## Flapjack XML/Binary Project Format Specification

Valid for projects created by

Flapjack **x.xx.xx.xx** (binary **V04**)

Iain Milne and Gordon Stephen

March 2011



**XMLRoot**

**Project**

**DataSet**...

[this block is mainly “data”]

**ChromosomeMap**...

**Marker**...

**QTL**...

->*ChromosomeMap*

**GraphData...**

**Line**...

**GenotypeData**...

->*ChromosomeMap*

**TraitValue**...

->*Trait*

**Trait**...

**StateTable**

**AlleleState**...

**DBAssociation**

[end\_block]

[this block all relates to visualization]

**Line** (see dummyLine notes in DataSet)

**GTViewSet**...

->*DataSet*

**LineInfo**... (x2: lines and hideLines)

->*Line*

**GTView**...

->*GTViewSet*

->*ChromosomeMap*

**MarkerInfo**... (x2: markers and hideMarkers)

->*Marker*

QTLInfo...

->QTL

**Bookmark**...

->*ChromosomeMap*

->*Line*

->*Marker*

[end\_block]

... means a list of this type

-> means a reference to this type

Items marked in red are subject to change. Note that these currently only relate to additional, supplementary data (traits and QTL), and any changes should only result in that data being removed/ignored from existing project files, rather than the project itself breaking.

All data API classes extend from **XMLRoot** that provides them with a GUID.

A Project contains a list of **DataSets**.

Raw data:

Each **DataSet** contains a list of **ChromosomeMaps** (one per chromosome from the original map file) and a list of **Lines** (one per line from the original genotype file). A list of **Traits** holds information on each trait type that has been imported (its name, but not the data). The **StateTable** represents the Flapjack alphabet. Finally, any URL linking information is held by the **DBAssociation** object.

A **ChromosomeMap** contains a list of **Markers**, along with a list of **QTL** that map to it, and a list of **GraphData** objects that contain graph information for the markers.

The **QTL** class inherits from the more generic **Feature**, and also contains a reference back to the **ChromosomeMap** that contains it.

Each **Line** contains a list of **GenotypeData** objects to hold the raw data (alleles), one per chromosome, with the **GenotypeData** containing a reference back to the appropriate **ChromosomeMap**. A **Line** also holds a list of **TraitValue** objects, each holding a value or score for a given **Trait**.

The **StateTable** represents the Flapjack alphabet, formed from a list of **AlleleState** objects, one per allele type (A, C, A/T, etc) found in the raw data.

Visualization data:

Flapjack can display multiple views of the same **DataSet**, each held in a **GTViewSet** object which contains a reference back to the **DataSet** it represents.

The **GTViewSet** contains a list of **LineInfo** wrappers; its order defining the order of visible **Lines** in the display (hidden lines are held in a separate list). Each **LineInfo** refers back to the actual **Line** data.

The **GTViewSet** also contains a list of **GTView** objects, each representing a single chromosome. A **GTView** refers back to its containing **GTViewSet** and to the **ChromosomeMap** it represents. A list of **MarkerInfo** wrappers defines the order of the visible **Markers** within the chromosome (hidden markers are held in a separate list). Each **MarkerInfo** refers back to the actual **Marker** data. A list of QTLInfo wrappers defines the QTL which are included in each view. Each QTLInfo refers back to the actual QTL data.

The **GTViewSet** also contains a list of **Bookmarks**. A **Bookmark** refers to the **ChromosomeMap** it was selected on, along with the clicked-on intersection information – for the **Line** and **Marker**.

Data API classes that don’t form part of the output file format:

DataSetTest

FeatureGroup

UndoManager

UndoManagerTest

IUndoState (interface)

**XMLRoot**

The **XMLRoot** class is used as the base type for all data API objects that are serialized to XML. It provides a single inheritable variable – guid – that defines a unique (within the context of the current project) identifier for an object instance.

Issued IDs start with a value of 0, and increase incrementally as each new object is created. They are written to the XML as a string to maintain compatibility with previous Flapjack projects that used an ID formed from a combination of 16 alphanumeric characters (eg, 6c8c1e0e8235e9c2). Internally, the ID is stored as a 32bit floating point number.

Attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

**Project**

The **Project** class is the containing object for a given Flapjack project. Only one project can exist at a time in Flapjack’s memory. A Project contains a list of **DataSet** objects, one per data set that has been imported into Flapjack.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Elements:

|  |  |  |
| --- | --- | --- |
| dataSets | data-sets | DataSet [list] |

**Project – BINARY**

|  |  |
| --- | --- |
| INT | dataSets count |
| **DataSet** [list] | ... |

All data – including the Flapjack classes - is written using the following primitives. Further details are available in the JavaDoc file for the java.io.DataOutputStream class.

|  |  |
| --- | --- |
| INT | Writes an int to the underlying output stream as four bytes, high byte first. |
| FLOAT | Converts the float argument to an int using the method below, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.  Conversion returns a representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout.  Bit 31 (the bit that is selected by the mask 0x80000000) represents the sign of the floating-point number. Bits 30-23 (the bits that are selected by the mask 0x7f800000) represent the exponent. Bits 22-0 (the bits that are selected by the mask 0x007fffff) represent the significand (sometimes called the mantissa) of the floating-point number.  If the argument is positive infinity, the result is 0x7f800000.If the argument is negative infinity, the result is 0xff800000. If the argument is NaN, the result is 0x7fc00000. |
| BOOLEAN | Writes a boolean to the underlying output stream as a 1-byte value (0=false, 1=true). |
| STRING | Firstly writes the length of the string as an INT.  Then writes the string itself as an array of bytes (UTF8 encoded). |
| BYTE | Writes out a byte to the underlying output stream as a 1-byte value. |

The guid values are never written to the binary format because every object gets a new value when it is loaded anyway.

**DataSet**

The **DataSet** class is the main container for all information associated with a given set of inputs (map file, genotype file, phenotypes, features/QTL). It contains the raw data, optimized for storage in a form that best allows for visualization, and additional, Flapjack-specific information on the views that have been created upon this data.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Elements:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| chromosomes | chromosome-maps | ChromosomeMap [list] |
| lines | lines | Line [list] |
| traits | traits | Trait [list] |
| dbAssociation | db-association | DBAssociation |
| stateTable | state-table | StateTable |
|  |  |  |
| viewSets | view-sets | GTViewSet [list] |

Additional comments:

It’s probably too late now, but a better location for this object may have been within each **GTViewSet**, therefore keeping all visualization related elements together in a single (list) of objects.

Dummy line information was removed with V03.

**DataSet – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| **StateTable** | statTable |
| INT | chromosomes count |
| **ChromosomeMap** [list] | chromosomes... |
| INT | lines count |
| **Line** [list] | lines... |
| INT | viewSets count |
| **GTViewSet** [list] | viewSets... |
| BOOLEAN | hasDummyLine |
| **DBAssociation** | dbAssociation |

Additional comments:

The binary format has an additional variable over the XML – hasDummyLine – that specifies whether a dummy line has been written into the file or not. This code was bugged and didn’t actually work however no complaints about it were ever received, so we assumed nobody ever saved a binary project with dummy lines. When split line functionality was added this bug was fixed, and the actual Line data was no longer serialized. The Boolean remains for backwards compatibility.

**ChromosomeMap**

A **ChromosomeMap** is used to contain all information about a given chromosome; its name and length, along with the markers that are located upon it and any features that can be mapped against it.

The variable isSpecialChromosome tags a chromosome that is being used to hold markers from more than one “original” chromosome, eg, a super-chromosome that might hold all markers together to allow for side-by-side viewing.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| length | length | float |
| isSpecialChromosome | special-chromosome | boolean |

Elements:

|  |  |  |
| --- | --- | --- |
| markers | markers | Marker [list] |
| QTLs | QTLs | QTL [list] |
| graphs | graphs | GraphData [list] |

**ChromosomeMap – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| FLOAT | length |
| BOOLEAN | isSpecialChromosome |
| INT | markers count |
| **Marker** [list] | markers... |
| INT | features count |
| **QTL** [list] | qtl... |
| INT | graphs count |
| **GraphData** [list] | graphs... |

Additional comments:

A chromosome holds a list of QTL which is now the only type of feature supported.

**Marker**

The **Marker** class is used to represent a marker or locus. It has a name and a position (on the chromosome). This data is read in during the initial import operation.

Each marker also maintains a list of allele frequency information, that is, it tracks the frequency of each allele state within the data set (see also **StateTable** and **AlleleState**), localized to just this marker. For example, if the data only contained the states A, C, G, and T, and a marker (over a sample 4 lines) contained AAAT, then the frequency information would be A=0.75, C=0, G=0, T-0.25.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| position | position | float |
| realPosition | real-position | float |

Additional comments:

The name is currently stored as a string, but over the huge data sets (> 250,000 markers) this starts to add up to a worryingly large amount of memory. A possibly solution (Flapjack only) might be to store only an integer reference that refers to a cache file on disk that holds the actual name(s). Saving/loading could handle the translation to ensure it is always the actual name that stays in the project file.

With the updates to support side-by-side viewing of markers from more than one chromosome, the **Marker** class has been changed to hold both a position and a realPosition variable, the latter holding its correct chromosome position from the “original” chromosome it came from, and position instead being used to specify where it is on the “fake” chromosome.

Marker instances with a realPosition value of -1000 are classed as “dummy markers” and are used as fillers between the markers from two different chromosomes in side-by-side views.

**Marker – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| FLOAT | position |
| FLOAT | realPosition |

**QTL**

The QTL contains basic information including its name and a starting (min) and ending (max) position on the chromosome.

Additional, display-only data are also stored here. This isVisible variable defines whether the feature should be drawn or not. The isAllowed variable defines whether a qtl is allowed to be displayed or not (regardless of its visibility). This relates to qtl that may include position information that would place them beyond the limits of their containing chromosome. Flapjack will continue to list them, but will obviously not render them. Colour information that determines what colour the feature will be rendered in is also stored.

The position variable defines a QTL’s exact position on the chromosome, allowing min and max to define its error margin. It also stores a reference back to the actual **ChromosomeMap** object that contains it.

The elements VNames and values are an optional set of strings that correspond to any additional columns that have been added to the imported file. VNames stores the header information, with the actual values (numerical or categorical) being stored in the values list.

Attributes:

|  |  |  |
| --- | --- | --- |
| guid | guid | string |
| name | name | string |
| min | min | float |
| max | max | float |
|  |  |  |
| isVisible | visible | boolean |
| isAllowed | allowed | boolean |
| red | red | integer (0-255) |
| green | green | integer (0-255) |
| blue | blue | integer (0-255) |
| position | position | float |
| trait | trait | string |
| experiment | experiment | string |
|  |  |  |
| chromosome | chromosome-map | ChromosomeMap [ref] |

Elements:

|  |  |  |
| --- | --- | --- |
| vNames | VNames | string [list] |
| values | values | string [list] |

Additional comments:

Used to be a subclass of Feature, all of the Feature functionality was rolled up into the QTL class (barring dbKey which has never been used).

**QTL – BINARY**

|  |  |
| --- | --- |
| FLOAT | position |
| INT | length of trait |
| STRING | trait |
| STRING | experiment |
| INT | vNames count |
| STRING [list] | vNames... |
| INT | values count |
| STRING [list] | values... |
| STRING | name |
| FLOAT | min |
| FLOAT | max |
| BOOLEAN | isVisible |
| BOOLEAN | isAllowed |
| INT | red |
| INT | green |
| INT | blue |

Additional notes:

The reference back to a QTL’s containing chromosome is not written, as this can be reconstructed at load time due to the **ChromosomeMap** that contains it being found before we get to its list of **QTL** objects.

**GraphData**

The **GraphData** class holds the data for drawing a single graph against the current chromosome. Flapjack supports multiple graphs being loaded at once, hence the **ChromosomeMap** holding a list of **GraphData** objects, one per actual graph. Although a “graph” is actually a construct across all chromosomes for a given data set, each **ChromosomeMap** holds its own graph (to aid visualization) although some variables within it will be shared across all objects, such as the min, max, and threshold.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| minimum | minimum | float |
| maximum | maximum | float |
| hasThreshold | has-threshold | boolean |
| threshold | threshold | float |

Elements:

|  |  |  |
| --- | --- | --- |
| data | data | string |

Additional comments:

The data for a graph is normalized after import, so all the values in the project file will be between 0 and 1. The original value can obviously be reconstructed using the minimum and maximum variables.

The data itself, while internally a float[] array to Flapjack is stored in the XML as a continuous string of numbers, separated by colons. This significantly reduces the amount of XML required.

**GraphData – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| FLOAT | minimum |
| FLOAT | maximum |
| BOOLEAN | hasThreshold |
| FLOAT | threshold |
| INT | data count |
| INT [array] | data... |

**Line**

The **Line** class is used to represent a line. It holds all the information for a given line (across all known markers) for both allele and phenotype trait data. Allele data is read during an initial import, with phenotype data being an additional import type.

An additional attribute – dbKey – is currently unused. All lines in all current projects will contain this with its value set to 0.

The allele data is stored in a list of type **GenotypeData**, one for each chromosome.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| dbKey | db-key | integer |

Elements:

|  |  |  |
| --- | --- | --- |
| genotypes | genotypes | GenotypeData [list] |
| traitValues | trait-values | TraitValue [list] |

**Line – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| INT | genotypes count |
| **GenotypeData** [list] | genotypes... |
| INT | traitValues count |
| **TraitValue** [list] | traitValues... |

**GenotypeData**

The **GenotypeData** class holds the raw allele information for a line. Due to the way Flapjack’s alphabet (state table) works, this information is held in either an array of bytes (up to 128 allele states available) or if more are needed, then an array of integers is used (at 4x the memory cost). Note that only one or the other will be used; never both.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| map | chromosome-map | ChromosomeMap [ref] |

Elements:

|  |  |  |
| --- | --- | --- |
| loci | loci | string |
| loci-int | loci-int | integer [list] |

Additional comments:

Note also that the loci data (which is an array of bytes internally to Flapjack) is serialized to XML as a single string. This is a BASE64 encoded representation of the array (I’m led to believe). This is purely because of the way the Castor XML API behaves.

**GenotypeData – BINARY**

|  |  |
| --- | --- |
| INT | index within **DataSet** of containing **ChromosomeMap** |
| BOOLEAN | true if byte based storage; false if int based |
|  |  |
| INT | loci count |
| BYTE [array] | loci... |
|  |  |
| INT | loci-int count |
| INT [array] | loci-int... |

Additional comments:

The binary format for this class differs from its XML counterpart.

Rather than storing a reference to the **ChromosomeMap**, we instead store the integer index of the map as it is found within the **DataSet**. This alone is enough to allow the map to be re-associated with the **GenotypeData** object during deserialization.

A boolean then dictates whether the following data will be an array of bytes or an array of integers, based on how Flapjack has stored the data within memory (integer arrays are only used once the number of entries in a **StateTable** rises above 128).

**TraitValue**

The **TraitValue** class holds information about the *values* for a line for a given trait.

The value is either the actual value (numerically) or the index number of its categorical value (from the list held by the actual **Trait** object). An additional, normalized across all lines, value is held in the variable normal.

The variable isDefined is set to false for lines that do not have any value for this trait.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| value | value | float |
| normal | normal | float |
| isDefined | defined | boolean |
|  |  |  |
| trait | trait | Trait [ref] |

**TraitValue – BINARY**

|  |  |
| --- | --- |
| FLOAT | value |
| FLOAT | normal |
| BOOLEAN | isDefined |
| INT | index within **DataSet** of containing **Trait** |

**Trait**

The **Trait** class provides information about a given trait. Its name, and optionally a list of categories of valid values that **TraitValue** objects can use when holding a value for this trait.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| experiment | experiment | string |

Elements:

|  |  |  |
| --- | --- | --- |
| categories | categories | string [list] |

**Trait – BINARY**

|  |  |
| --- | --- |
| STRING | name |
| INT | categories count |
| STRING [list] | categories... |
| STRING | experiment |

**StateTable**

The **StateTable** class holds the list of all possible **AlleleState** objects for a data set. This is essentially Flapjack’s alphabet. Each data point in the original file is stored within Flapjack as the index of that state within the state table.

So, for example, if a data set only contained the values A C, G, and T, then the state table would hold just four states: A =0, C = 1, G = 2, T = 3.

A line with data for 10 markers: A A G T T C A G T A would have that same data stored by Flapjack (in a **GenotypeData** object): 0 0 2 3 3 1 0 2 3 0

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Elements:

|  |  |  |
| --- | --- | --- |
| states | states | AlleleState [list] |

**StateTable – BINARY**

|  |  |
| --- | --- |
| INT | states count |
| **AlleleState** [list] | states... |

**AlleleState**

The **AlleleState** class holds information for a single instance of a type of (possible) allele within a data set.

For a given allele, say A/T we store the raw data as it was read from the file, and a list of all possible states within it (in this case, A and T).

An additional, automatically added state to represent unknown data is always the first **AlleleState** held by the **StateTable**. This allele state object has no raw data or possible states.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| isHomozygous | homozygous | boolean |
| isUnknown | unknown | boolean |

Elements:

|  |  |  |
| --- | --- | --- |
| rawData | raw-data | string |
| states | states | string [list] |

Additional notes:

Flapjack’s data model was designed to cope with more than just diploid data, although in reality this has never been used, and the visualization code can currently only cope with homozygous and bi-allelic heterozygous data types anyway.

The isUnknown variable is a mistake and was never meant to be written to the XML. It may get removed at some point.

**AlleleState** – BINARY

|  |  |
| --- | --- |
| BOOLEAN | isHomozygous |
| STRING | rawData |
| INT | states count |
| STRING [list] | states... |

**DBAssociation**

The **DBAssociation** class is used to link URL look-up strings with a data set.

Two possible look ups are available just now, for searching for either line or marker information by name.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| lineSearch | line-search | string |
| markerSearch | marker-search | string |

Additional information:

This object is likely to be expanded in the near future to hold information related to traits (phenotype and QTL or features).

**DBAssociation** – BINARY

|  |  |
| --- | --- |
| STRING | lineSearch |
| STRING | markerSearch |

**GTViewSet**

The **GTViewSet** class is used to represent a set of “views” upon the actual data. Each view within the set (using **GTView**) corresponds to one of the chromosomes.

Both **GTViewSet** and **GTView** are similar in that their main structure is formed for lists of lines or markers that are actually wrappers around the original objects. These classes (**LineInfo** and **MarkerInfo**) are held in lists that can be resorted, reordered, have elements removed/readded, etc; all useful functionality that can be done without affecting the original data.

Variables: colorScheme, randomColorSeed, comparisonLineIndex, comparisonLine, alleleFrequencyThreshold are all used to determine/display the colour scheme in use, which is global across all chromosomes of the view.

The traits string represents an internal array, written to the XML as a colon separated string of the indexes of any traits currently enabled within the view.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| name | name | string |
| viewIndex | view-index | integer |
| colorScheme | color-scheme | integer |
| randomColorSeed | random-color-seed | integer |
| comparisonLineIndex | comparison-line-index | integer |
| alleleFrequencyThreshold | allele-frequency-threshold | float |
| displayLineScores | display-line-scores | boolean |
| graphIndex | graph-index | integer |
|  |  |  |
| dataset | data-set | DataSet [ref] |
| comparisonLine | comparison-line | Line [ref] |

Elements:

|  |  |  |
| --- | --- | --- |
| lines | lines | LineInfo [list] |
| hideLines | hide-lines | LineInfo [list] |
| bookmarks | bookmarks | Bookmark [list] |
| views | views | GTView [list] |
| traits | selected-traits | String |
| graphs | Selected-graphs | String |

**GTViewSet** – BINARY

|  |  |
| --- | --- |
| STRING | name |
| INT | viewIndex |
| INT | colorScheme |
| INT | randomColorSeed |
| INT | comparisonLineIndex |
| FLOAT | alleleFrequencyThreshold |
| BOOLEAN | displayLineScores |
| STRING | traits |
| INT | lines count |
| **LineInfo** [list] | lines... |
| INT | hideLines count |
| **LineInfo** [list] | hideLines... |
| INT | views count |
| **GTView** [list] | views... |
| INT | bookmarks count |
| **Bookmark** [list] | bookmarks |
| INT | graphIndex |
| STRING | graphs |

**LineInfo**

The **LineInfo** class provides a wrapper around a **Line** object, and is used to maintain a list of lines as part of a view (**GTViewSet**). The list can be reordered, or have elements removed, all without affecting the original data. We store a reference to the line itself, along with its index position in the original data set.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| index | index | integer |
| selected | selected | boolean |
| score | score | float |
|  |  |  |
| line | line | Line [ref] |

**LineInfo** – BINARY

|  |  |
| --- | --- |
| INT | index |
| BOOLEAN | isSelected |
| FLOAT | score |

Additional comments:

No reference to the actual **Line** is stored as it can be rebuilt when deserializing using the index alone.

**GTView**

The **GTView** class represents a single chromosome within the current view.

The variable markersOrdered is true while the markers remain in their original map order. If the user moves them around, then its value changes to false.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| comparisonMarkerIndex | comparison-marker-index | integer |
| markersOrdered | markers-ordered | boolean |
|  | marker-count | integer |
|  | line-count | integer |
|  |  |  |
| comparisonMarker | comparison-marker | Marker [ref] |
| viewSet | view-set | GTViewSet [ref] |
| map | chromosome-map | ChromosomeMap [ref] |

Elements:

|  |  |  |
| --- | --- | --- |
| markers | markers | MarkerInfo [list] |
| hideMarkers | hide-markers | MarkerInfo [list] |
| qtls | qtls | QTLInfo [list] |

Additional notes:

The marker-count and line-count values are a mistake and are not meant to be in the XML (nor are they needed). They will be removed in a later version.

**GTView** – BINARY

|  |  |
| --- | --- |
| INT | comparisonMarkerIndex |
| BOOLEAN | markersOrdered |
| INT | markers count |
| **MarkerInfo** [list] | markers... |
| INT | hideMarkers count |
| **MarkerInfo** [list] | hideMarkers... |

**MarkerInfo**

The **MarkerInfo** class provides a wrapper around a **Marker**  object, and is used to maintain a list of lines as part of a chromosome view (**GTView**). The list can be reordered, or have elements removed, all without affecting the original data. We store a reference to the marker itself, along with its index position in the original data set.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| index | index | integer |
| selected | selected | boolean |
|  |  |  |
| marker | marker | Marker [ref] |

**MarkerInfo** – BINARY

|  |  |
| --- | --- |
| INT | index |
| BOOLEAN | selected |

Additional comments:

No reference to the actual **Marker** is stored as it can be rebuilt when deserializing using the index alone.

**QTLInfo**

The **QTLInfo** class provides a wrapper around a **QTL** object, and is used to maintain a list of qtls as part of a chromosome view (**GTView**). This allows each chromosome view to have its own distinct set of QTL. We store a reference to the qtl itself, along with its index position in the original data set.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| index | index | integer |
|  |  |  |
| qtl | qtl | QTL [ref] |

**QTLInfo** – BINARY

|  |  |
| --- | --- |
| INT | index |

Additional comments:

No reference to the actual **QTL** is stored as it can be rebuilt when deserializing using the index alone.

**Bookmark**

The **Bookmark** class stores a reference to a location within the view, as selected by the user. It is an intersection between a line and a marker (and therefore the allele at that position).

Note that we store references to the actual **Line** or **Marker** objects, rather than their corresponding **LineInfo** or **MarkerInfo**.

Inherited attributes:

|  |  |  |
| --- | --- | --- |
| Java | XML | Type |
| guid | guid | string |

Attributes:

|  |  |  |
| --- | --- | --- |
| chromosome | chromosome | ChromosomeMap [ref] |
| line | line | Line [ref] |
| marker | marker | Marker [ref] |

**Bookmark** – BINARY

|  |  |
| --- | --- |
| INT | index within **DataSet** of referenced **ChromosomeMap** |
| INT | index within DataSet of referenced **Line** |
| INT | index within **ChromosomeMap** of referenced **Marker** |

Other misc comments:

database key information?

traits and qtl/features?

Unlike maps/lines the trait qtl data is common to the dataset itself and isn’t customizable per view (due to time constraints). This breaks (a bit) the concept of keeping raw data separate from the visualization wrappers around the raw data, as traits and features have to hold both data and visual information in one place. (actually the heat map phenotype stuff is?)