# Classes Used in the Oligo Package

Benilton Carvalho

March, 2007

## 1 Introduction

This document describes the classes used in the **oligo** package. The **oligo** package uses essentially two groups of classes:

- Static data classes: these are chip-specific information. Each chip contains its own annotation, which is shared across experiments that used that array. These are generated by the **makePlatformDesign** and **pdInfoBuilder** packages.

- Experimental data classes: these classes refer to the experimental data, ie. CEL and XYS files that the user has. All the experimental data classes derive from **eSet** defined in **Biobase**.

The *platformDesign* is one of the static data classes and is generated by the **makePlatformDesign** package. It is a container for the chip-specific information. We are transitioning the creation of the chip-specific packages to the **pdInfoBuilder**, which makes more efficient use of memory (via SQLite) and is much more flexible than the environment approach used by **makePlatformDesign**.

## 2 *platformDesign* Class

The *platformDesign* class is the container for information on the expression (NimbleGen), tiling (Affymetrix, NimbleGen) and exon (Affymetrix) arrays. It contains the following slots:

- **manufacturer**: lower case string containing the name of the manufacturer of the array (eg., ■affymetrix■ or ■nimblegen■).

- **genomebuild**: lower case string containing the genome release information using the USCS notation, as described at `http://genome.ucsc.edu/FAQ/FAQreleases#release1`.

- **featureInfo**: an environment containing vectors of same length which fully characterizes the array being used. See details below.

| Slot | Type |
|---|---|
| manufacturer | *character* |
| genomebuild | *character* |
| featureInfo | *enviroment* |
| featureTypeDescription | *list* |
| type | *character* |
| nrow | *numeric* |
| ncol | *numeric* |
| nwells | *numeric* |
| lookup | *data.frame* |
| indexes | *list* |
| platforms | *character* |

Table 1: Description of the *platformDesign* class

- `type`: a string describing the type of the array (eg., ■`expression`■, ■`tiling`■, ■`exon`■, ■`SNP`■).

- `nrow` and `ncol`: array dimensions.

- `nwells`: number of wells (specific for NimbleGen data).

- `lookup`: data.frame used to map features in complex NimbleGen designs.

- `indexes`: not used anymore. To be removed.

- `platform`: not used anymore. To be removed.

## 2.1 Details on the `featureInfo` slot

The `featureInfo` is the home for the majority of the information used by oligo. `featureInfo` is an *environment* containing the following vectors:

- `X` and `Y`: X/Y coordinates on the array. Class: *integer*.

- `feature_set_name`: name of the featureset (probeset). Class: *character*.

- `feature_ID`: match ID between PM and MM. *integer*.

- `feature_type`: type of the feature. Class: *factor*. (PM/MM)

- `target_strand`: target strandness. Class: *factor*. (antisense/sense)

- `sequence`: probe sequence. Class: *character*.

- `length`: probe length. Class: *integer*.

- `chromosome`: chromosome. Class: *character*. (chr1/chr22/chrX)

- `ambiguous_feature`: indicator if sequence is mapped to more than one genomic location. *logical*

| Field | Expression | Tiling | Exon |
|---|:---:|:---:|:---:|
| `X` and `Y` | ✓ | ✓ | ✓ |
| `feature_set_name` | ✓ | ✓ | ✓ |
| `feature_ID` | ✓ | ✓ | ✓ |
| `feature_type` | ✓ | ✓ | ✓ |
| `target_strand` | ✓ | ✓ | ✓ |
| `sequence` | ✓ | ✓ | ✓ |
| `order_index` | ✓ | ✓ | ✓ |
| `length` | | ✓ | ✓ |
| `chromosome` | | ✓ | |
| `ambiguous_feature` | | ✓ | |
| `position` | | ✓ | |
| `location` | ✓ | | ✓ |
| `atomID` | | | ✓ |
| `gc_count` | | | ✓ |

Table 2: Fields in `featureInfo`

- `position`: genomic location within chromosome. *numeric*

- `location`: genomic location within chromosome. To be removed and merged with `position`.

- `atomID`: pairing key between PM-MM.

- `gc_count`: number of GC bases. To be removed, as this can be obtained from the sequence information.

### 2.1.1 Particularities of Tiling Arrays

For tiling arrays, I have been using the genomic position as `feature_set_name`, but it is not uncommon to have a probe sequence matching $k > 1$ genomic positions. In situations like this, the `feature_set_name` is set as the concatenation of the $k$ genomic positions using ■;■ as separator and `ambiguous_feature` is set `TRUE`. For example:

| sequence | position | feature_set_name | ambiguous_feature |
|---|:---:|:---:|:---:|
| `AAATC...GCCAT` | 12345 | ■12345■ | FALSE |
| `CCACG...ATTCC` | 34567 / 87654 | ■34567;87654■ | TRUE |

Table 3: Naming convention for tiling arrays

An even more effective naming convention would be `CHRnnPmmmmmm`, which would be more robust on designs that involve multiple chromosomes.

### 2.1.2 Particularities of Exon Arrays

A basic support of Exon Arrays is offered by oligo.

### 2.1.3 Particularities of SNP Arrays

The data packages for SNP arrays are now built via pdInfoBuilder package. The packages for the Affymetrix 100K and 500K sets are available via BioConductor.

### 2.1.4 About `order_index`

In the final version of the data packages, the fields described on Table 2 are ordered by `feature_set_name`, `feature_type` and `target_strand`. Note that this breaks the link between the intensity file (which is often ordered by X/Y location) and the annotation available in the *platformDesign* object.

In order to keep this link, we initially order the *platformDesign* object by X/Y location, so it matches the intensities files. Then we add the field `order_index`, which is only the row number. Later, the *featureInfo* object is re-ordered by `feature_set_name`, `feature_type` and `target_strand`. But with the presence of `order_index`, we can correctly map the intensities to their probe-level annotations.

## 3 *DBPDInfo* Class

The *DBPDInfo* class is the database approach for the *platformDesign* class. Table 3 describes the class structure.

| Slot | Type |
|------|------|
| getdb | `function` |
| tableInfo | `data.frame` |
| geometry | `integer` |
| manufacturer | `character` |
| genomebuild | `character` |

Table 4: Description of the *DBPDInfo* class

- `getdb`: function that accesses the external database (we use SQLite, via RSQLite);

- `tableInfo`: a data.frame with two columns (`tbl` and `row_count`). This data.frame contains the name and number of rows of each table available in the database.

- `geometry`: an integer vector of length 2, containing the number of rows and columns of the array;

- `manufacturer`: a string with the manufacturer's name;

- `genomebuild`: a string with the genome build information.

# 4  *FeatureSet* Class

The *FeatureSet* class is a virtual class to be used with the feature-level data and is created from the *eSet* class. Different classes are created from this:

- *ExpressionFeatureSet*: for expression arrays;

- *SnpFeatureSet*: for SNP arrays;

- *ExonFeatureSet*: for exon arrays;

- *TilingFeatureSet*: for tiling arrays.

# 5  *SnpQSet* Class

The *SnpQSet* class is created by the `snprma()` method. It contains four matrices, which contain the summarized information for SNP data. The four matrices are:

- `antisenseThetaA`: summarized data at the SNP-level for the antisense strand and allele A;

- `antisenseThetaB`: summarized data at the SNP-level for the antisense strand and allele B;

- `senseThetaA`: summarized data at the SNP-level for the sense strand and allele A;

- `senseThetaB`: summarized data at the SNP-level for the sense strand and allele B;

  This is the expected input to the genotyping algorithm, `crlmm()`.

# 6  *SnpCallSet* Class

The *SnpCallSet* class is a container for the output of genotyping algorithm, eg. `crlmm()`. It contains two matrices: `calls` and `callsConfidence`, which hold respectively the genotype calls and associated measures of confidence.

# 7  *SnpCopyNumberSet* Class

The *SnpCopyNumberSet* class is a container for the output of copy number analisys. It contains two matrices: `copyNumber` and `copyNumberConfidence`, which hold respectively the copy number estimates and associated measures of confidence.