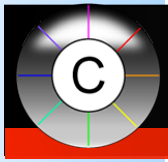


Cytoscape 2.x Developer's Tutorial



Outline

- Concepts
- API Layers
- Working with:
  - *Graph model (networks, nodes, edges)*
  - *View*
  - *Attributes*
- Break
- Q&A: Possible topics
  - *Tunables*
  - *Layouts*
  - *Groups*
  - *Vizmap*
  - *“Sequence of events”*

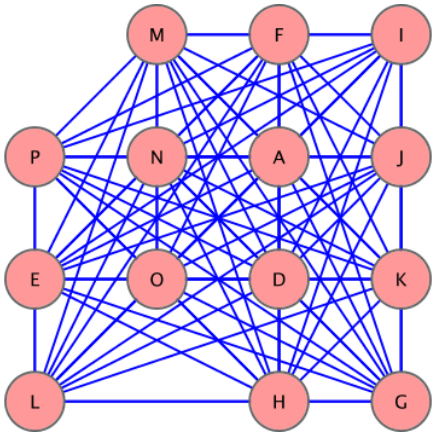
Cytoscape 2.8 Layers

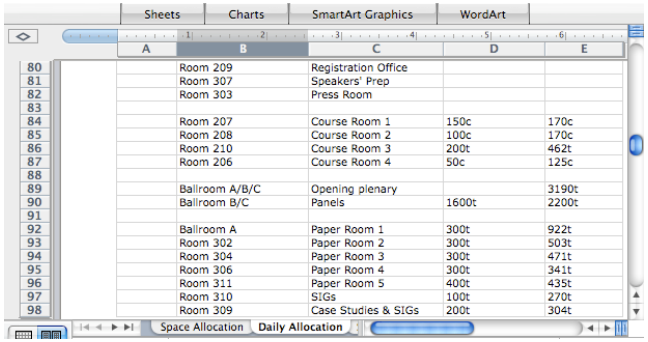
or

What's a "root graph" and why do I care?

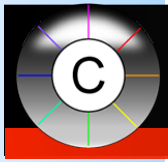
Layers: Concepts

- **Graphs and Data**





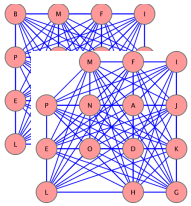
Graphs (CyNetworks) ← VizMapper → Data (CyAttributes)

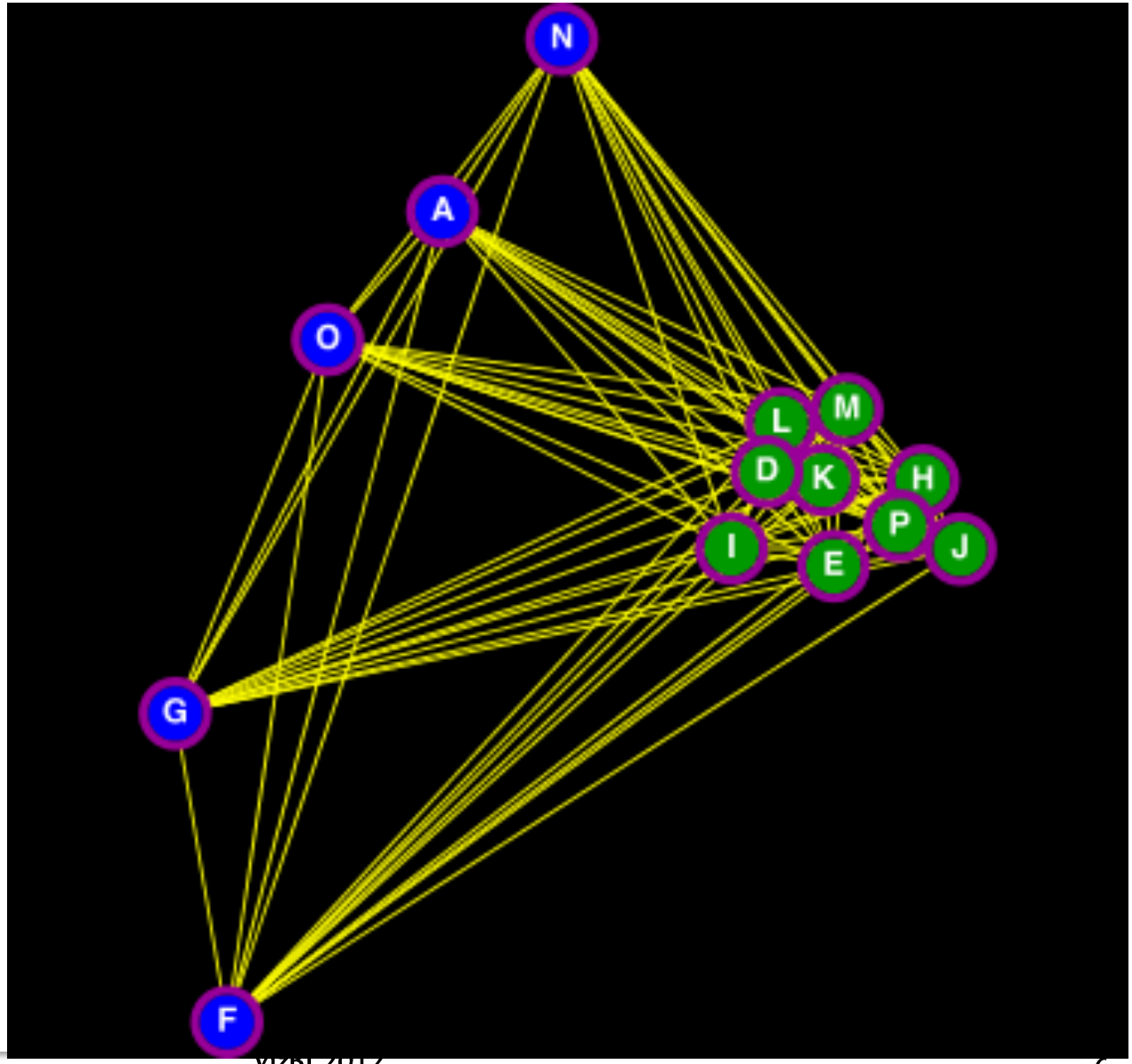


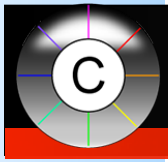
Layers: Concepts

- **Graphs and Data**
  - ***Graphs and Networks***
    - Root Graph: ONE graph to rule them all....
      - All nodes and edges are part of the root graph
    - A network is a subset (projection) of the root graph
    - A network view is a visualization of a network
    - The Vizmapper maps attributes values to visual presentation
  - ***Attributes***
    - Typed values that can be associated with CyNetworks, CyNodes, and CyEdges

Graph Concepts

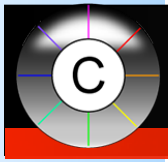






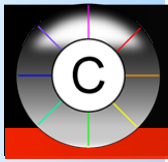
Attributes

- ***Attributes are not really "attributes"***
  - ***Attributes are data values that might be associated with networks, nodes, and edges***
    - You can read in attributes for nodes/edges you haven't created yet
    - Attributes are associated to nodes/edges by identifier (name)
  - ***Attributes are typed***
    - Support for Integers, Doubles, Booleans, Strings, LIsts, and Maps (dictionaries)
    - Attributes are accessed by type, attribute name, object identifier (name)



Root Graph Implications

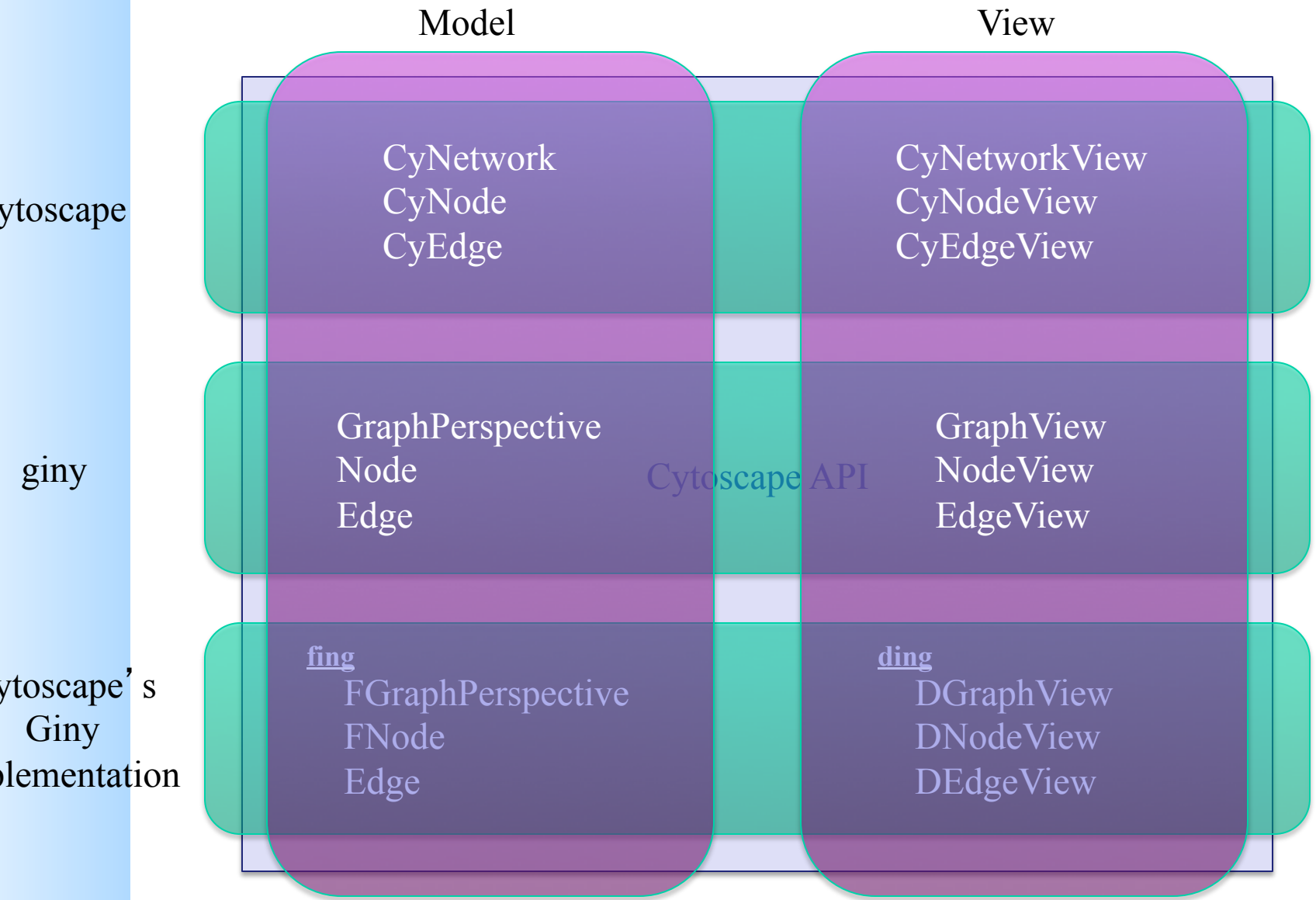
- **Two nodes with the same identifier in different networks are the same node**
  - ***They will have the same attributes***
  - ***VizMap overrides (which are implemented as attributes) will effect all presentations of the node***
- **Groups exist across all networks**

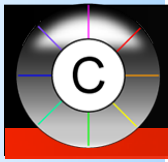


Layers

- **Graphs, Views, and Data (Attributes)**
  - ***Views depend on Graphs***
  - ***Data doesn't depend on anything***
- **Graph**
  - ***CyNetwork -> GraphPerspective (giny) -> FGraphPerspective (fing)***
  - ***CyNode -> Node (giny) -> FNode (fing)***
  - ***CyEdge -> Edge (giny) -> FEdge (fing)***
- **View**
  - ***CyNetworkView -> GraphView (giny) -> DGraphView (ding)***
  - ***CyNodeView -> NodeView (giny) -> DNodeView (ding)***
  - ***CyEdgeView -> EdgeView (giny) -> DEdgeView (ding)***
- **Data**
  - ***CyAttributes***

Layers


Model
View
Cytoscape
giny
Cytoscape's Giny Implementation
CyNetwork
CyNode
CyEdge
CyNetworkView
CyNodeView
CyEdgeView
GraphPerspective
Node
Edge
Cytoscape API
GraphView
NodeView
EdgeView
fing
FGraphPerspective
FNode
Edge
ding
DGraphView
DNodeView
DEdgeView

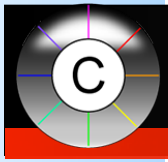


API

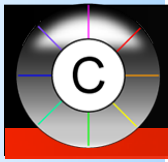
- **Navigating the Cytoscape API**
  - ***Start with the* Cytoscape *class***
    - Singleton class
    - Starting point for most Cytoscape functionality
  - ***Other useful singletons***
    - CyGroupManager: group-related information
    - CyLayouts: information about currently loaded layout algorithms

API

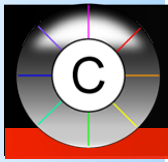
- ***It's not all in cytoscape…***
  - Cytoscape has four different groups of functionality:
    - cytoscape/: the core application
    - corelibs/: corelibs are parts of Cytoscape that are called by core routines, but never call any core routines. Much of the graph implementation is here.
    - coreplugins/: a significant piece of the Cytoscape functionality is "hidden" in the plugins that are delivered with cytoscape. As with any plugin, these are not directly available through the API. Including:
      - » Layouts
      - » Data browser
      - » Editor
      - » Table import, SBML reader, and PSI-MI readers
      - » Filters
    - csplugins/: User-developed plugins add significant functionality. Many of those are available as source



Questions?

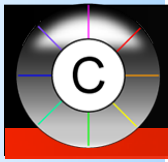


Working with networks, nodes and edges



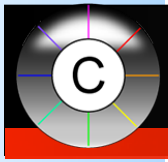
Creating nodes, edges, and networks

- `Cytoscape.getCyNode(name)`
- `Cytoscape.getCyEdge(source,name,target,interaction)`
- `Cytoscape.createNetwork(nodes, edges, name)`
- `Cytoscape.createNetwork(name)`
- `Cytoscape.setCurrentNetwork(name)`



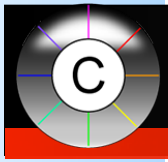
Getting networks, nodes, and edges

- `Cytoscape.getCurrentNetwork()`
- `Cytoscape.getNetwork(name)`
- `cyNetwork.nodesList()`
  - *Note, this returns a list of Nodes, not CyNodes. They can be safely cast*
- `cyNetwork.edgesList()`
  - *Note, this returns a list of Edges, not CyEdges. They can be safely cast*
- `cyNode.getRootGraphIndex()`
- `cyEdge.getRootGraphIndex()`
- `cyNetwork.getAdjacentEdgeIndicesArray(int,true,true,true)`
  - *Note, this takes root graph indices and returns and array of root graph indices*
- `cyNetwork.getConnectingEdges(list)`

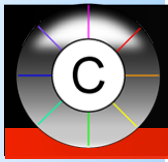


Selection

- cyNetwork.setSelectedEdgeState(edge, boolean)
- cyNetwork.setSelectedEdgeState(list, boolean)
- cyNetwork.setSelectedNodeState(node, boolean)
- cyNetwork.setSelectedNodeState(list, boolean)
- cyNetwork.getSelectedEdges()
- cyNetwork.getSelectedNodes()
- cyNetwork.isSelected(edge)
- cyNetwork.isSelected(node)

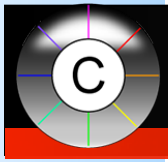


Working with views



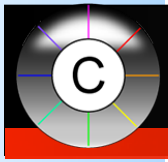
Creating a network view

- `Cytoscape.createNetworkView(network)`
- `Cytoscape.createNetworkView(network, title, layout)`



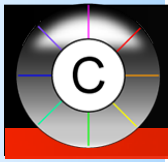
Getting network views

- `Cytoscape.getCurrentNetworkView()`
- `Cytoscape.getNetworkView(name)`



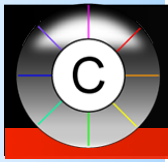
Getting edge and node views

- `cyNetworkView.getNodeView(node)`
- `cyNetworkView.getEdgeView(edge)`



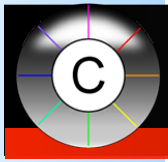
Changing appearance

- **Moving nodes**
  - `nodeView.setXPosition(double)`
  - `nodeView.setYPosition(double)`
- **Hiding nodes**
  - `cyNetwork.hideNode(node)`
  - `cyNetwork.hideNodes(list)`
  - `cyNetwork.hideEdge(edge)`
  - `cyNetwork.hideEdges(list)`
- **Reflecting changes**
  - `cyNetworkView.updateView()`



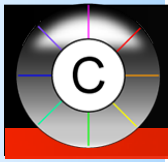
Working with attributes

- ***Think of attributes as tables***
  - **3 tables of attributes**
    - `Cytoscape.getNetworkAttributes()`
    - `Cytoscape.getNodeAttributes()`
    - `Cytoscape.getEdgeAttributes()`



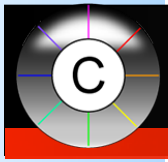
Attribute Types

- TYPE_BOOLEAN (Boolean)
- TYPE_FLOATING (Double)
- TYPE_INTEGER (Integer)
- TYPE_SIMPLE_LIST (List)
- TYPE_SIMPLE_MAP (Map)
- TYPE_STRING (String)



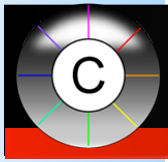
Setting attributes

- `cyAttributes.setAttribute(objectName,attrName,value)`
- `cyAttributes.setListAttribute(objectName,attrName,list)`
- `cyAttributes.setMapAttribute(objectName,attrName,map)`



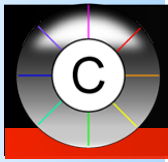
Getting attributes

- cyAttributes.hasAttribute(objectName,attrName)
- cyAttributes.getBooleanAttribute(objectName,attrName)
- cyAttributes.getDoubleAttribute(objectName,attrName)
- cyAttributes.getIntegerAttribute(objectName,attrName)
- cyAttributes.getListAttribute(objectName,attrName)
- cyAttributes.getMapAttribute(objectName,attrName)



Setting attribute characteristics

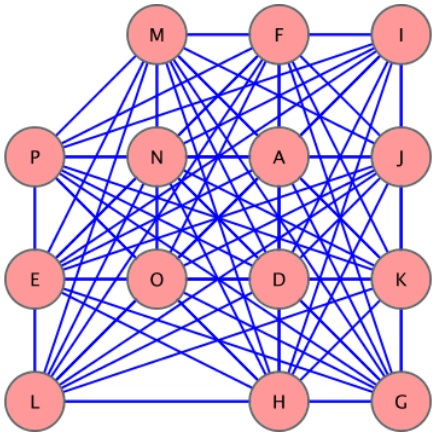
- `cyAttributes.setUserEditable(name,boolean)`
- `cyAttributes.setUserVisible(name,boolean)`

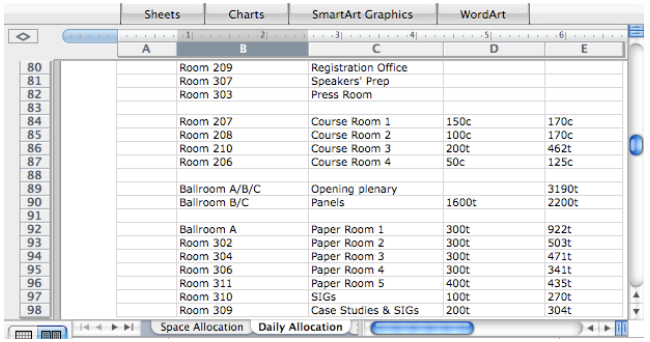


Break

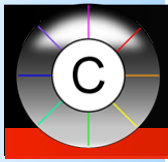
VizMapper

- **Remember central idea:**





Graphs (CyNetworks) ← VizMapper — Data (CyAttributes)



VisualMappingManager

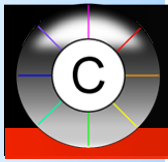
```
// get the network and view
   CyNetwork network = Cytoscape.getCurrentNetwork();
   CyNetworkView networkView = Cytoscape.getCurrentNetworkView();

   // get the VisualMappingManager and CalculatorCatalog
   VisualMappingManager manager = Cytoscape.getVisualMappingManager();
   CalculatorCatalog catalog = manager.getCalculatorCatalog();

   // check to see if a visual style with this name already exists
   VisualStyle vs = catalog.getVisualStyle(vsName);
   if (vs == null) {
     // if not, create it and add it to the catalog
     vs = createVisualStyle(network);
     catalog.addVisualStyle(vs);
   }

   networkView.setVisualStyle(vs.getName()); // not strictly necessary

   // actually apply the visual style
   manager.setVisualStyle(vs);
   networkView.redrawGraph(true,true);
```

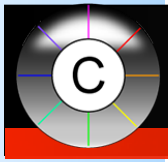


VisualStyles

```
VisualStyle createVisualStyle(CyNetwork network) {

   // A Style contains three appearance calculators: Node, Edge, and Global
   NodeAppearanceCalculator nodeAppCalc = new NodeAppearanceCalculator();
   EdgeAppearanceCalculator edgeAppCalc = new EdgeAppearanceCalculator();
   GlobalAppearanceCalculator globalAppCalc = new GlobalAppearanceCalculator();

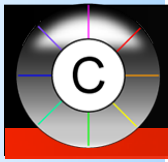

   // Passthrough Mapping - set node label
   PassThroughMapping pm = new PassThroughMapping(new String(), "attr2");
   Calculator nlc = new BasicCalculator("Example Node Label Calculator",
                                        pm, VisualPropertyType.NODE_LABEL);
   nodeAppCalc.setCalculator(nlc);
```



VisualStyles

```
// Discrete Mapping - set node shapes
DiscreteMapping disMapping = new DiscreteMapping(NodeShape.RECT,
                                                 ObjectMapping.NODE_MAPPING);
disMapping.setControllingAttributeName("attr1", network, false);
disMapping.putMapValue(new Integer(1), NodeShape.DIAMOND);
disMapping.putMapValue(new Integer(2), NodeShape.ELLIPSE);
disMapping.putMapValue(new Integer(3), NodeShape.TRIANGLE);

Calculator shapeCalculator = new BasicCalculator("Example Node Shape Calculator",
                                                 disMapping,
                            VisualPropertyType.NODE_SHAPE);
nodeAppCalc.setCalculator(shapeCalculator);
```



VisualStyles

```
// Continuous Mapping - set node color
   ContinuousMapping continuousMapping = new ContinuousMapping(Color.WHITE,
                                                               ObjectMapping.NODE_MAPPING);
   continuousMapping.setControllingAttributeName("attr3", network, false);

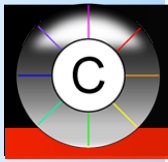
       Interpolator numToColor = new LinearNumberToColorInterpolator();
       continuousMapping.setInterpolator(numToColor);

   Color underColor = Color.GRAY;
   Color minColor = Color.RED;
   Color midColor = Color.WHITE;
   Color maxColor = Color.GREEN;
   Color overColor = Color.BLUE;

   // Create boundary conditions                    less than,   equals,  greater than
   BoundaryRangeValues bv0 = new BoundaryRangeValues(underColor, minColor, minColor);
   BoundaryRangeValues bv1 = new BoundaryRangeValues(midColor, midColor, midColor);
   BoundaryRangeValues bv2 = new BoundaryRangeValues(maxColor, maxColor, overColor);

   // Set the attribute point values associated with the boundary values
   continuousMapping.addPoint(0.0, bv0);
   continuousMapping.addPoint(1.0, bv1);
   continuousMapping.addPoint(2.0, bv2);

   Calculator nodeColorCalculator = new BasicCalculator("Example Node Color Calc",
                                                        continuousMapping,
                            VisualPropertyType.NODE_FILL_COLOR);
   nodeAppCalc.setCalculator(nodeColorCalculator);
```

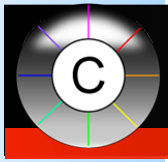


VisualStyles

```
// Discrete Mapping - Set edge target arrow shape
  DiscreteMapping arrowMapping = new DiscreteMapping(ArrowShape.NONE,
                                                     ObjectMapping.EDGE_MAPPING);
  arrowMapping.setControllingAttributeName("interaction", network, false);
  arrowMapping.putMapValue("pp", ArrowShape.ARROW);
  arrowMapping.putMapValue("pd", ArrowShape.CIRCLE);

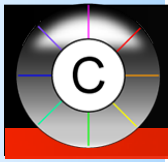
  Calculator edgeArrowCalculator = new BasicCalculator("Example Edge Arrow Shape
  Calculator",
                                                arrowMapping,
                                                VisualPropertyType.EDGE_TGTARROW_SHAPE);
  edgeAppCalc.setCalculator(edgeArrowCalculator);


  // Create the visual style
  VisualStyle visualStyle = new VisualStyle(vsName, nodeAppCalc, edgeAppCalc,
                                            globalAppCalc);
  return visualStyle;
}
```



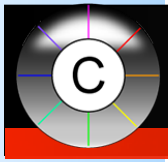
VisualStyles

- **Summary:**
  - ***Get the VisualMappingManager***
  - ***Get the Catalog***
  - ***Create your style***
    - Create Node, Edge, and Global AppearanceCalculators
      - Construct a calculator for each VisualProperty
        - » Create a PassThrough, Discrete, or Continuous mapper that maps an attribute to a VisualProperty
      - Set the calculator in the AppearanceCalculator
    - Create the style with all three AppearanceCalculators
  - ***Set your style in the NetworkView***
  - ***Add your style to the catalog***
  - ***Apply the style: applyStyle or redrawGraph***



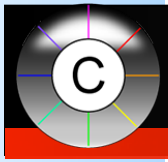
Tunables

- **Tunables**
  - ***Serves three purposes:***
    - **Persistence of values**
    - **Automatic UI generation**
    - **External access to plugin parameters**
  - ***Tunables are typed***



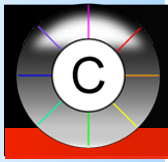
Tunables

- **Tunable objects take 7 arguments**
  - ***name:*** *the name of the tunable*
  - ***description:*** *the description of the tunable*
  - ***type:*** *the tunable type*
  - ***value****: the initial value of the tunable*
  - ***lowerBound****: the lower bound of the tunable*
  - ***upperBound****: the upper bound of the tunable*
  - ***flag****: flags to provide hints to the UI generators*



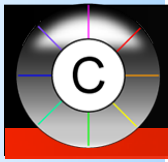
Tunable Types

- **Types**
  - ***INTEGER***
  - ***DOUBLE***
  - ***BOOLEAN***
  - ***STRING***
  - ***NODEATTRIBUTE***
  - ***EDGEATTRIBUTE***
  - ***LIST***
  - ***GROUP***
  - ***BUTTON***



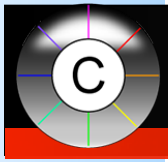
Tunable Bounds and Flags

- **Bounds and flags**
  - ***Bounds provide:***
    - upper/lower bounds of numeric quantities
    - The list for a List
    - An initial list for attributes (to allow alternative choices)
  - ***Flags provide hints for the UI***
    - MULTISELECT to allow multiple selection in a list
    - USESLIDER to use a slider for numeric values (must also have bounds)
    - IMMUTABLE to display the value but not allow editing
    - COLLAPSABLE to implement collapsible groups
    - NOINPUT for internal parameters that aren't presented to the user
    - NUMERICATTRIBUTE to restrict attribute presentation to numeric values



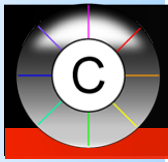
Tunable Interaction

- **Interaction**
  - ***BUTTON tunable***
    - ***Lower bound is an ActionListener***
    - ***Called when the button is pressed***
  - ***A TunableValueListener***
    - ***Can be added to any Tunable***
    - ***Caller will be notified when a tunable value changes***



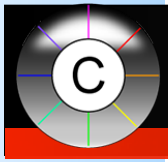
LayoutProperties

- **Two functions**
  - ***Collect all Tunables into a list***
  - ***Generate a single JPanel with all of the Tunable interfaces***
    - **LayoutProperties handles GROUPing**
- **Example: tunableSampler**



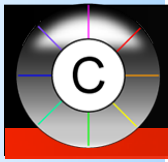
Layouts

- **Layouts should all inherit from AbstractLayout**
- **Registering layouts**
  - ***CyLayout.addLayout(algorithm, menu)***
  - ***Information about capabilities are determined through callbacks***
  - ***Registered layouts can be "found" by plugins by calling CyLayout.getLayout(name)***
- **Cytoscape's layout menu takes care of tasks**
  - ***Use LayoutTask if you want to call a layout yourself on a separate thread:***
    - LayoutTask(layout, view)
    - TaskManager.executeTask(LayoutTask, LayoutTask.getDefaultTaskConfig())



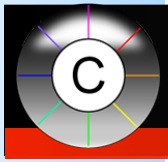
Layouts

- **AbstractLayout fields:**
  - ***Set<NodeView> staticNodes:***
    - The Set of nodes that should not be moved
  - ***CyNetworkView networkView:***
    - The view this layout should operate on
  - ***CyNetwork network:***
    - The network this layout should operate on
  - ***boolean selectedOnly:***
    - True if only selected (unlocked) nodes should be laid out
  - ***String edgeAttribute:***
    - The edge attribute to use for the layout
  - ***String nodeAttribute:***
    - The node attribute to use for the layout
  - ***boolean canceled:***
    - True if the user requested that the layout be aborted
  - ***TaskMonitor taskMonitor:***
    - The TaskMonitor used for the layout



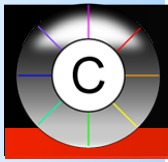
Layouts

- **Layout callbacks:**
  - ***String getName():***
    - Returns the short name for this layout
    - Name used to get this layout from CyLayouts
  - ***String toString():***
    - The readable name for this layout
    - Used for menus
  - ***boolean supportsSelectedOnly():***
    - True if this layout will support only operating on a subset of the nodes
  - ***byte[] supportsNodeAttributes():***
    - An array of supported attribute type for the layout (or null)
  - ***byte[] supportsEdgeAttributes():***
    - An array of supported attribute type for the layout (or null)
  - ***List<String> getInitialAttributeList():***
    - A list of values to add to the list of attributes
    - Allows for an "unweighted" value or any special list values to be added to the menu presented to the user



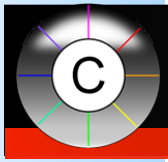
Layouts

- **Layout callbacks:**
  - ***LayoutProperties getSettings():***
    - Return your list of Tunables
  - ***updateSettings():***
    - Update all of the Tunables values
  - ***revertSettings():***
    - Revert all Tunables values to the default
  - ***JPanel getSettingsPanel():***
    - Return the settings panel for this layout
    - Return from LayoutProperties.getTunablePanel()



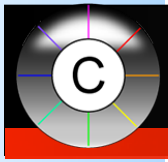
Layouts

- **Using an existing layout**
  - ***Get the CyLayoutAlgorithm:***
    - CyLayoutAlgorithm alg = CyLayouts.getLayout(String);
  - ***Get the Tunables:***
    - LayoutProperties props = alg.getSettings();
  - ***Set the Tunables you want:***
    - Tunable t1 = props.get(String);
    - t1.setValue(Object);
    - alg.updateSettings();
  - ***Set appropriate flags (will also be Tunables):***
    - alg.setSelectedOnly(boolean);
    - alg.setLayoutAttribute(String);
  - ***Do the layout:***
    - alg.doLayout(CyNetworkView, TaskMonitor);

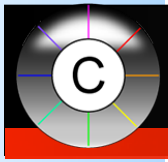


Layouts

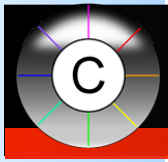
- **Using an existing layout**
  - ***Get the CyLayoutAlgorithm:***
    - CyLayoutAlgorithm alg = CyLayouts.getLayout(String);
  - ***Get the Tunables:***
    - LayoutProperties props = alg.getSettings();
  - ***Set the Tunables you want:***
    - Tunable t1 = props.get(String);
    - t1.setValue(Object);
    - alg.updateSettings();
  - ***Set appropriate flags (will also be Tunables):***
    - alg.setSelectedOnly(boolean);
    - alg.setLayoutAttribute(String);
  - ***Do the layout:***
    - alg.doLayout(CyNetworkView, TaskMonitor);



Questions?

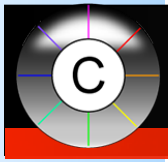


Groups



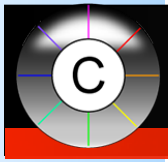
Group concepts

- **A group is a collection of nodes and their edges**
- **A node may be defined in multiple groups**
- **A group is represented by a node, which might itself be part of a group**
  - ***Groups are hierarchical***



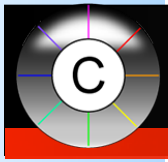
Creating groups

- `CyGroupManager.createGroup(name,list, viewer)`



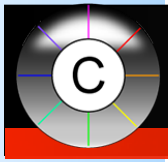
Using groups

- `cyGroup.addNode(node)`
- `cyGroup.addInnerEdge(edge)`
- `cyGroup.addOuterEdge(edge)`
- `cyGroup.getNodes()`
- `cyGroup.getInnerEdges()`
- `cyGroup.getOuterEdges()`
- `cyGroup.getGroupNode()`
- `cyGroup.getState()`
- `cyGroup.removeNode()`
- `cyGroup.setState(int)`
- `cyNode.inGroup(group)`
- `cyNode.getGroups()`
- `cyNode.isaGroup()`



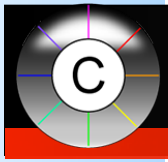
GroupViewers

- **Group is visualized by a CyGroupViewer**
- **The CyGroupViewer for a group can be changed**
- **Common usage for creating lots of groups:**
  - ***Create group with no viewer***
  - ***Set viewer:***
    - `CyGroupManager.setGroupViewer(group, viewer, view, false)`
  - ***On last group:***
    - `CyGroupManager.setGroupViewer(group, viewer, view, true)`



Putting it all together: plugins

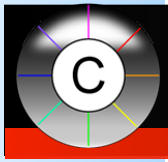
- **A class in your plugin should extend CytoscapePlugin**
  - *The constructor for this class will be called when your plugin is loaded*
- **Store your jar in {cytoscape}/plugins**
- **Plugin information is provided by a property file**
  - *Usually in resources/plugins.props*



Layout plugin

```
public class SampleLayout extends CytoscapePlugin {

  /**
   * The constructor does very little.  All we need to do is to register
   * our layout algorithm.  The CyLayouts mechanism will worry about how
   * to get it in the right menu, etc.
   */
  public SampleLayout() {
    CyLayouts.addLayout(new SampleLayoutAlgorithm(), "Scooter's Layouts");
  }
}
```

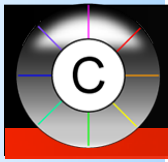


Layout plugin

```
public class NamedSelection extends CytoscapePlugin
                            implements CyGroupViewer,
                                       NodeContextMenuListener,
                                       PropertyChangeListener {

  /**
   * Create our action and add it to the plugins menu
   */
  public NamedSelection() {
    // Listen for network changes (so we can add our context menu)
    // Now that the group panel supports group creation, do we still
    // want to do this?
    try {
      // Listen for network and session load events
      Cytoscape.getPropertyChangeSupport().addPropertyChangeListener(
                                                    Cytoscape.NETWORK_LOADED, this);
      Cytoscape.getPropertyChangeSupport().addPropertyChangeListener(
                                                    Cytoscape.SESSION_LOADED, this);
    } catch (ClassCastException e) {
      myLogger.error(e.getMessage());
    }
    // Create our main plugin menu
    JMenu menu = new JMenu("Named Selection Tool");
    menu.addMenuListener(new NamedSelectionMenuListener(null));

    JMenu pluginMenu = Cytoscape.getDesktop().getCyMenus().getMenuBar().getMenu("Plugins");
    pluginMenu.add(menu);
```

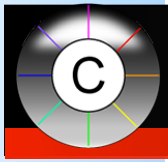


Group Viewer plugin

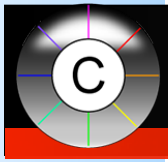
```
    // Add our interface to CytoPanel 1
    groupPanel = new GroupPanel(this);
    Cytoscape.getDesktop().getCytoPanel(SwingConstants.WEST).add("Groups", groupPanel);
    // We want to listen for graph perspective changes (primarily SELECT/UNSELECT)
    Cytoscape.getCurrentNetworkView().addGraphViewChangeListener(groupPanel);

    // Register with CyGroup
    CyGroupManager.registerGroupViewer(this);
    myLogger = CyLogger.getLogger(NamedSelection.class);
    myLogger.info("namedSelectionPlugin "+VERSION+" initialized");
    networkList = new ArrayList();

    // Add the currently loaded network
    networkList.add(Cytoscape.getCurrentNetwork());
  }
}
```

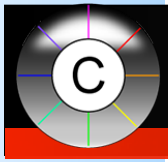


Questions?



CyCommands

- In Cytoscape 2.x, it's very difficult to share code between plugins
- CyCommands provide a workaround until we get to Cytoscape 3's OSGi-based mechanism



CyCommands

- Basic concepts:
  - *Namespace: a reserved string that only your plugin may use*
    - Avoids mixtures of comments
  - *Command: a string that refers to specific functionality your plugin will implement*
  - *CyCommandManager: a singleton class that is used for registering namespaces and commands, and for executing commands*

```java
import java.util.Collection;
import java.util.Map;

import cytoscape.command.AbstractCommandHandler;
import cytoscape.command.CyCommandException;
import cytoscape.command.CyCommandNamespace;
import cytoscape.command.CyCommandResult;
import cytoscape.layout.Tunable;


public class MyCommandHandler extends AbstractCommandHandler {
   // Define our command name
   private static String COMMAND = "my command";

   // Settings
   private static String ARGKEY1 = "argkey1";
   private static String ARGKEY2 = "argkey2";
   private static String ARGKEY3 = "argkey3";

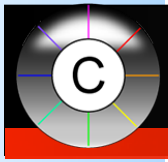
   public MyCommandHandler(CyCommandNamespace ns) {
     super(ns);
     addDescription(COMMAND, "This command does something really cool");
     addArgument(COMMAND, ARGKEY1);
     addArgument(COMMAND, ARGKEY2);
     addArgument(COMMAND, ARGKEY3, "defaultValue");

     // Add other commands here...
   }


   public CyCommandResult execute(String command, Map<String, Object>args) throws CyCommandException {
    // Either add your code here, or call
    // return execute(command, createTunableCollection(args));
    // and implement your code in the "Tunable" method below

   }

   public CyCommandResult execute(String command, Collection<Tunable>args) throws CyCommandException {
    // Your execution code goes here, or call
    // return execute(command, createKVMap(args));
    // and implement your code in the Map method above
   }
 }
```



CyCommands

```
import java.util.HashMap;
import java.util.Map;
import cytoscape.command.CyCommandException;
import cytoscape.command.CyCommandManager;
import cytoscape.command.CyCommandResult;

public class MyCode {

  public MyCode () {
    // Set up args
    Map<String, Object> args = new HashMap<String, Object>();
    args.put("arg1", "value");
    args.put("arg2", "value");
    // etc...

    try {
      CyCommandResult result = CyCommandHandler.execute("namespace", "my command", args);
      // result will have all of the values returned (if any) and any messages or errors
    } catch (CyCommandException e) {
      // Handle the exception
    }
  }
}
```

```
public class NodeCharts {

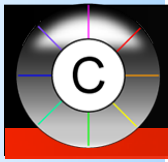
  private static String VALUELIST = "valuelist";
  private static String LABELLIST = "labellist";

  public static void updateNodeCharts(MetaNode mn, CyLogger logger) {

    // Create our argument map
       Map<String, Object> args = new HashMap<String, Object>();
       args.put("node", group.getGroupNode().getIdentifier());
       args.put("colorlist", "contrasting");

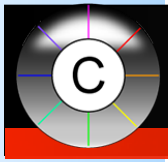
    // Depending on the attribute type -- do the right thing
       switch (attributeType) {
       case CyAttributes.TYPE_SIMPLE_LIST:
         getHistogramLabelsAndValues(args, group, nodeAttributes, nodeChartAttribute, chartType);
         break;
       case CyAttributes.TYPE_INTEGER:
       case CyAttributes.TYPE_FLOATING:
         getNumericLabelsAndValues(args, group, nodeAttributes, nodeChartAttribute, chartType);
         break;
       case CyAttributes.TYPE_BOOLEAN:
       case CyAttributes.TYPE_STRING:
         getStringLabelsAndValues(args, group, nodeAttributes, nodeChartAttribute, chartType);
         break;
       default:
       }

       try {
           // First, make sure we don't have any nodechart on the node
           CyCommandResult result = CyCommandManager.execute("nodecharts", "clear", args);
           // Now, add the node chart
           result = CyCommandManager.execute("nodecharts", chartType, args);
       } catch (CyCommandException cce) {
           logger.warning("node chart command failed: "+cce.getMessage());
       }
  }
```

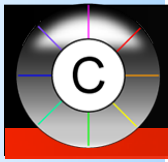


CyCommands

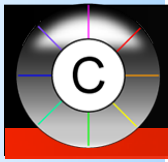
- Basic approach
  - *exporting commands*
  - *using commands*
  - *example (from the metaNodePlugin2)*
- Tools
  - *coreCommands*
    - A set of commands implemented in the Cytoscape core
  - *commandTool*
    - A plugin that provides a command-line interface to available commands



Questions?

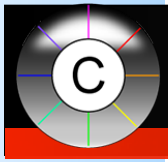


Cytoscape 3.0



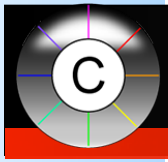
Cytoscape 3.0

- **Significant update from Cytoscape 2.x**
  - ***No more giny layer***
  - ***Much more modular***
    - Cyclic dependencies virtually eliminated
    - Cytoscape model doesn't depend on UI
      - Can use parts of Cytoscape more easily
    - Cleaner class hierarchy
      - Lower layers send events, upper layers call methods or execute tasks
    - All functionality is exported through factory methods and interfaces
  - ***Based on OSGi***
    - Apps (new name for plugins) can export functionality



Cytoscape 3.0 Model

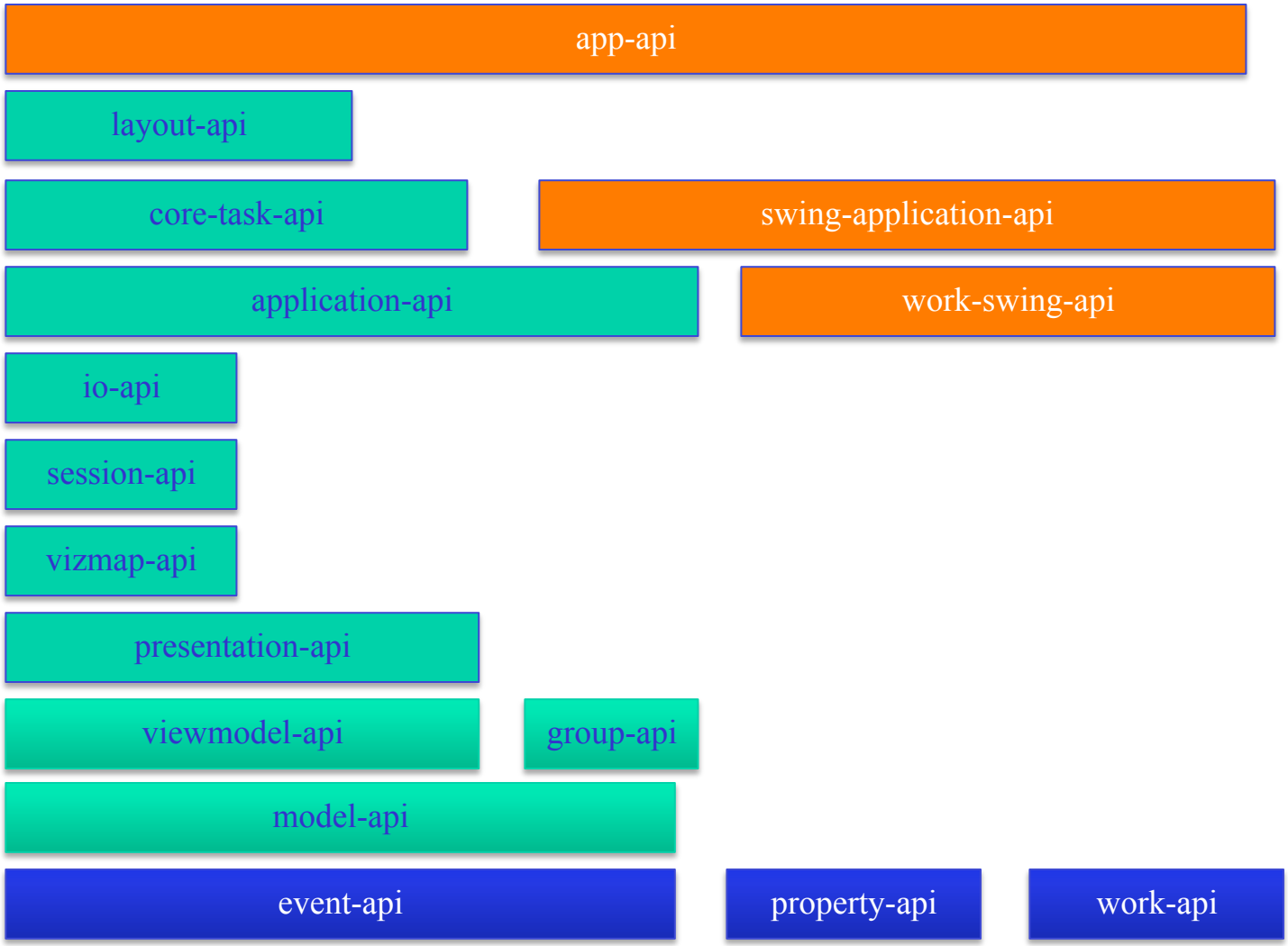
- **RootNetwork/Subnetwork**
  - ***Cytoscape 3 supports multiple root networks***
    - Nodes and edges are shared within a single root network
  - ***Networks can have subnetworks***
    - Allows for a hierarchical structure
  - ***Much cleaner API***
- **CyTables**
  - ***Generalization of Cytoscape 2.x's CyAttributes***
  - ***CyRow and CyColumn interfaces***

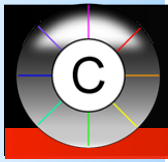


Cytoscape 3 Core

- Event Handling (event-api)
- Data Model (model-api)
- Groups (group-api)
- View Model (viewmodel-api)
- Task Execution (work-api, work-swing-api, core-task-api)
- VizMapper (vizmap-api, vizmap-gui-api)
- IO (io-api, webservice-api)
- Layouts (layout-api)
- App Framework (app-api)
- Presentation (presentation-api)
- Session (session-api)
- Application (application-api, swing-application-api)
- Utilities

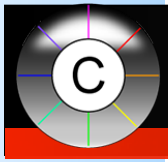
Cytoscape 3 Core





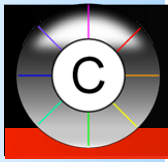
Cytoscape 3.0 Apps

- **Two types of "Apps"**
  - ***Simple***
    - Inherit from org.cytoscape.app.AbstractCyApp
    - CyAppAdaptor provides lots of methods to get access to Cytoscape core functionality
    - Pros:
      - Easy. Pretty much the same development approach as Cytoscape 2.x
    - Cons:
      - Single loader. Same version conflict possibilities as 2.x.
      - No easy way to share functionality.
  - ***Bundle***



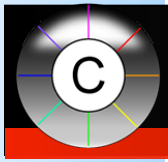
Cytoscape 3.0 Apps

- **Two types of "Apps"**
  - ***Bundle***
    - OSGi-based interface
    - Implement a CyActivator class
      - Extends org.cytoscape.service.util.AbstractCyActivator
      - Within the start() method:
        - » Use getService() to get services from environment
        - » Use registerService() to register the services provided by your bundle
    - Pros:
      - Loaded independently. Your bundle can have a different version of library than other bundles.
      - Since you specifically register services, other plugins can "get" those services.
    - Cons:
      - More difficult development environment.
      - Need to be really careful with your API/Implementation separation (a good thing) since others will depend on it.



Cytoscape 3.0 Getting Started

- **Bundles**
  - ***Separate API from implementation***
    - API should be (almost) exclusively interfaces
  - ***Many things won't have an API***
    - No services to export, etc.
  - ***Think through what services you want to use and which you want to export***



Cytoscape 3.0: Example

- **API**
  - ***org.cytoscape.group.CyGroup is a model-level object***
  - ***org.cytoscape.group.CyGroupFactory is a factory service that creates groups***
  - ***org.cytoscape.group.CyGroupManager is an object that manages all of the groups***

```
public class CyGroupFactoryImpl implements CyGroupFactory {
  private static final Logger logger = LoggerFactory.getLogger(CyGroupFactoryImpl.class);

  private final CyEventHelper help;
  private final CyGroupManagerImpl mgr;
  private final CyServiceRegistrar serviceRegistrar;

  /**
   * Creates a new CyNetworkFactoryImpl object.
   *
   * @param help An instance of CyEventHelper.
   */
  public CyGroupFactoryImpl(final CyEventHelper help, final CyGroupManagerImpl mgr,
            final CyServiceRegistrar serviceRegistrar)
  {
    if (help == null)
      throw new NullPointerException("CyEventHelper is null!");

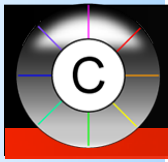
    if (mgr == null)
      throw new NullPointerException("CyGroupManager is null!");

    if (serviceRegistrar == null)
      throw new NullPointerException("CyServiceRegistrar is null!");

    this.help             = help;
    this.mgr              = mgr;
    this.serviceRegistrar = serviceRegistrar;
  }

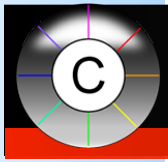
  /**
   * {@inheritDoc}
   */
  @Override
  public CyGroup createGroup(CyNetwork network) {
    return createGroup(network, null, null, null);
  }

  /* Other create methods here… */
}
```



Cytoscape 3.0: Events

- **Two types of events:**
  - ***CyEvent: single event***
    - Use when events of interest occur singly. For example, collapsing a group
  - ***CyPayloadEvent: several events***
    - Use when many events might happen in bulk. For example, adding several rows to a table
- **Use CyEventHelper to fire events:**
  - ***cyEventHelper.fireEvent(event)***
- **Register the listener object to receive events:**



Cytoscape 3.0: Example

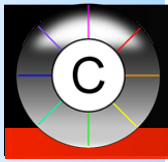
```
public class CyActivator extends AbstractCyActivator {
  public CyActivator() {
    super();
  }

  public void start(BundleContext bc) {
    CyEventHelper cyEventHelperServiceRef = getService(bc,CyEventHelper.class);
    CyServiceRegistrar cyServiceRegistrarServiceRef = getService(bc,CyServiceRegistrar.class);
    CyGroupManager cyGroupManager = getService(bc,CyGroupManager.class);
    CyNetworkViewManager cyNetworkViewManager = getService(bc,CyNetworkViewManager.class);
    CyNetworkManager cyNetworkManager = getService(bc,CyNetworkManager.class);
    VisualMappingManager styleManager = getService(bc,VisualMappingManager.class);

    GroupViewCollapseHandler gvcHandler =
      new GroupViewCollapseHandler(cyGroupManager, cyNetworkViewManager, styleManager);

    registerService(bc,gvcHandler,GroupAboutToCollapseListener.class, new Properties());
    registerService(bc,gvcHandler,GroupCollapsedListener.class, new Properties());

    // Register listeners on node selection so that we can synchronize selected nodes
    GroupViewSelectionListener gvsListener =
      new GroupViewSelectionListener(cyGroupManager, cyNetworkManager, cyNetworkViewManager);
    registerService(bc,gvsListener,RowsSetListener.class, new Properties());
  }
}
```



Cytoscape 3.0

- **Caveats**
  - ***Things are still changing***
  - ***Documentation is incomplete***
    - Methods are all documented, but higher level documentation is lacking
- **However**
  - ***Give it a try!***
    - We're really interested in what's hard and how to make it better