One-Click Model Reports:
Connect Revit to the InDesign API

Oliver Green & Aaron Perry
Allford Hall Monaghan Morris
Thanks for Joining Us
Session Overview

Introduction
About AHMM, Digital Design Group, Speakers

Model Reviews
Overview of our Model Review Process, QA at AHMM

InDesign’s API
Introduction to InDesign API, Interprocess Communications

Automated Model Reviews
Assembling Everything Into a Finished Product
About AHMM

Allford Hall Monaghan Morris

AHMM is a large architecture firm (480+) with offices in London, Bristol and Oklahoma

Works across all sectors and sizes

Stirling Prize winners 2015
AJ100 Practice of the Year 2018
Building Magazine Practice of the Year 2018
Digital Design Group

Office-Wide Support

Within AHMM, the Digital Design Group offers full time project assistance in all areas of Digital Design strategy, application support, content and computation.

The DDG develops and tests strategies for model maintenance, best-practice workflows, training, standards and QA.

AHMM has extensive experience developing custom tools in-house to assist architectural teams.
Practice BIM manager and lead of AHMM’s digital design group since 2015

Responsible for digital design across the entire practice, its multiple offices, stretching all live and future projects. This involves mitigating risk, engaging client / contractor, managing infrastructure and software, driving change and inspiring staff to embrace digital design authoring, review and visualisation technology.
About the Speaker

Oliver Green

Designs and develops custom tools to assist AHMM’s architects. This involves anything from building design tools, model analysis, data management to full process automation.

Formerly worked as an architect
Before that, a video games designer

Self-taught developer, using Python, C# and Dynamo in daily work
The Value of Model Metrics?
Digital Projects Dashboard

We could never open all models and manually review them on a weekly basis. It is very difficult to know when something is going wrong on a project, updates from some teams is very light.

AHMM run a monitoring tool that records all usage of Revit. When key activities occur on a project, we are notified. Healthy/unhealthy projects and comparison, Training/Support requirements and cross projects business insights.
KnowledgeSmart Assessments

I don’t know what you don’t know

Custom AHMM assessments to understand where knowledge gaps exist for us to then develop a training roadmap/plan
AHMM run regular training sessions almost every day
40+ Internally-developed training courses - from 1 hour workshops to full day Revit training sessions
Our experienced team delivers training in a standardised way, contextualised within the way we work at AHMM
A detailed 40-page InDesign document we prepare for each project per stage
Some parts automated export from Revit, other parts human-authored commentary

Not just a data export; a way of measuring Revit skills & imparting applied knowledge
Not Just a Technology Talk
Development at AHMM

We develop our own tools in-house to allow for custom UI, high-performance functionality that meets our needs. A library of pre-built and audited content, Dynamo & Python scripts, our C# Revit Ribbon (+ WPF front-end)

Whatever we can do to "let architects be architects"
We use InDesign at lot at AHMM - it's powerful, flexible and creates beautiful reports that are easy to edit. Automating InDesign has been on our wish list for a long time.

Not just for architects – potentially helpful for all parts of AEC.
Last Year's AU London Presentation was also about creating reports
A different take - using Dynamo. This is more manual, but more accessible technologically

Also a talk about data visualisation and displaying accurate information in a succinct way
Semi-Automated Model Review

Initial forays streamlining InDesign workflows – best practice templates, styles and using Text Variables

We created Dynamo definitions to generate images highlighting aspects of the model

This saved lots of time... but still involved lots of manual copy-pasting data from an Excel export!
A year ago we demonstrated generating 536 Room Data Sheets in 1 minute
InDesign’s database publishing tool – combines structured data with a page template

We wanted the ability to generate the kinds of InDesign documents a user would normally create – from scratch
Initial Explorations
Access To The InDesign Product Family SDK (Software Developer Kit)

The InDesign SDKs are for C++ programmers and scripters who want to learn how to write plug-ins and scripts for Adobe InDesign, InCopy, and InDesign Server. They are designed to give an introduction to plug-in and script development, show how to create some simple plug-ins and scripts, and teach the architecture behind the InDesign product family.

InDesign SDK

Initial research showed yes - there's an InDesign API, and an SDK with documentation
Downloaded SDK & read through docs. InDesign Server or short, simple scripts

Lots and lots of JavaScript mentions, some TypeScript and AppleScript
Documentation online and in downloadable SDK
Some helpful, but incomplete 'mind maps' online. Not always intuitive

Wanted a fully .NET-based solution if possible; easier to integrate with existing Revit / WPF tech we use

InDesign API Documentation

Documentation online and in downloadable SDK
Some helpful, but incomplete 'mind maps' online. Not always intuitive

Wanted a fully .NET-based solution if possible; easier to integrate with existing Revit / WPF tech we use
InDesign API Documentation

Documentation online and in downloadable SDK
Some helpful, but incomplete 'mind maps' online. Not always intuitive

Wanted a fully .NET-based solution if possible; easier to integrate with existing Revit / WPF tech I use
Early 2019 I started reading about InDesign’s API in depth - I found just a few examples using C#. Example script would open a new InDesign document and create five blank pages.

Being able to see a test script implied it should be possible to create something in C# that talks to InDesign.
Thank You user3791372!

This was the initial script I read (on Stackoverflow) that I based our proof of concept on.

Sometimes, this is all you need to set off developing something.
Simple Proof of Concept Workflow

I put together a proposal for a 'minimum test case' to see if we could create a working proof of concept. Our end goal was to read a Revit model's information into a ready-made InDesign template.

POC was to see if I could click a button in Revit that would open InDesign and a new document and edit it.
Ribbon Buttons in Revit

We already had some experience with building our own ribbon, so I can give you an outline of what we did

There are two straightforward ways to launch your own code from the Revit ribbon: these are referred to as *ExternalCommands* and *ExternalApplications*
Intro to External Commands
ExternalCommands & ExternalApplications

Both are a way of bringing external code into Revit
We used Microsoft Visual Studio as our "Integrated Development Environment" (i.e. where we write our code)

Written in C# using the Revit API's Classes
Code gets compiled into a .dll file and placed somewhere Revit can see it (e.g. AppData)

On Revit application startup, it loads in these resources

With both approaches the .dll file will also contain the code that fires when you launch your command or click on each custom-made button

Within this code, you can access Revit's API to make adjustments to the model just like in a Macro
Visual Studio Overview

Visual Studio is the program everyone uses to compile their Revit addins.

It’s a code-editor, a UI designer, database administration tool, debugger, and more all in a single program.

It’s where we compile all our scripts together into a single DLL, which Revit can then run.
**ExternalCommands**

If your tool is an ExternalCommand, Revit will read the .dll file and load it in as a button.

This button will appear in Revit under the Add-ins tab in the External Tools drop-down menu.

The ExternalCommand approach is a bit like firing a macro - it's a one-off command you're launching.
If your tool is an ExternalApplication it probably lives in its own custom Revit ribbon tab.

Revit reads your .dll upon opening and creates your ribbon tabs, ribbon panels and buttons.

The Revit ribbon buttons essentially fire off their own ExternalCommands, but there are some differences.

The ribbon is always accessible from within Revit, meaning you can retain data between tool runs.
Further Reading

If you'd like to learn how to build your own custom Revit ribbon, or ExternalCommand button I would recommend archi-lab's blog

http://archi-lab.net

Clear step-by-step instructions on how to build ExternalCommands and ExternalApplications from scratch

If this helps you, consider supporting Konrad!
We want our code to fire when we press a button in Revit, so we're using an ExternalApplication to create our custom AHMM ribbon and place buttons in there. These buttons will fire off our custom code.

Next, we want to learn how to open an instance of the InDesign application from within our custom code.
Interprocess Communications
Revit and the .NET Framework

Revit is a Windows application and it runs on top of the .NET Framework. Microsoft's .NET Framework is the lower-level architecture that unifies software running on a Windows machine, provides important class libraries and resources. This lets us have conversations with other applications, access external servers, databases, run tests, etc.

We have access to all different parts of Windows to play with if we want - can access speakers, webcams, files.
Windows provides many approaches for sending data between applications - 'interprocess communication' (or IPC) is one of them. Refer to MSDN Website for developers who want to develop on top of Microsoft systems.

"Interprocess Communications" just means getting pieces of software to talk with one another.

I found this page, which outlines multiple IPC technologies: IPC, Sockets, Data Copy, Pipes...
COM (Component Object Model)

COM is old (1993) technology but is still supported by .NET using the System.Runtime.InteropServices namespace. For instance, Dynamo's Bumblebee package uses COM to open & edit Excel documents.

Spoiler Alert: As we will see later, this is the approach I would have to use.
Visual Studio Setup
Back to Visual Studio

Our code in Visual Studio would need to bring everything together.

It’s where we have the code for creating our Revit ribbon, buttons and model review functionality (e.g. counting the number of warnings).

It’s also where we need to have the code which launches and manipulates InDesign.
Add Reference to InDesign API

The online script had lots of reference to InDesign’s native classes (such as Page).

In order to resolve these references, I needed to add a reference to InDesign’s API.

I was entirely sure how to do this so, some Googling later, I found it’s called ResourcesForVisualStudio.tbl, which is a COM type library file.

That’s how I knew I was going to use the COM approach to Interprocess Communications.
Add Reference to InDesign API

If you have InDesign installed, you can find it at: C:\ProgramData\Adobe\InDesign\Version 11.0\en_GB\Scripting Support\11.0\Resources for Visual Basic.tlb

(Exact path address depends on Version but will be almost exactly the same)

Right-Click on your Solution > Add Reference > Browse and select .tlb file

Add 'using InDesign;' to your using directives at the top of your C# file, references should resolve
using System;
using InDesign;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI;
using Autodesk.Revit.Attributes;

[Transaction(TransactionMode.Manual)]
public class InDesignSampleAU2019 : IExternalCommand
{
    public Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
    {
        try
        {
            Type type = Type.GetTypeFromProgID("InDesign.Application");
            Application inDesignApp = (Application)Activator.CreateInstance(type);
            //Create 5 new blank pages
            for (var i = 1; i < 5; i++)
                inDesignDocument.Pages.Add(idLocationOptions.IdAtBeginning);
            return Result.Succeeded;
        }
        catch (Exception ex)
        {
            message = ex.Message;
            return Result.Failed;
        }
    }
}
Error - cannot be ignored

Unable to cast COM object of type 'System._ComObject' to interface type 'Indesign.Application'. This operation failed because the QueryInterface call on the COM component for the interface with IID '{A9E4C662-6C0E-11D1-901D-0060803C02AE}' failed due to the following error: No such interface supported

Show  More Info  Expand >>

OK  Cancel
Note: COM error messages are almost worthless!
Fixing COM Error

Find your InDesign .tlb file and delete it
Then run InDesign as Administrator
This will recreate the .tlb file, which should no longer throw an error

The test script worked, opening InDesign and creating 5 new blank pages!

Windows 10 with InDesign CC

ExternalApplication is compiled to x64 architecture / solution platform.
Back to Proof of Concept

We create an instance of the InDesign application using COM’s Activator.CreateInstance() method. This gives us a handle on an opened session of InDesign. From here, we need to create a Document to edit. I create a blank document as a proof of concept. Once we have a Document we can do nearly anything we want to. In our example code we created five pages.
Preparing The Model Report Process
What Can We Automate?

Proof of Concept worked so I printed out our Model Report and marked up in red what I thought we could automate.

It was a lot, maybe 80% of the information in the report.

Around 40 pages - a structured & formatted report, made up of a mix of tables, images, critical commentary.
Having marked up our report I could differentiate different key tasks our tool would need to complete. We already demonstrated we had working access to the InDesign API from within Revit.

Next, I had to work out if, and how, these could be automated in code.
1. Pop-Up Dialog for Input Info

The process would seek to confirm a few key values from the user at the start of each run.

This is a WPF pop-up dialog window to let users enter the project name and number.

Where possible, they are pre-populated from the Project Information parameters, but ultimately having a user confirm they're useful and valid values.
2. Create New File from Template

I would need to begin the process by opening an instance of the InDesign application using COM. I found examples of how to do this online. I needed to specify InDesign.Application as the type, as Revit has its own native Application Type.

```csharp
Type type = Type.GetTypeFromProgID("InDesign.Application");
InDesign.Application indesignApplication = (InDesign.Application)Activator.CreateInstance(type);
```

Activator.CreateInstance is used to open the application, which I cast back to the InDesign.Application type, allowing me access to the application's members (methods, events, properties, etc).

Finally, I was able to open an copy of an existing template file and create a handle for it:

```csharp
```
2. Create New File from Template

```csharp
[Transaction(TransactionMode.Manual)]

public class InDesignSampleAU2019 : IExternalCommand
{
    public Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
    {
        // Setting the context Revit Application & Document
        UIApplication revitUIApp = commandData.Application;
        UIDocument revitUIDoc = revitUIApp.ActiveUIDocument;

        try
        {
            Type type = Type.GetTypeFromProgID("InDesign.Application");
            Application inDesignApp = (Application)Activator.CreateInstance(type);
            const string TemplatePath = @"\path\to\indesign\template\file.indt";

            // Create 5 new blank pages
            for (var i = 0; i < 5; i++)
            {
                inDesignDocument.Pages.Add(idLocationOptions.idAtBeginning);
            }

            return Result.Succeeded;
        }
        catch (Exception ex)
        {
            return Result.Failed(message);
        }
    }
```
3. Find & Replace Text

With the template copy open, we can get to work!

The first edit I'd want to make would be to target specific placeholder words in our Model Review template and replace them with meaningful values.

The template we designed for automation purposes was created full of placeholder words for certain values, such as "There are NoWarnings in the model". I wanted to build a method to switch out these placeholder values.
3. Find & Replace Text

The InDesign API has find & replace functionality using GREP - Global Regular Expressions Print. GREP operations are very quick & efficient, using regular expressions. I created a method called FindAndReplaceGREP():

```csharp
void FindAndReplaceGREP(string stringToFind, string stringToReplace) {
    inDesignApplication.ChangeGrepPreferences //to initially set up parsing rules. These will not change
    inDesignApplication.FindGrepPreferences.findWhat = stringToFind;
    var findGrep = inDesignDocument.FindGrep();
    inDesignApplication.ChangeGrepPreferences.changeTo = stringToReplace;
    inDesignDocument.ChangeGrep();
}
```

With that method set up, this is all that's required to find and replace text within the target document.
3. Find & Replace Text

I created a 'GREP Dictionary', which is just a dictionary to associate certain specific words to their replacement values.

Then I could iterate through all entries in this dictionary to set the values I wanted, while keeping all data together.

```csharp
Dictionary<string, string> grepDictionary = new Dictionary<string, string>()
{
    {"ProjectName", modelReviewInfo.TextBoxProjectName.Text},
    {"NoImportInstances", allImportInstances.Count.ToString()},
    {"NoCADLinks", allCADlinkTypes.Count.ToString()},
    {"NoRevITLinks", allRevITlinkIds.Count.ToString()},
    {"NoOOGLinks", numberOfOOGLinks.ToString()},
    {"NoImportedCAD", numberOfImportedCad.ToString()},
    {"NoUnplacedViews", numberOfUnplacedViews.ToString()},
    {"NoViewTemplates", allViewTemplates.Count.ToString()},
    {"NoModelGroups", allModelGroups.Count.ToString()},
    {"NoDetailGroups", allDetailGroups.Count.ToString()},
    {"NoGenericModelTypes", allGenericModelTypes.Count.ToString()},
    {"NoGenericModelInstances", allGenericModelElements.Count.ToString()},
    {"NoWorksets", allWorksets.Count.ToString()},
    {"FileSizeMB", fileSizeMB.ToString()},
    {"FileSizeRAM", Math.Round(fileSizeMB * (20.0/1000), 0).ToString()},
    {"NoDesignOptions", allDesignOptions.Count.ToString()},
    {"NoDesignOptionSets", allDesignOptionSetIds.Count.ToString()};

// Iterate through the grepDictionary to replace keywords
foreach (KeyValuePair<string, string> item in grepDictionary)
{
    FindAndReplaceGREP(item.Key, item.Value);
}
```
4. Update Text Variables

I wanted to target the document’s Text Variables, which are defined once and implemented in many places across the document.

Accessed via `Document.TextVariables` which returns a list of TextVariables we can loop through.

In a similar manner to my ‘GREP dictionary’, I created a ‘Text Variables Dictionary’ to associate text variable names to their values.

The values can be set using `TextVariable.VariableOptions.Contents`
5. Target Tables & Input Data

Tables were difficult to access because the API is a bit odd in places.

Tables cannot be accessed conveniently using Document.Tables, or even Page.Tables.

They are accessed as Document > Pages > TextFrames. Tables are a kind of TextFrame, but so are normal text boxes in InDesign.

There are tests to determine which is which, but this all seemed a bit backwards as I wanted the ability to target specific tables.

1.0 Detailed Findings & Model Size

1.6 External CAD Files

There are NoCADLinks unique CAD files in the model, which have been placed a total of NoimportedInstances times. Of these CAD files, NoDWGLinks are in DWG format and NoDXFLinks are DXF format (DWG files can misbehave when brought into Revit).

NoimportedCAD imported CAD file(s) were found in the model; remove these and add them as links instead.

Please review this list to ensure that the files placed are still needed or remove them otherwise.

If any duplicates exist, you should consider converting these into detail items.

Please Note: This list is not exhaustive.

Comment: Are there duplicated instances of the same link? This could indicate users are linking in information per view; we advise against this as it can lead to lots of duplicates (when these views are duplicated). The files should always be linked and we advise against DGNs due to poor performance. Are link names sensible?

<table>
<thead>
<tr>
<th>Name</th>
<th>Level</th>
<th>Linked?</th>
<th>By View?</th>
<th>Element ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Target Tables & Input Data

In InDesign all elements have an arbitrary ID number, like 6 or 52.

The InDesign API lets us select items by their ID.

But it would be a bad idea to hard-code the IDs of each table in the template; this could easily change as I make new copies of or update the model review template.

I wanted an approach to guarantee I was selecting the right table every time, regardless of its ID.
5. Target Tables & Input Data

However, InDesign lets you apply something called 'Script Labels' (readable strings) to elements. We can't search for elements by their Script Label. Therefore, I created a dictionary to map each table (with its script label) to its internal ID when the `ExternalCommand` first runs.

This would let us create a method to search for a table by its label and have it returned to us (courtesy of its ID).
5. Target Tables & Input Data

We can't access a document's Tables directly.

Since tables are a subclass of TextFrame, we have to iterate through all of these, querying whether each TextFrame has a Script Label applied to it. In order to access TextFrames, we need to iterate through the pages in the document.

We can do this and populate our Script Label / ID reference dictionary. We need to create this dictionary only once at the start of our script run.

We can then use it for reference to look up Tables.
I created a method called `FindTable`, which simply takes the ScriptLabel name as its key, and which returns the Table I’m after.

We now have the ability to target specific tables by their name:

```csharp
Table linkedRevitFilesTable = FindTable("LinkedRevitFilesTable");
```

The table’s contents are accessed via its Contents property, which needs to be passed an array of strings.
6. Normal .NET Operations

There were some key values I wanted the report to be able to display which I knew could be accessed using the .NET class libraries

I read the current date and time and formatted them using `DateTime.Now.ToString("yyyyMMdd");`

I set the Text Variable for the Report Author's name to read their login name, e.g. OGreen, using `System.Environment.UserName`

These were used to set Text Variables or for Find & Replace operations
6. Normal .NET Operations

I used .NET's threading libraries and WPF to display a live-updating progress bar

This updates in its own thread to get around Revit's single-threadedness

Progress percentage was somewhat arbitrary; I decided how many steps there were and wrote code to update the progress bar after each step

How does one accurately reflect progress? Seems to be a classic programming debate...
6. Normal .NET Operations

Finally, I used some Precompiler Directives in my code to adjust certain operations for different versions of Revit’s API.

For instance, I couldn’t access the number of model warnings via the Revit 2017 API to write this value into our document.
7. Launch UI Menu Commands

As the last step in our report generation, I needed the ability to update the Table of Contents. This is normally accessed in the UI via the Layout Menu > Update Table of Contents.

Menu Commands in InDesign each have a specific 'Command ID'.

I found a free Javascript script online, written by a Lancaster-based typesetter called Peter Kahrel.
7. Launch UI Menu Commands

The script creates a mini menu of all CommandIds in InDesign

I could then sort these menu commands or search using keywords to find the ID of the menu action I needed (it was 71442)

I needed to find the TextFrame containing the Table of Contents using its Script Label, and select it in code using Application.Selection
7. Launch UI Menu Commands

Then I could use the MenuAction and
indesignApplication.MenuActions.ItemByID(71442);

To select the Action, followed by
menuAction.Invoke() to invoke the command

This Updated the Table of Contents
Live Demonstration
Conclusion

Our Model Review tool did everything we wanted it to: copying a template, filling tables & key values

About 80% of the report's content was auto-generated, saving many hours per model review

It was naturally limited in its scope: "Let's not spend months on this, but what could we reasonably achieve?"

It is certainly possible to take this further
Further Possibilities

Use Revit API to create isolated images of warning elements / worksets, save an image and dynamically update template’s image placeholders.

Intelligent commenting based on pre-existing knowledge of filesize, RIBA stage and number of elements in the model.

Automatic formatting of paragraphs according to a condition.

Using WPF data visualisation libraries to create charts & graphs (e.g. LiveCharts).
Now It's Your Turn

Have any reports you want to automate?

We have uploaded skeleton code samples to AHMM's Github repository.

We have just scratched the surface of what Revit and InDesign's API can do together.
Resources

archi-lab blog for very detailed posts on creating ExternalCommands and ExternalApplications

The Building Coder blog for Revit API reference

InDesign's SDK for API docs and examples

YouTube: Jamie King’s channel for helpful explanations of C# concepts

Our Github - to get started
Thank You for Listening