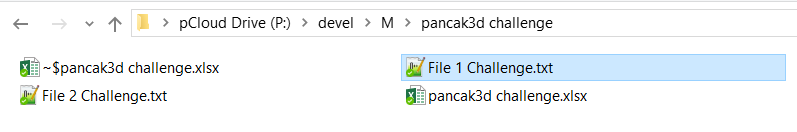
I’ll start off with my standard template which can be found at <https://github.com/tirlibibi17/excel-pq/tree/master/PQ%20Template>

I’ve renamed it to *pancak3d challenge.xlsx* and it’s in the same directory as *File 1 Challenge.txt* and *File 2 Challenge.txt*.



Let’s start with File 1. It’s located at P:\devel\M\pancak3d challenge\File 1 Challenge.txt

Let’s create a configuration key for file1.

Now we’ll just open the file as text with a hard-coded path. We’ll change it later.

Let’s start by making the filename configurable.

Let the parsing begin. What we want is to assign each group of rows a unique index that we can group on.

Now that we’ve identified the group separators, we can just fill down and we have a unique index that we can group on.

As you can see, most groups contain inmate data, but some are garbage. Let’s not worry about them right now and focus on the ones with inmate information.

To do that, let’s duplicate this query and extract a sample table.

Next, all we need to do is parse this bunch of rows. I’ll cover that in the next part.

PART 2

After reviewing the code, I realized that I’d messed up in the previous step because I still need the index columns. So let’s go back and fix that and calculate a local index I’ll need to grab the address information. This local index is going to come in handy when we want to get the different parts of the address.

OK. So now, how do we parse Column1? I have a hunch it’s fixed width text. Let’s see if we can visually confirm that. Isn’t it beautiful how the columns align nicely?

Let’s copy the text to Notepad++ and figure out the starting positions of the columns we’re interested in.

I’m going to cheat now because I’ve already identified the parts I’m interested in.

Now let’s change that query step to remove the list of columns and to add the positions we want to split on. Oh, and we also want to get rid of the annoying automatically added Changed Type step.

Now this guy isn’t a great example because his second name is misspelled and, more importantly, he only has one charge. Let’s go and choose another one.

Better. We’ll check later if this works for 3 or more.

Let’s start by getting the Name and address parts.

This is where the LocalIndex comes in handy (or not LOL)

What we’ve done here is extract some specific rows we want, and then reset the context to how it was before.

Now let’s look at the charges.

What I’ve done here (not without some trouble), is to trim the charges column to remove trailing spaces, and filter out the blank lines, and then only kept the Custom and Arrest charge columns. The Custom column is only interesting because it’s the same for all rows, so it’s something I can group on.

OK, but I’m only interested in the “Arrest charge” column. No problem, I’ll change the code in the step. Notice how the Custom column has disappeared and the all column now contains Lists (or rather one row with a list).

So now I have all the arrest charges combined in one neat column. Let’s add the name and address information.

Now for the parts of the address.

For file 1, we have no Agency information, but we still need to have the column present.

OK, great. We’ve parsed one record. How do we parse all of them? I’ll cover that in part 3.

PART 3

Before we move on, you probably noticed that I created a local index column that I ended up not using because I can just use the row numbers. Let’s go and clean that up.

Now for the really cool part, let’s create a function that will apply all these steps to any table we throw at it.

For that, I’m going to need to create a parameter for the function so I can pass it a table. Let’s call that parameter tbl1 (because it’s for File1) and make it optional so that the Parameter manager lets me create it with no value.

Now let’s split the query I just created (Parse1Query) in 2:

* The sample table
* The logic to parse that table

So now we have 2 queries. Let’s assign Parse1SampleTable to parameter tbl. Unfortunately, there’s no point and click way of doing that, but it’s not very complicated.

Now, I’ll just go and change the source of Parse1Query to point to tbl1 instead of Parse1SampleTable, so that the editor knows that tbl1 is a parameter to pass to the function it’s going to automatically generate.

What have we done here? We’ve created a function from a query, which will continue to be linked to that query. You’ll see that a bit later when I realize my function has a problem and I need to fix the query.

Let’s use the function now and see what happens. Pretty good. Let’s expand into separate columns and see how it looks. Not too shabby.

I’ll pause now and be back in Part 4 for the final cleanup.

PART 4

Looking at the table, there are a bunch of null values that have me wondering if something is wrong with my function. Let’s take a look at the table in “all” they were generated from.

As you can see, Row 1 is a header, and so is Row 11. So those null values are normal. In fact, they’ll be useful when I’ve fixed the function to filter out only the real inmate records.

But obviously, row 12 (index 70) should parse and it doesn’t. Let’s go see why.

Interesting. Now rows. Let’s go step by step in the query and try to figure out why.

Aha! Mr. Corona has no arrest charges, so what I’m going to do is add a row with a null value before filtering out the blanks.

OK, so I get an error because I haven’t specified all the fields. I’m lazy and I know that I’ll only be keeping the arrest charge and custom column down the road, so let’s remove them now.

So now I have something that works for a guy with no arrest charges. Let’s go back and see what happens to a guy who has arrest charges.

Just as I thought, I now have 2 rows. No matter. I know I only want the bottom row. If there’s only one row, I’ll end up with the first row, which is fine.

Looks like all the nulls are headers. Let’s just check the last one.

OK, that’s the file footer, so I can safely remove it.