# Extracting pages from a PDF document and saving them as separate image files, C++/CX edition with co_await

devblogs.microsoft.com/oldnewthing/20170630-00

June 30, 2017

Raymond Chen

Last time, we converted the C++/CX version of the PDF Document sample program so that it saved the pages to disk as image files, using PPL tasks. Today we'll do it again, but using the proposed `co_await` syntax.

```
#include <pplawait.h>

void Scenario1_Render::ViewPage()
{
    ViewPageAsync();
}

task<void> Scenario1_Render::ViewPageAsync()
{
    rootPage->NotifyUser("", NotifyType::StatusMessage);

    // If the text is not a valid number, then wcstoul returns 0,
    // which is an invalid page number, so we don't need to
    // special-case that possibility.
    unsigned long pageNumber = wcstoul(PageNumberBox->Text->Data(),
                                       nullptr, 10);

    if ((pageNumber < 1) || (pageNumber > pdfDocument->PageCount))
    {
        rootPage->NotifyUser("Invalid page number.",
                             NotifyType::ErrorMessage);
        return;
    }

    Output->Source = nullptr;
    ProgressControl->Visibility =
        Windows::UI::Xaml::Visibility::Visible;

    // Convert from 1-based page number to 0-based page index.
    unsigned long pageIndex = pageNumber - 1;

    auto picker = ref new FileSavePicker();
    picker->FileTypeChoices->Insert("PNG image",
        ref new Platform::Collections::Vector<String^>({ ".png" }));
    auto outfile = co_await picker->PickSaveFileAsync();
    if (outfile)
    {
        auto page = pdfDocument->GetPage(pageIndex);

        auto transaction =
            co_await outfile->OpenTransactedWriteAsync();
        auto options = ref new PdfPageRenderOptions();
        options->DestinationHeight = (unsigned)(page->Size.Height * 2);
        options->DestinationWidth = (unsigned)(page->Size.Width * 2);
        co_await page->RenderToStreamAsync(transaction->Stream, options);
        delete transaction;
        delete page;
    }
    ProgressControl->Visibility =
        Windows::UI::Xaml::Visibility::Collapsed;
}
```

```
// Plus appropriate changes to the Scenario1_Render.xaml.h
// header file to match the new function, left to the reader.
```

We are using the proposed `co_await` language keyword which turns the function into a state machine, the same way that the `await` keyword does in C#. But instead of using C# `Task<T>` objects, the `co_await` keyword converts the function into whatever you specified in the traits type. In our case, we use the `pplawait.h` header file, which describes how to convert an `IAsyncAction` or `IAsyncOperation<T>` into a `Concurrency::task<T>`.

But wait! (await?) I'm not done yet. We'll pick up the story next time.

**Bonus chatter**: Here's the C++/WinRT version. This is a theoretical exercise because the XAML compiler doesn't support C++/WinRT yet, but I'll put it here for completeness.

```
task<void> Scenario1_Render::ViewPageAsync()
{
    rootPage->NotifyUser("", NotifyType::StatusMessage);

    // If the text is not a valid number, then wcstoul returns 0,
    // which is an invalid page number, so we don't need to
    // special-case that possibility.
    unsigned long pageNumber = wcstoul(PageNumberBox->Text->Data(),
                                       nullptr, 10);

    if ((pageNumber < 1) || (pageNumber > pdfDocument.PageCount()))
    {
        rootPage->NotifyUser("Invalid page number.",
                             NotifyType::ErrorMessage);
        return;
    }

    Output->Source = nullptr;
    ProgressControl->Visibility =
        Windows::UI::Xaml::Visibility::Visible;

    // Convert from 1-based page number to 0-based page index.
    unsigned long pageIndex = pageNumber - 1;

    FileSavePicker picker;
    picker.FileTypeChoices().Insert("PNG image", { ".png" });
    auto outfile = co_await picker.PickSaveFileAsync();
    if (outfile)
    {
        auto page = pdfDocument.GetPage(pageIndex);

        auto transaction =
            co_await outfile.OpenTransactedWriteAsync();
        PdfPageRenderOptions options;
        options.DestinationHeight((unsigned)(page->Size.Height * 2));
        options.DestinationWidth((unsigned)(page->Size.Width * 2));
        co_await page.RenderToStreamAsync(transaction.Stream(), options);
        transaction.Close();
        page.Close();
    }
    ProgressControl->Visibility =
        Windows::UI::Xaml::Visibility::Collapsed;
}
```

The changes are as follows:

- Method calls use dot notation rather than arrow notation.
- Fetching properties is done by calling a method (via dot) with the name of the property, and no parameters.
- Setting properties is done by calling a method (via dot) and passing the desired new value as the parameter.

- Closing an object is done by calling the `Close()` method, as opposed to C++/CX which overloaded the `delete` operator.
- Constructing an object is done by merely declaring it.
- Wrapping a pointer is done by constructing an object around it. (Though it is common to use assignment-style construction.)
- C++/WinRT lets you pass an initializer list when an aggregate is expected.
- We didn't use it here (thanks to the magic of `co_await`) but the parameter passed to task continuations is an `adapter` if the underlying class does not have a default constructor.

Raymond Chen

**Follow**