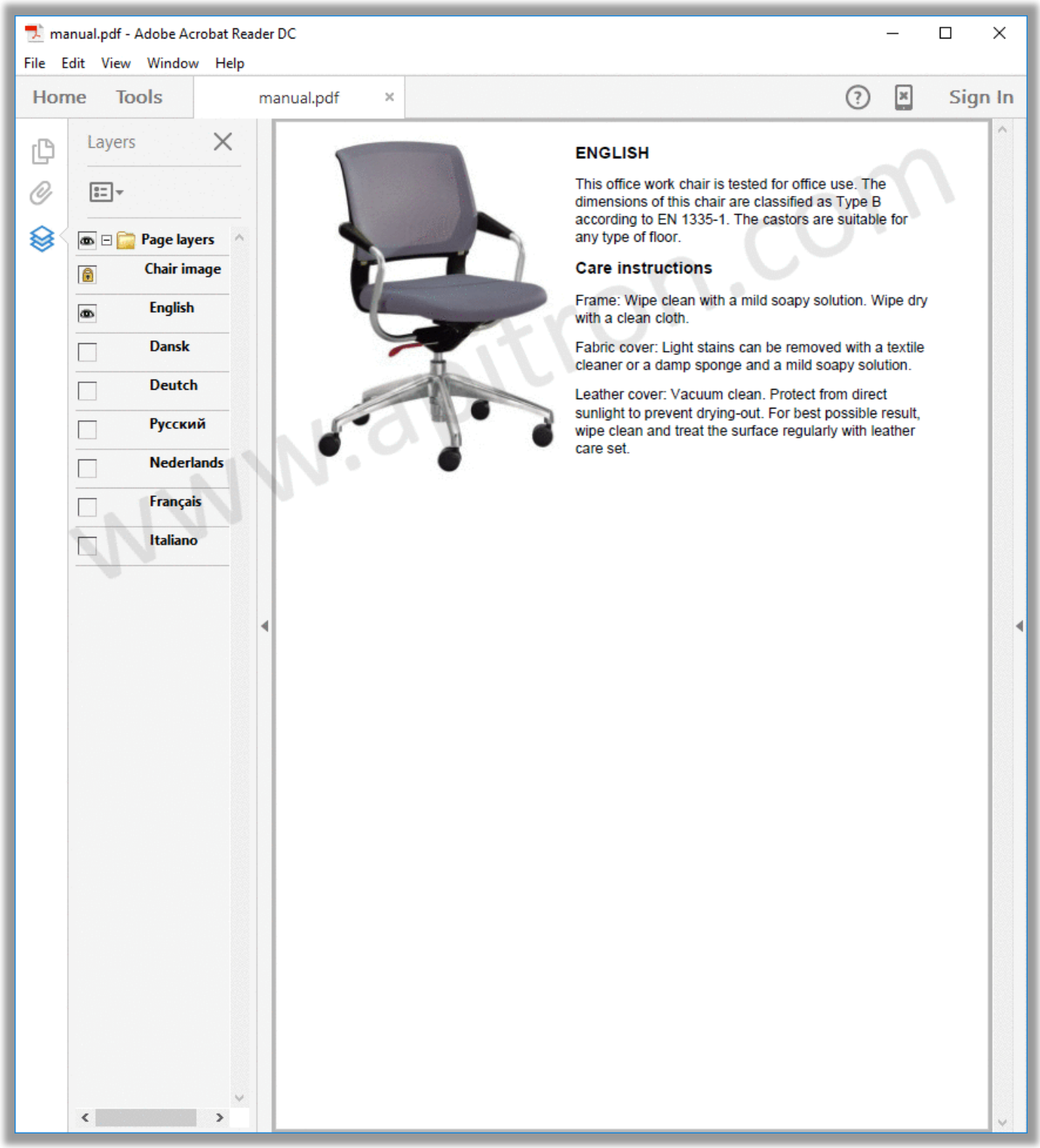
3. Let the document know about the created configuration and layers used

4. Marked all layers with corresponding registered layer ids

5. Saved the PDF document

The complete code sample can be downloaded from our github repo (link).

Resulting PDF document looks as follows:




**Pic. 1 Multilanguage PDF document with layers**

You see the locked layer containing chair image and language layers available for viewing. These language layers work as radio buttons group, when one is turned on others go off.

## Summary

The Apitron PDF Kit for .NET is a powerful tool for creation and manipulation of PDF and PDF/A documents. It's cross-platform and can be used to create .NET, Mono and Xamarin applications for Windows, iOS, Android and other operation systems. You can read more about the library on the product page. Contact us if you have any questions and we'll be glad to assist you.