**pyduq Grammar**

# **Overview**

This document describes the syntax of the meta-data grammar for pyduq.

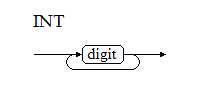
# **Grammar**

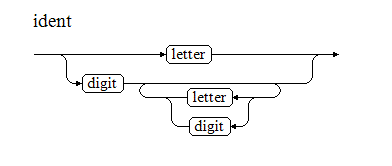
This section defines and describes the supported grammar in pyduq. The current implementation of the grammar is in a python dictionary of dictionaries. In the future, the grammar will support other instantiations including XML and JSON.

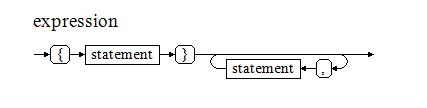
The elements can appear in any order following the attribute name.

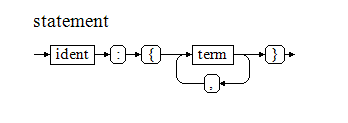
## **EBNF Diagrams**

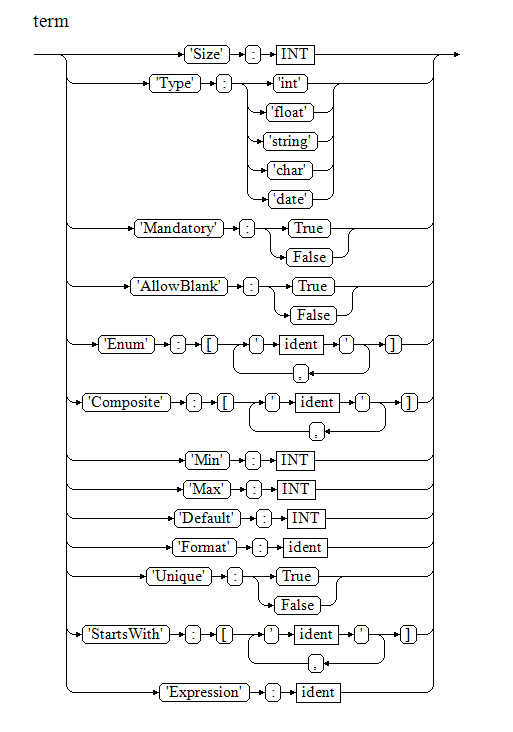
The following diagrams illustrate the graphical structure of the grammar:











## **Description**

|  |
| --- |
| Metadata =  {  '<*attribute\_name*>':  {  '**Size**': n,  '**Type**': ‘xxx’,  '**Mandatory**': True | False,  '**AllowBlank**': True | False,  '**Unique**': True | False,  '**Composite**': [‘<attribute\_name>’,…,’<attribute\_name>’],  '**Enum**': [‘x’,…,’x’],  ‘**Format**’: ‘<python regular expression>’,  ‘**Min**’: n,  ‘**Max**’: n,  ‘**Default**’: ‘xxx’,  ‘**Expression**’: ‘<python source code>’,  ‘**StartsWith**’: [‘x’,…,’x’]  }  …  } |

## **Elements**

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Description | Example | Notes |
| <attribute\_name> | The unique attribute or field name in the data set. | ‘Name’  ‘Date\_of\_Birth’  ‘Phone Number’ | * These are used as indices in dictionaries and lists therefore must be uniquely named. * Names are case-sensitive. |
| Size | Specifies the maximum size of the expected values as an integer. | ‘Size’: 10 |  |
| Type | Specifies the expected data type. | Type: ‘int’ | Allowable values are:   * int * float * date * string * char |
| Mandatory | Specifies if a value is required or not. | Mandatory: True | Allowable values:   * True * False |
| AllowBlank | Determines whether or not blank values are acceptable for non-mandatory fields. | AllowBlank: True | * See Data Quality dimensions[[1]](#footnote-1) |
| Unique | Specifies if values in this attribute must be unique. | Unique: False |  |
| Composite | Specifies a composite key made up of two or more attributes from the dataset. | Composite: [‘First Name, ‘Last Name’]  Composite: [‘%1’, ‘Last Name’] | * pyduq will report an error if the combination of attributes is not unique. * %1 can be used as a placeholder for the attribute to which this is applied. |
| Enum | Specifies the set (enumeration) of valid values. | Enum: [‘N’,’S’,’E’,’W’]  Enum: [‘0’,’1’,’2’,’3’] |  |
| Format | Specifies a format that the value must match. | Format: ‘.\*’ | See Python regular expressions. |
| Min | Specifies the minimum acceptable value. | Min: ‘1’ | Must be numeric. |
| Max | Specifies the maximum acceptable value. | Max: ‘99’ | Must be numeric. |
| Default | Specifies an acceptable default value. | Default: ‘99’ | Caters for attributes that use a default value in place of blanks. |
| Expression | Execute the specified python source code. | ,'Expression': '"0" <= "%1" <= "999"' | This should be used with caution. |
| StartsWith | Specifies one or more string to match values against. | StartsWith: ‘ABC’  StartsWith: [‘ABC’,’123’] | This could also be achieved using an expression or format. |

Example:

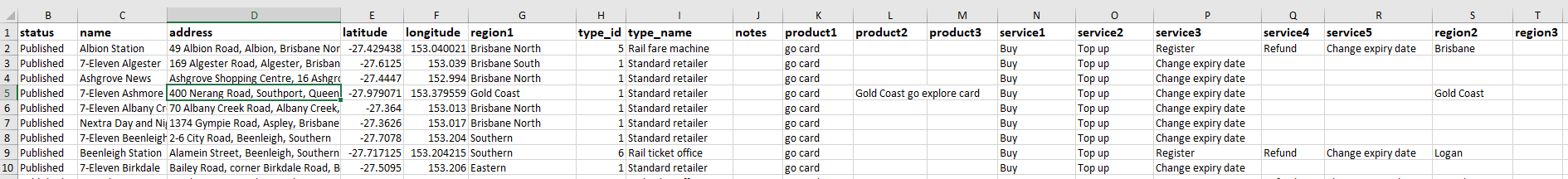
Many towns operate digital ticketing systems for their public transport operations to streamline services and provide a better commuter experience. In Queensland, Australia we use “Go Card” a plastic card with an embedded microchip to use for paying for travel on our public rail, bus, and ferry transport network. Go Cards can be purchased and topped up with additional funds at a number of retail outlets, and fixed ticketing machines. The data set used in this example provides the location and type of service offered by Go Card retailers and fixed ticketing machines. This data is provided by the State Government as an open data set[[2]](#footnote-2).

The data set does not come with a meta-data definition, so we need to create one by profiling the data. This can be done easily suing Microsoft Excel. The grammar does not need to be complete to be of use – it can be initially defined by:

1. Copying the file header role as the attributes
2. Estimating a size based on visually inspecting some data
3. Estimating the data type based on visually inspecting the data
4. The remaining terms (mandatory, unique, and so on) can be added as more is learnt about the data through visual inspection and fine tuning the meta-data

The structure of the data includes repeating columns (region, product and service) with identical naming. These have been suffixed with a number to create unique columns.

Example data:



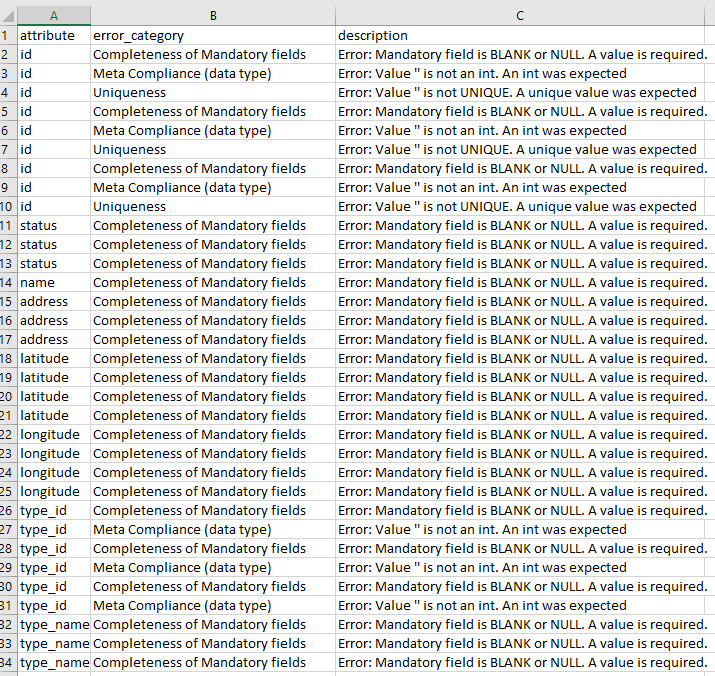
The following metadata was the first attempt at defining the dataset. The result generated a lot of errors – mostly due to field sizes being too small (for example, the ‘id’ column was set to 2 however there are values greater than 99, so it fails). Also, the ‘id’ column was though to be a unique identifier however 3 blanks were reported –these are presumed to be legitimate errors.

|  |
| --- |
| meta = {  'id': {'Size': 2, 'Type': "int", 'Mandatory': True, 'Unique': True},  'status': {'Size': 9, 'Type': "string", 'Mandatory': True },  'name': {'Size': 50, 'Type': "string", 'Mandatory': True},  'address': {'Size': 10, 'Type': "date", 'Mandatory': True},  'latitude': {'Size': 10, 'Type': "string", 'Mandatory': True},  'longitude': {'Size': 10, 'Type': "string", 'Mandatory': True},  'region1': {'Size': 20, 'Type': "string", 'Mandatory': True},  'type\_id': {'Size': 1,'Type': "int", 'Mandatory': True},  'type\_name': {'Size': 20, 'Type': "string", 'Mandatory': True},  'notes': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product1': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product2': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product3': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service1': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service2': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service3': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service4': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service5': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'region2': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'region3': {'Type': "string", 'Mandatory': False, 'AllowBlank': True}  } |

After using the errors to further tune the meta-data, it now looks like this:

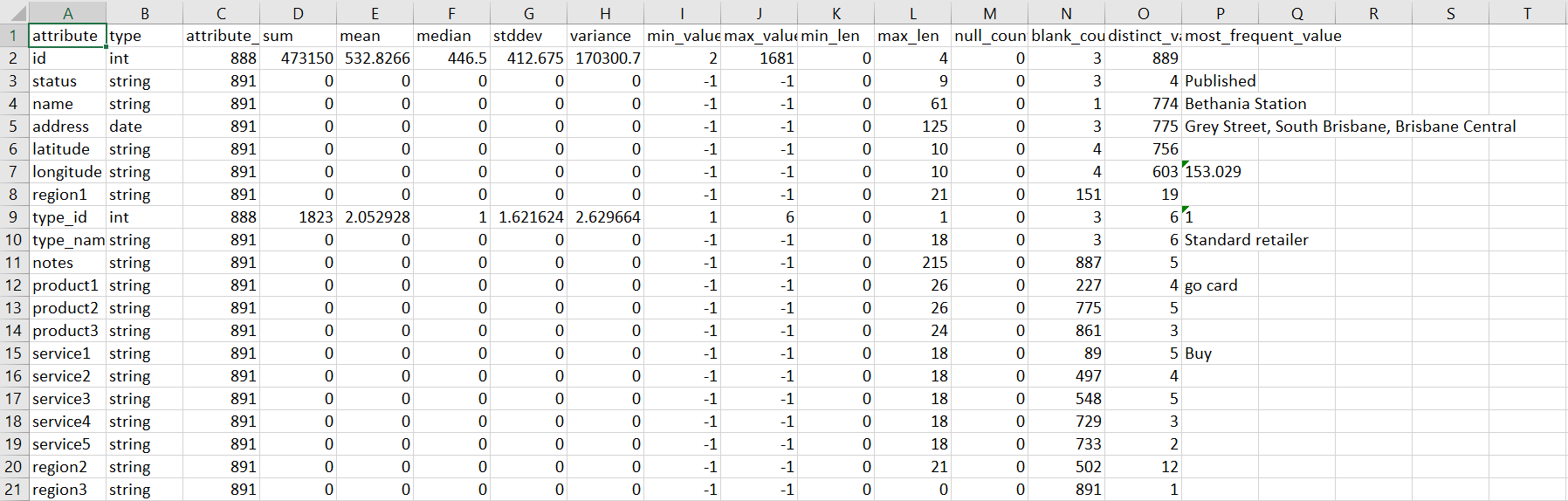
|  |
| --- |
| meta = {  'id': {'Size': 4, 'Type': "int", 'Mandatory': True, 'Unique': True},  'status': {'Size': 10, 'Type': "string", 'Mandatory': True },  'name': {'Size': 70, 'Type': "string", 'Mandatory': True},  'address': {'Size': 140, 'Type': "date", 'Mandatory': True},  'latitude': {'Size': 10, 'Type': "string", 'Mandatory': True},  'longitude': {'Size': 10, 'Type': "string", 'Mandatory': True},  'region1': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'type\_id': {'Size': 1, 'Type': "int", 'Mandatory': True},  'type\_name': {'Size': 20, 'Type': "string", 'Mandatory': True},  'notes': {'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product1': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product2': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'product3': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service1': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service2': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service3': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service4': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'service5': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'region2': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True},  'region3': {'Size': 30, 'Type': "string", 'Mandatory': False, 'AllowBlank': True}  } |

The following residual errors exist which appear to be legitimate errors:



The number of errors could be further reduced by allowing blank values on all attributes if you take it for granted that there will be blanks and manage the data accordingly.

pyduq performs a profile of the data which can also be used to fine tune the model. Included in the output is the min and max values for each numeric attribute, maximum attribute lengths and counts of nulls and blanks as seen below:



1. Blanks are generally not permitted as per the Completeness data quality dimension. See: <http://dke.uqcloud.net/DataQualityPatterns/?page_id=165&cha_title=Completeness%20of%20mandatory%20attributes> [↑](#footnote-ref-1)
2. See: <https://translink.com.au/about-translink/open-data> [↑](#footnote-ref-2)