**Notes from Hiral Vora code review.**

**Topics:**

* Styles
* Parsers
* Factory
* Loading
* Actions
* Operators
* Quickload
* Graphs
* Server
* Selections
* Plugins

**Conventions**:

Assume all package names start with com.affymetrix. So when I talk about package “igb.view,” I mean com.affymetrix.igb.view.

**Bug**:

On Apple 10.9, Java 7, IGB/Netbeans crashes if computer is connected to external VGA monitor in the conference room.

**Netbeans**

Developing IGB, use Netbeans 7.2. You have to use Netbeans to modify Annotation tab and other parts of the GUI because we are using GUIBuilder. It’s fine to install the basic version.

**IGB Startup**

**Hiral Vora**

**12/12/2013**

The next section explains what happens when users selects a genome from the Start Screen, i.e., how data are loaded and what code is used.

When IGB launches, there are two tabs on top:

**Data Access** – bottom tab

**Current Genome** – side tab

The code for these tabs is located in igb.view package, which is core/igb/src.

These two tabs are internal to IGB in that their code resides within igb package. The code for other tabs (except SpliceView) are under plugins and therefore are considered to be external to IGB.

Most (if not all) IGB form-like components were implemented using the NetBeans GUIBuilder, which is one of the main reasons we are using NetBeans. If you need to make any changes to these components, you need to do it using the GUI Builder.

The code for the Current Genome tab resides in two classes:

SeqGroupViewGUI – contains GUI code

SeqGroupView – contains logic underlying the GUI code

And also in one file that should NEVER be edited:

SeqGroupViewGUI.form – an XML file

In general, wherever you see a file with “GUI” in the name, there is also a file with the same name but without the “GUI” suffix. You will also see a file with the same name as the GUI file but with extension .form.

Resize behavior:

For most components (except Annotation) there is no minimize size. The result is that the components look different when space is limited. The components contained in the “form” may get crammed together or look strange. To ensure that a tabbed panel never changes the size and location of its components, set the minimize size property in the GUIBuilder.

To edit a “GUI” file, open it in the editor in NetBeans and click Design.

**QUESTION**: SeqGroupViewGUI has some text that says “Custom Border at Runtime.” How does that work?

**ANSWER:**

**QUESTION**: How does the blue-underline link work? Where can I find the code that does that neat trick? If we wanted to add an icon to the border (e.g., a “home” icon) what would we try first?

**ANSWER:**

**Loading Data**

Code that manages loading data resides in igb.view.load.

The code for the Data Access panel is in GeneralLoadViewGUI.

SeqGroupViewGUI components include:

* A species menu that lists available species.
* A genome version menu that lists available genomes for that species.

genometryInpl.general.GenericVersion represents genome version

SeqGroupView is responsible for maintaining the mapping between species names (strings) and GenericVersion objects, which represent the genome versions that are available.

**Tip**: If you are looking for any kind of buggy behavior in the CurrentGenome tab, start at SeqGroupView and go from there.

Take note of “group” (an AnnotatedSeqGroup) and “gServer” (a GenericServer) arguments in the constructor – these are the key players here.

GenericServer – represents different types of servers, data providers (DAS1, QuickLoad (QL), local, DAS2)

Note: “local” load mode mostly uses QL code. So, from the point of view of a developer, there are actually only three types of servers, not four.

When the user selects a species, IGB then looks for GenericVersion objects representing different genome versions, which are (in the genomics world) called assemblies. The names of these different assemblies then get added to the genome version menu in the Data Access Panel.

Once a GenericVersion is selected, IGB looks for data sets that are available for that particular assembly. Within the code, these data sets correspond to GenericFeature objects, which then become listed in the Data Access Panel.

GenericFeature objects are in genometryImpl.general, same package where GenericVersion resides.

**Note**: GenericVersion.java RARELY changes; it’s for capturing information and contains few (no?) methods. Same goes with the GenericServer, it just stores data.

Look at the constructor for GenericFeature. Note how a GenericFeature object keeps a pointer to a GenericVersion object and also a genometryImpl.symloader.SymLoader object. This is important to enable to trace it back to where it came from.

Files are read and parsed by SymLoader objects. They are only required for QL or local files. They capture the details of different file formats and implement parsing methods.

This is called a “loader” in part to distinguish it from a parser package that resides in the DAS code. However, it’s actually a parser because it reads (parses) files and creates Symmetry objects, which represent genomic data like exons, gene models, and so on. Note that because DAS data sources have their own system for parsing data, a DAS GenericFeature will have no pointer to a SymLoader object.

When a GenericFeature is made, another part of the code figures out what the format of the file is based on its extension and then creates the correct SymLoader object for that particular file type.

**Some history:**

Objects used to read data files originally called “parsers” in IGB; they were in genometryImpl.parsers. This old system is still used by the DAS code. The DAS code, and also the “useq” code, was mainly developed by David Nix and coworkers at University of Utah. So although IGB is no longer using the parser package, we haven’t removed it because the DAS code depends on it and we didn’t want to risk breaking it that code. Eventually, the DAS2 code could be modified to enable it to use symloaders instead of the parsers, but not today.

This can be confusing for new developers because there appear to be two BED parsers, for example, when actually IGB is only using one – the one in the SymLoader package.

Open SymLoader in your editor.

Observe it contains a List called strategyList that lists methods for loading data; these correspond to the **Load Mode** menus in the **Data Access** tab.

Also, look at getChromosomeList. This code enables IGB to add new chromosomes to the Current Genome tab whenever a file is parsed that refers to chromosomes not currently known about in the current genome. This is very useful when users just have a file containing data and chromosome names because IGB will still be able to show the data.

**Tip**: If you want to create new loading strategy, you’ll need to add it here and ALSO add it to the corresponding Enum in LoadUtils.

As an example of how adding a new load mode strategy works, see Xam.java. (It has a strange name because it’s meant to handle BAM files and also SAM files, the ASCII equivalent of a BAM file.)

Currently there are three strategies recorded in SymLoader:

getGenome – load the entire file right away

getChromosome – load just the data for one chromosome. Note that we’ve removed this load mode strategy from the GUI, but it still exists in the code base because other code is using it.

getRegion – most often used, accepts a SeqSpan, captures where the user is looking , calls method “parse”

The parse methods always return a list of SeqSymmetries. SeqSymmetries are the fundamental data “unit” in IGB. They capture the start and end positions of genomic feature such as reads, gene models, and so on. They also capture properties and maintain links to children, e.g., exons that making a gene model.

view.load.GeneralLoadUtils – contains methods for loading.

**Tip**: If you need to look at anything related to loading, region and what sequence, start at GeneralLoadUtils.

The key method: view.load.GeneralLoadUtils.loadAndDisplaySpan – figures out what to do and what methods to use. Look here first and go from there.

Symmetries are related to actual data, what it is and where it’s located on the map. Symmetries contain information about properties, children, but don’t know anything about how they should be displayed. Information about appearance is contained in Glyph objects (part of the graphics system) and in TrackStyle objects.

How IGB manages colors and track appearance:

igb.tiers package.TrackStyle implements ITrackStyle

When Hiral started, there were many instances of style. Refactored to get rid of all of it except TrackStyle, captures all the colors and other visual style information, corresponds to selections user can make in the Annotation and Graph tabs.

IGB maintains preferences for individual data sets between sessions using Preferences.

If you want to add label foreground color, add a variable in this file. Start from here and then move forward to make the change in UI. Any changes to the look of a track, do it here.

Ex) RGB – related to individual symmetry, it’s a property of the symmetry object. The TrackStyle only captures what is common to the entire track.

**Tip**: You can remove “connected” attribute, should remove this from the documentation.

After the data are loaded and SeqSymmetry objects have been created, a factory object creates Glyphs.

igb.view.factories contains factory objects that manufacture Glyphs using SeqSymmetry and TrackStyle objects.

**Note**: Only DefaultTierGlyph is used ... The others are there from earlier efforts we didn’t move ahead with but ought to explore.

**QUESTION**: Explain MAX\_CHILD\_IN\_SLOP\_ROW in DefaultTierGlyph. Is this over-ridden in any other classes?

**ANSWER**:

One such useful experiment was an idea Hiral implement in which a user would click-drag tracks up and down in order to view more data. The other developers had some trouble to understand what he did – it was a totally new thing. We are leaving it in the codebase so that we can come back to it and experiment with how it worked. Ultimately, we want track-level zooming and scrolling. Look at these classes implemented by Hiral and go from there.

Also see: AnnotationTierGlyph, used other classes too, scattered around. The feature enabled individual track scrolling and zooming and resizing. Not snap based, unlike right not the track edges “snap” when the user tried to resize by click-dragging the edges of the label.

getChild – does the drawing

Glyphs that started with “Efficient” are optimized for IGB, then got moved to Genoviz because they don’t change.

Entire class used to create Glyphs and determine what type of Glyph, style of atrribute, color, height, etc.

Factories create Glyphs for TierGlyph, gives the Glyphs to the TierGlyph

A Symmetry defines a Glyph’s x value, but not its the y value.

The y value of a Glyph is defined by Packer objects.

A TierGlyph has a packer (more than one?)

Packers process Glyphs in order of their x values. Changing this would slow things down a lot, mainly because of selection and draw methods.

TierGlyph inherits from Glyph.

A TierGlyph has child Glyphs that are sorted in order of their x position. It needs them to be sorted this way so that it can quickly determine which Glyph’s draw methods to invoke based on the currently visible region, which is determined by the View object.

If you want to change in the y dimension, change the packer. But keep in mind that many parts of IGB heavily rely on the fact that everything is laid out using x values.

If you can let that go, you can come up with a new packer. But you’ll lose optimization related to selection. Rubber band selection would likely become slow.

Drawing would also probably become slow.

There are two types: Expand packer and collapse packer

Expand – you pack everything on top of each other, in stacks.

Collapes packer – you draw everything in one row, this is the simplest one, look at this one first to understand how they work.

Packers have “pack” method.

About package:

Genoviz SDK:

Don’t need to look into Genoviz...the only time it changes is when new things get added.

Genometry – in general, most of it is stable except for the parsers, but it rarely changes, only new things get added, such as operators – will talk about tomorrow then new type of filters, new parsers.

For example, if you developed a new bed format with a 15th field, THEN you would look at the parser to incorporate the change. Otherwise you don’t need to look at it. Only need an overieiw of what it does and how it does it.

**General Topics**

**Actions – genometryImpl.event.GenericAction (abstract)**

All the actions in IGB are derived from abstract class GenericAction, extended from AbstractAction, part of the JDK. Our GenericAction. Everything else is inside IGB, igb.action,

We use GenericAction to implement actions that can be done by a user. This allows buttons, icons, toolbar, tooltips. There are actions for every File menu and every menu item. So if you wanted to implement tooltips specifically for the menu or large or small icons, then we would implement it here.

Also, Actions associated with plugins would be in the respective plugin package. If an Action is not in Action, look in igb.shared.

Why we subclassed the base java action class: We wanted functionality not in the class. We wanted an ActionCallBackListener for the interactive tutorial function (under Help menu). Once you click on a button, you want to move focus to a new location. The tutorial needs this. It needs to know when an action has been completed. TutorialManager is a callback listener. It can fire events.

QUESTION: Is it feasible to support “undo” ?

ANSWER: Not exactly. See bookmarks which implements “undo” – has an inbuilt Undo.

**Operators**

Operators Located in genometryImpl.Operator or in plugins.

**Operator Interface:**

The most important method is “operate” – it’s the business logic. If you want to write a new operator, what it’s supposed to do should go into the method.

**Note**: The abstraction makes it is very easy to add a new Operator. This is a good place for new developer to begin.

The operate method accepts a BioSeq and list of SeqSymmetries.

Ex)

CopyGraphOperator

getOperandCountMax – you need to tell it the minimum and maximum number of SeqSymmetry objects.

The Operator controls what it gets.

Self-contained and isolated. If something breaks you just remove an operator.

See also getOutputCategory -- returns a FileTypeCategory – an ENUM in genometryImpl.parsers.FileTypeCategory

supportsTwoTracks – whether or not the operator’s output will have strand information

Transformers operators – what it means it will apply to one track only, so out don’t have to implement lots of methods.

AbstractAnnotationOperator – also overrides Operator

**Note**: All the abstract classes have “Abstract” in their name.

If you want to write an Operator look at the Abstract classes to find one that will work best.

**Colors**

Added various “color by” features in IGB 8, this is implemented in the color packages as colorproviders

Now have individual color provider classes

genometryImpl.color

See: class Colorprovider

Paired is another example to look at.

So if we want some new way to color Glyphs by SeqSymmetry properties, write a color by.

**Exercise**: How would you implement a way to color by based on a keyword search of descriptions from the gene models?

You would use: igb.colorproviders.Property

Note how within Property constructor, you can add new calles to addParameter. When this is done, the GUI that enables the user to choose parameters will automatically show them. So..for example, if you want to add anything to the UI, this will automatically generate them. The UI code that does this is in **WHERE**?

Note the GUI doesn’t use GUIBuilder – this is because it is too dynamic. GUIBuilder is good for creating forms, e.g., interfaces that don’t change.

**QUESTION**: How does the Track Operations GUI is work? It’s also dynamic.

See igb.shared.ConfigureOptionsPanel to modify the look of the GUI for ColorBy

Note: ColorComboBox – is a Jidesoft component

**QUESTION**: How do we incorporate a new “jar” release from Jidesoft or other provider?

**Filters**

genometryImpl.filter

All implement SymmetryFilterI (interface)

Ex) DuplicateFilter that enables filtering out, it just hides stuff.

Operators can create new stuff

**QUESTION**: How did Hiral ensure that users can’t select hidden items? He solved it by make changes to the packer. If a feature is hidden, the Packer ignores it. The change necessary chnge was to alter the selection code; if it’s hidden, don’t select it; the property of being hidden is part of the Glyph (isVisible or not).

**Note**: Filter by uses the same UI code as Color by...the same UI.

**Tip**: “Just learn one thing and you get the other thing for free.”

**QUESTION**: How do you try new things without risking disrupting the source code that we need to maintain for IGB 8?

**ANSWER**: IGB is a platform for trying out new ideas in genomic data visualization. However, you need to be smart about what you change. Put all of the new behaviors into a subclass that over-rides (changes) the behavior of the class you want to change. Then modify the part of the code that calls the parent class. That way, to change back to the old behavior, you make just a few changes to a few lines of code.

Example) Hiral noticed that IGB allows users to track only one track at a time. However, this is awkward when a user has many tracks visible at once. So he created a new class called MultiGlyphDragger which inherits from GlyphDragger. He then modified the TierLabelManager dragLabel method to use the new class.

QUESTION: What svn revisions show the change to TierLabelManager.dragLabel?

**Servers**

GenericServer in genometryImpl.general

See: ServerTypeI

Important methods are:

discoverFeature

getSpeciesAndVersions

loadFeature – loads data from all the servers into the display

getResidues – loads sequence data from whatever server is first (?)

**Tip**: Always look at the interface to see documentation. Implementing classes typically won’t contain documentation. If you want to understand what the methods are supposed to do, you need to look at the interface documentation.

Each server type has its own package. There are four hardcoded server types in ServerTypeI.

Quickload servers: When you launch IGB, it will start a separate thread for each QuickLoad server. Eventually each thread related to QuickLoad will reach loadGenomeNames (it’s synchronized) and will read the contents.txt file associated with the server.

**QUESTION**: Same thing for DAS servers?

A complication: No server will provide species names, which IGB needs in order to populate the species menu. Instead, IGB determines species names using genome names and synonym files. That’s why you need the string A\_thaliana in the species.txt file so that IGB will know what species a A\_thaliana\_Jun\_2009 genome belongs to.

Note: Most of the other names are from UCSC ones, e.g, mm7 and mm8. Different genome versions from the same species (e.g., mm for mouse) use the same prefix to indicate species and different number to indicate version.

It’s also required in synonyms.txt is actually for genome versions only. Gonna have A\_thaliana\_Jun\_2009 and a corresponding UCSC name. This is for the second combo box so that it knows they are same thing.

Two synonyms files related to genome versions: synonyms.txt and species.txt.

Synoyms.txt is used to determine which genome version names refer to the same assembly. IGB uses a value from this file to match species.txt synonyms. It just needs to be SOMEWHERE in that line of data.

When user selects a genome version, IGB reads annots.xml to determine what features data sets are available. This ONLY happens when a user selects a genome version. The method loadAnnotationsNames is called and then IGB reads genome.txt modChromInfo.txt

QuickLoadServerType – that’s what reads all these files.

Warning: Once a server loads a feature you can’t get rid of it.

**QUESTION**: How do we add a new attribute to file tag in annots.xml?

HV 12/13/13

**UI elements**

SeqMap – the scroller, coordinates box, pointer icon, zoomer, etc. SlicedView is another instance of a SeqMap, lacking some of the widgets.

SeqMap is in igb.view as SeqMapView

It’s a container that is supposed to hold any tool that manipulates data inside the AffyTieredMap, which is a component contained in the SeqMapView.

AffyTieredMap is relatively stable. It is using code from GenoViz and Genometry. Although it’s stable, it can’t be moved out because it’s using code from other libraries.

Whenever data model code is stable and related to data, it is moved to Genometry.

Genometry is treated as a library. A library is supposed to be stable.

Whenever code is stable and related to visualization, it is moved to Genoviz.

Genoviz is also considered a library.

AffyTieredMap located inside igb.tiers.

It’s an extension of a NeoMap, which is supposed to hold Glyphs inside it. Because it is optimized for IGB, it only accepts TieredGlyphs, with just a few exceptions, such as the Hairline (zoomstripe) Glyphs.

AffyTieredMap has a TierLabelGlyph.

The tracks are actually Tiers, which is a Glyph and is invisible. A TierLabelGlyph draws itself but it just draws a background.

TierGlyph calls draw on all its children. Later, View or NeoCanvas will “trim it out” and avoid calling “draw” on children that are not visible.

Hiral’s comment: “It just works.”

Can initialize the background color of the NeoMap in SeqMapView.

In addition to the AffyTieredMap, there is a toplevel instance that holds both of them together. It has a TierLabel.

Under igb.tiers there is AffyLabeledTierMap. There are two instances of these. One map contains the labels, the other map holds the tier.

DON’T look here to fix errors where tracks and track labels get out of sync. That requires getting into the Genoviz SDK code. Uses a lot of math, does a lot of calculation and what to draw and how to draw it.

The toolbar is IGBToolBar, an extension from JDK to enable drag and drop of toolbar icons.

NOTE: you can drag and drop toolbar icons.

Available Data is in FeatureTreeView inside igb.view.load

There’s a section where you click on the “I” to get a web browser to open. See: “renderFeature” – note there is HTML, but cause some problems with spaces. Most of the time if works but sometimes not.

When users selects a GenericViewer, it accepts that and gets a list of features that are available. Only calls code within itself or from the other classes in the same package. Also handles clicking on the server URL itself.

Connected to Data management table. There has to be interaction so that the table and the tree view do not get out of synch.

LoadView and GeneralLoadView call into the tree, but the tree itself doesn’t call anything outsides itself, usually.

Data Management Table – located in the same package view.load.DataManagementTable

The DMT is connected by style.

**QUESTION**: How would you make the Track Name centered instead of right-justified?

**ANSWER**: You would start at DataManagementTable, which holds a JTable. You’d create or use a different cell rendered object. It might already exist in Genoviz SDK or in other general loation in IGB. Genoviz SDK – in genoviz.swing or genoviz.swing.recordpayback

If you write your own, you could move it to Genoviz or Genometry, but only once it has been used in a couple of different places and is robust. If not, then make it an internal class. You could have more than one renderer for a cell depending on how it’s being displayed.

**New topic: Graphs**

Graph: A type of a SeqSymmetry, it contains everything needed to manipulate or perform calculations on the graph. And there are extensions of these. They are all graphs and use GraphSym is the fundamental data object. It’s different from the rest of the SeqSymmetries.

genometryImpl.symmetry.GraphSym extends RootSeqSymmetry, a bit different from other symmetry objects

Q: Why genometryImpl – it’s a weird name?

A: Changing it to genomtry could cause problems and conflicts with DAS because DAS has genometry package.

GraphGlyph is also very different. It contains an instance of GraphSym and heavily relies on the GraphSym.

In the general glyphs, data from the seqsymmetry gets propagated to the Glyph upon construction and that’s the last time it’s needed.

However, even though a graph factory makes a graph, it doesn’t get propagated. Instead, it puts an instance inside (?) and creates a strong bond. But in this case it’s OK.

Graph doesn’t have a plus or minus, no expand or collapse, most of the Style properties were designed for annotations. So there is a GraphState and holds state of a graph and uses.

GraphState is in genometryImpl.style

GraphState has a style

A lot of math in GraphGlyph.

AffyTieredMap accepts only a tier apart from tier, but it also holds an invisible glyph that holds a graph.

Graph floating:

Lance and Hiral tried to generalize the floating aspects of moving graphs but were not able to generalize it to work for annotations as well.

Color by and filter by were only possible because he generalized it.

Something to keep in mind: don’t develop anything where the learning curve is too high, and when you change it, users get upset. Always try to mimic an application that already exists.

There are different types of GraphGlyphs for different types or styles of graphs. See igb.graphTypes

Abstract class called GraphStyle. There is ONE GraphGlyph (and a couple of subclasses) contains a GraphyStyle and if you want to make a new type of look for graphs, you could make a new GraphStyle class, which would then be “on the menu” so to speak as an option for what a graph could look like for a user.

However, we would need to change the UI ... from radio buttons to a menu or something like that to enable addition of more styles.

Mismatch you can apply different styles to.

MismatchPileup, has one x value and multiple y levels (A,T,G,C, and “other”) Looks like a bar type.

Whatever new graph type won’t apply

**MISC QUESTIONS**

**Q**. What do we do when we need to incorporate a new “cert” ?

**Q**: I want to use a new icon for alert messages in IGB. Currently, the file is called igb.gif. How would I change it to something else?

**A**: The current file igb.gif is under version control in core/common/resources/images. To change to a new file, modify common.CommonUtils.getApplicationIcon