Links as First-Class Objects

1.0 This Document

This document describes some miscellaneous thoughts about the project to support the new workspace GUI by giving links attributes.

Throughout this document, we use terms from the

- Qt Selector Requirements document (http://cadswes.colorado.edu/~philw/guss/docSlotSelect/QtSelectorRequirements.pdf)
- Qt Selector GUI Design (http://cadswes.colorado.edu/~philw/guss/docSlotSelect/QtSelectorGuiDesign.pdf)

Where such terms are used for the first time (or for the first time in a while), we so indicate with the use of **bold** font.

2.0 Requirements of the Sim Library

Links will have attributes that support the GUI requirements to display links according to user preferences. These attributes will not be accessible through rules. Links will not be named.

Users will be able to define “groups of links” to display similarly. The group specifications will be persistent, so that new links can be added to the groups as the links are created.

The simulation library will provide links, link groups, and a link manager, and means for the GUI to query and manipulate

1. existence and specifacation of link groups through the link manager
2. membership of a link group through the group
3. member-of-what-link-groups through a link
4. link is-member-of link group through the group or link
5. resolution of a link group to a set of links through the link group

3.0 Link Groups

A link group is defined by one or two endpoint specifications, each to be applied to an endpoint of a link to determine if a link belongs to the group. If the link group definition contains one endpoint specification, the link group contains all links for which at least one endpoint meets the single specification (criterion).

For the purpose of this discussion, we will call this single selection selectionA, and refer to the endpoint that meets the criterion endA.

If a link group also contains a specification for the other end of the link, we refer to selectionB and endB.

Example (the syntax used here is of no significance):

```
selectionA : endA.name ~= “El Vado.*”,
```

```
selectionB : endB.name ~= "*.inflow"
```

This defines the collection of links between any slot on the object named "El Vado" and any slot named "inflow".

An endpoint specification is a GUS slot **selection** (instance of `RootSelection`) initially created through the GUS GUI. The GUI for creation of a link group will invoke, for each endpoint selection, the GUS slot selection GUI, the result of which is an instance of a **Slot General Selection** class.

The GUI for defining a link group will require the user to define a selection for one end (endA), and for that purpose will pop up a GUS slot selector, as described above. Then the GUI will allow the user to (optionally) define a selection for the endB, and the GUI design will imply that the group is defined by those links that meet *both* of these criteria (the result sets are intersected, with the additional requirement that the domain of the second selection is the range of the first).

REQUIREMENT of `RootSelection`: the result of one selection must be passed to another selection as the range of the second selection. This could be accomplished by letting the selection class contain a pointerr to its domain, which could be a subbasin, another selection, or a `set <Slot*>`. Sufficient and simplest would probably be to add a method `RootSelection::setDomain(cwSlist<Root*>)` and `RootSelection::setRange(cwDlist<Root*>)`, which do NOT check the objects in the sets to see that they conform to the result type (because that would create an include-file-ordering nightmare).

These endpoint specifications (GUS **selections**) persist, as attributes of the link group. They have the following two qualities:

- *completeness*: a selection may be **intermediate** (Qt Selector Requirements, 1.2.5) or **terminal**. An intermediate selection implies a terminal selection, which is the set of slots on all objects selected by the intermediate selection. For example, if the intermediate selection is "all Reservoirs", the implied terminal selection is "all slots on all Reservoirs". The link group (client code) will not accept intermediate selections.
- *resolution*: when a selection is bound to a set of slot instances (the result of the selection query), we will say it is *resolved*. A selection may be resolved once, and its result set is fixed thereafter, in which case it is **static**. If it is not static, it may be resolved many times, so that its result set is kept up-to-date with respect to the state of the model; such a selection is **dynamic**.

A static selection is resolved at the time the selection is created. Once resolved, the selection holds a set of pointers to the slots. The pointers in the set do not change except when a slot is deleted. The client code is responsible for deciding whether a change in visibility of the slot is a material issue.

A static selection instance's method `isFixedSet()` returns true.

A dynamic selection is resolved by the client code (via `resolveDynamicSelection()`), which has the responsibility for determining when the selection is to be resolved. Thus, in this case, the client has the responsibility for handling (by registering and receiving callbacks or by other means) such things as slot or object name changes, visibility changes, etc.

The link group editor GUI will invoke the GUS slot selector in a way that permits both static and dynamic selection; the scope of the selections will always be generic slots (`Slot *`).

As we mentioned above, the endpoint specifications (selections) will persist as long a the link group exists. Thus, they become part of the model, and they

- can be edited (the specifications can be modified by the user, resulting a new result set),
- are written to the model file, and
- are restored when the model is loaded.

4.0 Editing Endpoint Specifications

The GUS GUI (`GusDialog()? TODO`) can be invoked on an existing selection, in which case that selection is editable by the user through the GUS GUI. The user will be able to change a selection from dynamic to static and vv.; will be able to change the selection's filters and final selections.

REQUIREMENT of `GusDialog`: The GusDialog (?TODO) has to take a `RootSelection*` argument.

Editing a group first causes its resolved set to be disabled (TODO: RootSelection methods that apply?). If dynamic, the set is cleared. After the edit, the group resolves() itself, which causes a callback to the GUI. The GUI has to register this callback at startup - it registers with the manager. The callback issued will indicate which group is resolved to the GUI can take a look at the links it contains. (Does the GUI also have to know the prior set? We can probably handle that) (TODO).

5.0 Model Files

The selections have methods for converting to ASCII representation; in the case of static selections, the ASCII representation expresses the member slots by current name. Upon loading a model, the ASCII representation is converted to a resolved selection. (Filters might also be expressed in the ASCII representation.)

In the case of dynamic selections, the ASCII representation expresses the selection's filters. The result set is not expressed in the ASCII representation, so when the model is loaded, these selections are not resolved. It is the client code's responsiblity, as always, to determine when to resolve these selections.

REQUIREMENT of `RootSelection`: The class must offer a Boolean method `isResolved() const`.

5.1 Model Save

After all the slots' setSeries are written, we have a section for links (already there, called Object Relationships). Here we write all the links as they are now. This writing is presently from outlinks. We will have to remove that and do the writing from the Link class, as we also have to write out the other attributes. (What will we do about it if we don't want virtual functions but do want class derivation so the attributes are known only to the Qt library?)

We add a section for the link groups, and write them.

5.2 Model Load

Create the objects, slots.

Create the links - but now SimWS::addLink calls the link manager to create a Link instance. No callbacks are registered yet.

Create the groups, all unresolved.

At end of model load, the link manager :

- resolves all the groups: when a group decides that a link belongs to it, it registers a callback on the link so that a change to the link's endpoints causes a callback to the group.
- for each link, sets up callbacks on its slot endpoints
- tells the GUI that the model has loaded so the GUI can draw the links.

6.0 Accounting

These selections will be made in a context known to the GUI: physical network or accounting network. The link group GUI will indicate to the GUS selector whether account slots are to be considered when creating or editing a selection.

In the accounting context, the link group editor will in fact be the supply configuration GUI (which is now a galaxy dialog and may be a very different beast under Qt). A supply may ultimately be implemented as a special case of a link; the GUS account selector, might be used to identify a source for a supply. Keeping this in mind, we wish to design link groups in a way that will enable code-reuse for supplies in the future.

7.0 Priority of Link Groups

The workspace GUI needs to prioirtize link groups, so a link group will have a data member to hold a priority. The priority is assigned by TODO???.

8.0 Link Group C++ Public Interface

```
// TODO Link Group Public Interface
class LinkGroup
{
   RootSelection *_endA;
   RootSelection *_endB;
   cwSlist<Link *> _links;
   RWCString name;
public:
   const RWCString &getName();
   // GUI needs to be able to edit these endpoints:
   RootSelection *endA();
   RootSelection *endB(); // NULL means not used
   const cwSlist<Link *> &links(); // unordered
   const cwSlist<Link *> &resolve();
   bool contains(const Link *) const;
   bool empty() const;

   // TODO: methods to write to model file
   // TODO: methods to load from model file

   LinkGroup(RWCString &name, RootSelection *A, RootSelection *B=NULL);
   ~LinkGroup();
```

```
};
```

9.0 Link Group Manager

The link group manager is an entity that resides in the Sim library. It holds all link groups, allows access to link groups by

- name
- membership (e.g., those containing a certain link)
- priority

It also updates the link group membership when a link is deleted or created.

The link group manager defines some predefined link groups, a minimum being:

- all links
- main channel (links between MAIN_CHANNEL_DOWNSTREAM slots and MAIN_CHANNEL_UPSTREAM slots)

10.0 CALLBACKS

The manager and the LinkGroup GUI and Workspace GUI will share responsibility for dealing with these callbacks:

Callback Type	Data	Registered on, called when...	Recipient	Action / Proposed Action
WS_LINK_ADDED	WSLinkChangedData (SimWS.hpp) , Includes from- and to-SimObjs; will have to be changed to include the slots.	SimWS, after slot:::addLink works; initiated by model load or from link editor dlg	SimWSGUI	Draw line between sim objects.
WS_LINK_DELETED				Erase line between sim objs.
LINKGROUP_CHANGED (new)	Pointer to link group that has changed	LinkMgr	SimWSGUI	
SLOT_LINKS_CHANGED	SlotStatusChangeData (Slot.hpp)	Slot::addLink, Slot::deleteLink	Presently none	Resolve dynamic selectors
MULTISLOT_LINKS_CHANGED, subtypes: MultiSlotLinksChangedData::Added MultiSlotLinksChangedData::Deleted MultiSlotLinksChangedData::AllDeleted MultiSlotLinksChangedData::NameChanged	MultiSlotLinksChanged-Data	MultiSlot	MultiSlotDlg GroundWaterStor WaterUser	Update the dialog, EngrObjs update tables when columns added or deleted. /

TABLE 1. Callbacks involved, current status, proposed changes

Callback Type	Data	Registered on, called when...	Recipient	Action / Proposed Action
SLOT_UNITTYPE_CHANGED	SlotStatusChangeData (Slot.hpp), contains SimObj*, Slot *, row, col	The slots	All the slot dialogs	Update open dialogs / Reevaluate LinkGroups that contain filter on unit type
SUBBASIN_ADDED	SubBasinChange (includes the subbasin)	SubBasinMgr	SimWSGUI / LinkMgr	Updates o subbasin menu / Reevaluate LinkGroups that contain filter on subbasin name, only for RENAMED; for ADDED, DELETED do not register
SUBBASIN_DELETED				
SUBBASIN_RENAMED				
SUBBASIN_MEMBER_ADDED	SubBasinChange (includes subbasin, member)	SubBasinMgr	SubBasinMgr	Updates SubBasinDlg / ReEvaluate LinkGroups that contain filter on subbasin membership (for ADDED, DELETED), and on subbasin name (for RENAMED)
SUBBASIN_MEMBER_DELETED				
SUBBASIN_MEMBER_RENAMED				

TABLE 1. Callbacks involved, current status, proposed changes

11.0 Link (Group) Manager C++ Interface

```
// TODO Link Manager Public Interface

// return set of link groups with filter
typedef bool (*acceptFunc)(const LinkGroup *, void *data);

// filter func to return all link groups
bool acceptAll(const LinkGroup *, void *data){ return true; }

class LinkMgr : public Root
{
public:
   LinkMgr();
   ~LinkMgr();
   // TODO: methods to write to model file
   // TODO: methods to load model file

   void getGroups(cwSlist<const LinkGroup *>&, acceptFunc, void *data) const;

   void addCallbacks();
   void deleteCallbacks();

   void addLinkGroup(const LinkGroup *);
   void removeLinkGroup(const LinkGroup *);

   Link *addLink(const Slot *from, const Slot *to);
   void delLink(Link *);
   Link *findLink(const Slot *from, const Slot *to);
   findLinks(const Slot *from, cwSlist<const Link *>&);// unfortunately,
```

```
   // this has to be kept in sync with the slot's outlinks structure, gak.
   // ISN'T THERE A WAY TO AVOID THIS DUPLICATION? TODO
};
```

12.0 Link Object

A link is a member of one or more link groups. Every link is a member of the link group "all links" mentioned above.

A link object contains attributes used by the workspace GUI. These attributes include

- Sim properites: The two slots it links
- Graphical properties: Visibility, Line Width, Line Style, Color

Alternatively, the Link class is in Sim/ and the GUI library derives a class from Link to contain the additional data members that it needs. Only the GUI needs the graphical properties, and most of these attributes are embedded in inherited Qt Classes. Also, each link group will need the same set of graphical attributes. It probably does not make sense to duplicate these attributes on the Sim-side. We need to think about and discussion model loading/saving issues.

13.0 Link C++ Interface

```
// TODO Link Manager Public Interface
class Link
{
private:
   const Slot *from, Slot *to; // Not really ordered. Just a name conve-
nience.
   Link *next; // for linking together in link mgr; this list constitutes the
   // "all links" list.
public:
   Link(const Slot *one, const Slot *two);
   ~Link();
};
```

14.0 GUI Interface

The GUI interface for the Link Groups will resemble the RplGroup editor (see figure 1). Users wil be able to create custom display groups just as they can create RplGroups. A display group will consist of an object selection, and a link selection. Visbility of the entire group, the object part, or the link part can be toggled directly from the editor. The toggle is represented by a green checkmark and red 'x'. Double-clicking the link part will bring up a link configuration dialog (see figure 2), which will allow the user to edit slot selections defining the links and the graphical properties of those links. TODO: Are there any graphical properties of object groups other than visiblity? Transparancy maybe? Note: I don't know how hard/easy tranparent icons are going to be. If groups of objects have no other properties other than visiblity, double-clicking the object part will just bring up GUS.

The display group editor will display the groups in prioritized order. Graphical properties of groups higher in the list will have precedence over groups lower in the list. The editor will also include built-in display groups with non-editable selections. At a minimum, this will include a “default” display group that defines the default graphical properties of links and objects. We could also add built-in display groups for categories like main channel.

Figure 1. Display Group Editor

Figure 2. Link configuration dialog