File Format: PGF (Affymetrix Power Tools v1.6.1)

The PGF (probe group file) provides information about what probes are contained within a probeset and information about the nature of the probes necessary for analysis. The current PGF file format (version 1) is only specified for expression style probesets.

The PGF file is based on version 2 of the **TSV file format** .

Specifications

- Required headers
    - chip_type: indicates the chip type (as stored in the CEL file) which the pgf file describes. Multiple chip_type headers may be present indicating the pgf file can be used with any one of the listed chip types.
    - lib_set_name: indicates the name of a collection of related library files for a given chip. For example, PGF and CLF files intended to be used together should have the same lib_set_name. Only a single lib_set_name is allowed.
    - lib_set_version: indicates the version of a collection of related library files for a given chip. For example, PGF and CLF files intended to be used together should have the same lib_set_version. Only a single lib_set_version is allowed.
    - pgf_format_version: currently the only documented and supported version is 1.0.
    - header0: indicates the values in the pgf file for the top level data (probeset level). A pgf file probeset level entry must contain probeset_id. This field must be unique over all the probeset_ids in the PGF file. Optional, but typical fields include type and probeset_name.
    - header1: the second level of data in the PGF file (atom level). An atom refers to a particular collection of probes that are interrogating the same position. For expression arrays an atom is usually a probe pair (pm and mm probe pair) from an array like HG-U133 Plus 2.0 or a single pm probe from an array like Human Exon 1.0 ST which does not contain mismatch probes. The atom level data must have an atom_id field. atom_ids are expected to be unique within the PGF file. In other words, atoms cannot be present in more than one probeset or present multiple times in any given probeset.
    - header2: the third level of data (probe level) contains information about a particular probe in a particular probeset. A probe may be present in more than one probeset, but it may only appear once in a given probeset. Required fields include probe_id and type. Optional, but typical fields include gc_count, probe_length, interrogation_position, and probe_sequence. A probe_id is not

required to be unique within the PGF file -- ie that probe may be used in another probeset.

- Optional headers
  - create_date: timestamp for when the file was created
  - guid: a unique identifier for the fileA
  - other headers may be present
- Columns
  - per TSV format, order of columns is not guarateed
  - per TSV format, additional columns may be present
  - Probeset Level (level 0)
    - probeset_id (required): an integer id >= 0 which is unique over all the probeset_ids in the pgf file
    - type (optional): provides classification information for the probeset. See Type section.
    - probeset_name (optional)
  - **Atom** Level (level 1)
    - atom_id (required): an integer id >= 0 which is unique over all the atom_ids in the pgf file
    - type (optional): provides classification information for the atom. See Type section.
    - exon_position (optional): the position of the probe interrogation position relative to the target sequence
  - **Probe** Level (level 2)
    - probe_id (required): an integer id >= 0 which is a foreign key into the CLF file; a specific probe may be present in more than one probeset and as such is not guarateed to be unique in the PGF file. Also note that the additional columns of information at the probe level may be context dependent. So for example a particular probe could potentially be a PM probe in one probeset and an MM probe in another. While unlikely, this is a possibility.
    - type (required): provides classification information for the probe. See the Type section.
    - gc_count (optional): the number of G and C bases in the probe
    - probe_length (optional): the length of the probe
    - interrogation_position (optional): the interrogation position of the probe (typically 13 for 25-mer PM/MM probes)
    - probe_sequence (optional): the sequence of the probe on the array in from array surface to solution. For most standard Affymetrix arrays at this writing, this would be in a 3' to 5' direction. So for a sense target (st) probe (see the type field for the probe level) you would need to complement the sequence in this field before looking for matches to transcript sequences; for an antisense target (at) you would need to reverse this sequence.

Types

Type columns in PGF files use the following string format to catagorize probesets, atoms, and probes:

```
simple_type:=[a-z0-9\_\-]+
```

So an example simple type

```
pm
mm
st
at
control
affx
spike
```

Furthermore, types can be nested. For example a particular spike may be from Affymetrix and is intended for use as a control. As a result you would combine the simple types to reflect this:

```
control->affx->spike
```

Thus

```
nested_type:=(simple_type|nested_type)->(simple_type)
```

Lastly, a given probeset, atom, or probe may belong to multiple independent types. For example, a probeset may be both a normalization control gene and part of the main design:

```
normgene->exon:main
```

Thus

```
compound_type:=(simple_type|nested_type|compound_type):(simple_ty
pe|nested_type)
```

Currently type values are not strongly enumerated. Values used in current commercial PGF files include:

- Probeset Type
  - main: probesets which are a part of the main design for which the array was designed
  - normgene->exon: probe sets against exon regions of a set of housekeeping genes
  - normgene->intron: probe sets against intron regions of a set of housekeeping genes

    - control->affx: standard Affymetrix spike control probeset (ie bacterial and polyA spikes)
    - control->bgp->antigenomic: antigenomic background probes
    - control->bgp->genomic: genomic background probes
    - control->spike->arabidopsis: arabidopsis spike control probesets
    - rescue->FLmRNA->unmapped: probesets against mRNA sequences which did not align to the genome
    - types removed in more recent PGF files:
        - control->QC: manufacturing control probesets
        - control->chip: internal control probesets used for gridding and other internal functions
- **Atom** Type
    - (none)
- **Probe** Type
    - pm: perfect match probe relative to intended target
    - mm: specific mismatch probe
    - st: a probe designed for sense target
    - at: a probe designed for antisense target
    - types removed in more recent PGF files:
        - blank: a blank feature
        - generic: an internal control feature
        - jumbo-checkerboard: an internal gridding feature
        - thermo: an internal control feature
        - trigrid: an internal gridding feature

Parsing and Writing

The official C++ parser used by affy can be found in APT under sdk/file/TsvFile/PgfFile.h. When possible, parsing and writing of PGF files should be done using this code.

Notes

Specific applications may require extra/optional columns in the PGF file. Thus a valid PGF file may fail for a particular application or analysis algorithm because the information needed by that application and/or algorithm is not contained in the PGF file.

It should be noted that there is no significance to the ordering of probes within atoms and atoms within probesets or even probesets within the PGF file.

IDs do not have to be unique between different levels. In other words, the ID space for probeset_ids is separate from the ID space for atom_ids and probe_ids.

Example 1

```
#%chip_type=HuEx-1_0-st-v2
#%chip_type=HuEx-1_0-st-v1
#%chip_type=HuEx-1_0-st-ta1
```

```
#%lib_set_name=HuEx-1_0-st
#%lib_set_version=r2
#%create_date=Tue Sep 19 15:18:05 PDT 2006
#%guid=0000008635-1158704285-0183259307-0389325148-0127012107
#%pgf_format_version=1.0
#%header0=probeset_id   type
#%header1=      atom_id
#%header2=              probe_id        type    gc_count
probe_length    interrogation_position  probe_sequence
2590411 main
        1
                5402769 pm:st   12      25      13
CGAAGTTGTTCATTTCCCCGAAGAC
        2
                4684894 pm:st   13      25      13
ATGAGGTCACGACGGTAGGACTAAC
        3
                3869021 pm:st   11      25      13
AGGAGTACAGGGTAAGATATGGTCT
        4
                3774604 pm:st   14      25      13
CCCCGAAGACCCTAAGATGAGGTCA
...
```

Example 2 -- Human Genome U133 2.0 Plus PGF File Excerpt

Here is a hypothetical example of a PGF file for an expression array with PM/MM pairs.

```
#%pgf_format_version=1.0
#%chip_type=HG-U133_Plus_2
#%lib_set_name=HG-U133_Plus_2
#%lib_set_version=1
#%create_date=Tue Mar 29 16:48:05 2005
#%header0=probeset_id   type    probeset_name
#%header1=      atom_id
#%header2=              probe_id        type    gc_count
probe_length    interrogation_position  probe_sequence
exon_position
1354897         204339_s_at
        1354898
                1221821 pm:target->at   13      25      13
ACAACGACCGTTCCGGAATCGACAT       1703
                1222985 mm:target->at   13      25      13
ACAACGACCGTTGCGGAATCGACAT       1703
        1354899
```

```
                788881  pm:target->at   8       25      13
TTACATCATACTTTCTTGTCTCTAG       1355
                790045  mm:target->at   8       25      13
TTACATCATACTATCTTGTCTCTAG       1355
        1354900
                516645  pm:target->at   12      25      13
GAATCTTATCACGAGTTCCACCTCC       1518
                517809  mm:target->at   12      25      13
GAATCTTATCACCAGTTCCACCTCC       1518
        1354901
                736948  pm:target->at   12      25      13
GAGTTCCCTCGATTCCATAAACGAG       1543
```

Related Pages

- TsvFile Design Notes
- TSV File Format

Developer Notes

- If you are defining a file format which is based on TsvFile, use this as an example.
- PgfFile is not subclassed as we dont want it to inherit methods from TsvFile. This prevents calling a method by accident.
- m_tsv is a member so it appears when a PgfFile is made. (ie: minus messy memory management)
- We make "tsv" public so if the programmer wants to work with the underlying tsv object may do so.
