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

devblogs.microsoft.com/oldnewthing/20170629-00

June 29, 2017

Raymond Chen

At the start of this series, we converted the C# version of the PDF Document sample program so that it saved the pages to disk as image files. Today we'll port those changes to C++/CX with tasks.

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

    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" }));
    create_task(picker->PickSaveFileAsync())
        .then([this, pageIndex](StorageFile^ outfile)
    {
        if (outfile)
        {
            auto page = pdfDocument->GetPage(pageIndex);

            return create_task(outfile->OpenTransactedWriteAsync())
                .then([this, page](StorageStreamTransaction^ transaction)
            {
                auto options = ref new PdfPageRenderOptions();
                options->DestinationHeight = (unsigned)(page->Size.Height * 2);
                options->DestinationWidth = (unsigned)(page->Size.Width * 2);
                return create_task(page->RenderToStreamAsync(transaction->Stream,
options))
                    .then([this, page, transaction]()
                {
                    delete transaction;
                    delete page;
                });
            });
        }
        else
        {
            return task_from_result();
        }
    }).then([this]()
    {
```

```
        ProgressControl->Visibility =
            Windows::UI::Xaml::Visibility::Collapsed;
    });
}
```

This code is structured the same as the JavaScript Promise-based version. But unlike JavaScript, we cannot capture local variables by reference because the stack will unwind before the continuation runs. We have to capture them by value.

As with JavaScript, C++/CX lacks a `using` keyword, so we must explicitly close the closable objects when we are done with them. In C++/CX, this projected as the `delete` keyword.

Since C++/CX is a strongly-typed language, our tasks are a bit more annoying because we have to make sure all code paths return the same type, because a function can have only one return type. This means adding a `return task_from_result()` to the `else` branch so that all code paths return a `task<void>`.

You might be able to guess what the next step in our adventure is going to be. We'll take it up next time.

Bonus chatter: Here's the C++/WinRT version. This is a bit of a spurious exercise because the XAML compiler doesn't support C++/WinRT yet, which means that you'll see a mix of C++/CX code (when interacting with XAML) and C++/WinRT code.

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

    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" });
    create_task(picker.PickSaveFileAsync())
        .then([this, pageIndex](adapter outfile)
    {
        if (outfile)
        {
            auto page = pdfDocument.GetPage(pageIndex);

            return create_task(outfile.OpenTransactedWriteAsync())
                .then([this, page](adapter transaction)
            {
                PdfPageRenderOptions options;
                options.DestinationHeight((unsigned)(page->Size.Height * 2));
                options.DestinationWidth((unsigned)(page->Size.Width * 2));
                create_task(page.RenderToStreamAsync(transaction.Stream(), options))
                    .then([this, page, transaction]()
                {
                    transaction.Close();
                    page.Close();
                });
            });
        }
        else
        {
            return task_from_result();
        }
    }).then([this]()
    {
        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.
- The parameter to task continuations is an `adapter`.

Raymond Chen

Follow