Population of Large System Tables from Excel

A solution to make large system table management easier

Chris Moore

Overview

System tables are stored as XML files in PolicyCenter, the XML files are generally updated via the Product Designer, this updates the XML and on restart of the server or forced via Product Designer the data in the system table is created, updated or removed. While this process works well for simple system tables, even 100 rows can prove to be tedious to create.

Making use of Excel to define the information is much simpler to do, the problem is converting the Excel information into XML that will function for system tables. This document will detail the process and code used to convert the excel spreadsheet into a useable XML file. The example used will take the process from a simple to complex variations and how each can be handled.

While the code used is Python this is not a tutorial on how to code in Python. As the system tables are individual the solution has not been made generic, rather will require some code per table to be created, in the code examples are various techniques that can be copied and altered with minimum effort.

Process Part 1

Python was used as the language to write the code as it proves to be simpler to perform the processing than other languages, and has an extensive set of libraries that ease the code needed.

The simplest way to work with Excel information is use the `pandas` library, this is able to read and write information in multiple formats, its `DataFrame` correlates with an Excel spreadsheet.

```
import pandas as pd
```

This is the standard for `pandas` and will be referred to as `pd` withing the rest of the code.

1. The spreadsheet is read into a `DataFrame` `read_excel` is used to perform this action, if you have multiple sheets you will need to add the sheet name to the read. Refer to the `pandas` documentation for more interesting things to do while reading an Excel file.
2. In the sample spreadsheet that I used, there was an `ID` column that should have contained a unique value, as per usual it did not. As the `Id` would be used as `PublicId` and the `Code` in the system table, the first order was to remove then duplicates. I my case I would use the original extract later in the process, this is why the copy was made first.
3. A new column is added to the `DataFrame` that records the length of the text in the Question column, this is used later to avoid processing text information that would cause issues.
4. If the name of the column coming from the Excel spreadsheet does not match the name of the attribute in the system table, these can be renamed, in this example `Question` is renamed to `QuestionText`
5. Additional columns are added to the `DataFrame` for `public-id` and `Code`. Along with the creation of these new columns an array of the `Id` used in the final output is created, this is used later to avoid processing information that would cause errors in PolicyCenter.
6. Finally the XML file is written, the parameters use format the XML in the way it is expected in PolicyCenter for a System Table.

```
import pandas as pd

questions = pd.read_excel('/Users/cmoore/Development/SampleData/Questions
on Question Sets.xlsx')
#
# Create a copy of the Dataframe as the original will be used later to
build the other xml files
# Remove rows with Duplicate ID and rows with duplicate Questions. Add New
Column with the length of the question
#
questionCopy = questions.copy()
questionCopy.drop_duplicates(subset='Id', keep='first', inplace=True)
questionCopy.drop_duplicates(subset='Question', keep='first',
inplace=True)
questionCopy['QuestionLength'] = questionCopy['Question'].map(len)
#
# Rename the Question column to QuestionText as this is what will be
```

```
needed for the xml
#
questionCopy.rename(columns={'Question': 'QuestionText'}, inplace=True)
#
# Remove any questions that are too long and reset the index to the new
Dataframe
#
xml = questionCopy[questionCopy['QuestionLength'] < 200].reset_index()
xml.drop_duplicates(subset='QuestionText', keep='first')
#
# Create the Public-id, Code and ValueType columns
#
xml['public-id'] = xml['Id']
xml['Code'] = xml['public-id']
question_check = xml['Id'].array
xml = xml.assign(ValueType='string')
#
# Generate the xml file
#
xml.to_xml('/Users/cmoore/Development/SampleData/industry_code_questions.x
ml', index=False, root_name='import',
           row_name='IndustryCodeQuestion_Ext', attr_cols=['public-id'],
           elem_cols=['Code', 'QuestionText', 'ValueType'])
```

Process Part 2

In my example I needed to create a second XML for a related system table, to achieve this a new `DataFrame` was built

1. Each of the columns needed were defined as empty arrays
2. The original data from the spreadsheet was used to create the iterator, this allows the `DataFrame` to be processed row by row. Processing row by row is need as some of the information needs to be changed to match what the destination expects. For one of the changes to `WorkflowType` uses a custom function, other functions could be added as needed.
3. The `Id` for the row is checked against the set of know processed `Id`, if there is not a match the next row is used.
4. Data from the row is manipulated as needed and added to the relevant array.
5. A new `DataFrame` is created using the previously populated arrays.
6. The `DataFrame` is converted to XML, in this case the XML is returned as a string and not written as there is some changes that need to happen, these can't be done with the `to_xml` function.
7. The element `IndustryQuestionCode` as it is a foreign key has to be formatted differently. `<IndustryQuestionCode>XX</IndustryQuestionCode>` needs to change to `<IndustryCodeQuestion public-id="XX"/>`. A simple replace on the xml string is enough to achieve this.
8. The now correct XML is written away using the standard file operations.

```
#
# Build arrays for each of the columns to be added to the DateFrame, where
questions have been removed from the
# Industry Question these will be avoided in the Industry Question Set
xml.
#
risk_type = []
code = []
decline_question = []
referral_question = []
industry_code_question = []
effective_date = []
sequence = []
workflow_type = []


def convert_workflow_type(workflow_type: str):
    if workflow_type.startswith('Turnover'):
        return 'Turnover'
    if workflow_type.startswith('Activity'):
        return 'Activity'
    if workflow_type.startswith('Qualifications'):
        return 'QualificationsExperience'

for question_row in questions.itertuples():
    pandas = question_row
    if not pandas.Id in question_check:
```

```
            continue

        risk_type_name = pandas.InsuranceTypeName.replace(' ', '')
        risk_type.append(risk_type_name)
        code.append(f'{risk_type_name}:{pandas.Id}')
        if pandas.Decline == 1:
            decline_question.append('true')
        else:
            decline_question.append('false')

        if pandas.Refer == 1:
            referral_question.append('true')
        else:
            referral_question.append('false')
        industry_code_question.append(pandas.Id)
        effective_date.append('2020-01-01 00:00:00.000')
        sequence.append(pandas.Level)

    workflow_type.append(convert_workflow_type(pandas.WorkflowName.replace('
    ', '')))
    #
    # Create the Dataframe by adding each of the arrays. The arrays will form
    the columns in the Dataframe
    #
    question_set = pd.DataFrame({
        'public-id': code,
        'RiskType': risk_type,
        'Code': code,
        'DeclineQuestion': decline_question,
        'ReferralQuestion': referral_question,
        'EffectiveDate': effective_date,
        'Sequence': sequence,
        'WorkflowType': workflow_type,
        'IndustryCodeQuestion': industry_code_question
    })
    question_set.drop_duplicates(subset='public-id', keep='first',
    inplace=True)
    #
    # Convert the new Dataframe to xml, this does not write the xml to disk as
    we still need to make some changes
    # to alow PolicyCenter to recognise the XML as the correct format for the
    System Table.
    #
    xml_string = question_set.to_xml(index=False,
                                     root_name='import',
                                     row_name='IndustryCodeQuestionSet_Ext',
    attr_cols=['public-id'],
                                     elem_cols=['Code', 'RiskType',
    'DeclineQuestion', 'ReferralQuestion', 'EffectiveDate',
                                                'Sequence', 'WorkflowType',
    'IndustryCodeQuestion'])
    #
    # Rewrite the xml to have the correct format for Foreign Keys, a simple
    replace is used
```

```
#
new_xml = xml_string
new_xml = new_xml.replace('<IndustryCodeQuestion>', '<IndustryCodeQuestion
public-id="')
new_xml = new_xml.replace('</IndustryCodeQuestion>', '"/>')

with
open('/Users/cmoore/Development/SampleData/industry_code_question_sets.xml
', 'w') as f:
    f.write(new_xml)
    f.flush()
```

Results

Below are extracts from the two files generated, the sample I used contained over 50,000 rows of data and takes only a couple of seconds to process. There are many changes that could be made to the code, and I hope that this provides the catalyst to do just that.

industry_code_questions.xml

```
<?xml version='1.0' encoding='utf-8'?>
<import>
  <IndustryCodeQuestion_Ext public-id="10">
    <Code>10</Code>
    <QuestionText>What is the total turnover including fee income in the
past year? If a new start up please use an estimate for the first year of
trading.</QuestionText>
    <ValueType>string</ValueType>
  </IndustryCodeQuestion_Ext>
  <IndustryCodeQuestion_Ext public-id="11">
    <Code>11</Code>
    <QuestionText>Estimate for whole current year?</QuestionText>
    <ValueType>string</ValueType>
  </IndustryCodeQuestion_Ext>
  <IndustryCodeQuestion_Ext public-id="12">
    <Code>12</Code>
    <QuestionText>What is your largest fee from a single client?
</QuestionText>
    <ValueType>string</ValueType>
  </IndustryCodeQuestion_Ext>
  <IndustryCodeQuestion_Ext public-id="13">
    <Code>13</Code>
    <QuestionText>Please provide details of your largest fee including:
who is your client?, what type of work is undertaken?, when was this work
completed?</QuestionText>
    <ValueType>string</ValueType>
  </IndustryCodeQuestion_Ext>
  <IndustryCodeQuestion_Ext public-id="14">
    <Code>14</Code>
    <QuestionText>Is your company domiciled in the United Kingdom?
</QuestionText>
    <ValueType>string</ValueType>
  </IndustryCodeQuestion_Ext>
```

industry_code_question_sets.xml

```
<?xml version='1.0' encoding='utf-8'?>
<import>
  <IndustryCodeQuestionSet_Ext public-id="ProfessionalIndemnity:10">
    <Code>ProfessionalIndemnity:10</Code>
    <RiskType>ProfessionalIndemnity</RiskType>
```

```
    <DeclineQuestion>false</DeclineQuestion>
    <ReferralQuestion>false</ReferralQuestion>
    <EffectiveDate>2020-01-01 00:00:00.000</EffectiveDate>
    <Sequence>1</Sequence>
    <WorkflowType>Turnover</WorkflowType>
    <IndustryCodeQuestion public-id="10"/>
  </IndustryCodeQuestionSet_Ext>
  <IndustryCodeQuestionSet_Ext public-id="ProfessionalIndemnity:11">
    <Code>ProfessionalIndemnity:11</Code>
    <RiskType>ProfessionalIndemnity</RiskType>
    <DeclineQuestion>false</DeclineQuestion>
    <ReferralQuestion>false</ReferralQuestion>
    <EffectiveDate>2020-01-01 00:00:00.000</EffectiveDate>
    <Sequence>1</Sequence>
    <WorkflowType>Turnover</WorkflowType>
    <IndustryCodeQuestion public-id="11"/>
  </IndustryCodeQuestionSet_Ext>
  <IndustryCodeQuestionSet_Ext public-id="ProfessionalIndemnity:12">
    <Code>ProfessionalIndemnity:12</Code>
    <RiskType>ProfessionalIndemnity</RiskType>
    <DeclineQuestion>false</DeclineQuestion>
    <ReferralQuestion>true</ReferralQuestion>
    <EffectiveDate>2020-01-01 00:00:00.000</EffectiveDate>
    <Sequence>1</Sequence>
    <WorkflowType>Turnover</WorkflowType>
    <IndustryCodeQuestion public-id="12"/>
  </IndustryCodeQuestionSet_Ext>
```