**Servillian Solutions Inc**

Spectrum Feature: Tournament Scheduler

Projected Release: Version 2.2, Second Quarter 2012

**Design Document**

prepared by Servillian Technology Ltd

**Revision History**

|  |  |
| --- | --- |
| Person | Date |
| Bradley Holbrook | January 31, 2012 |
| Sam Bassett | Feb 7, 2012 |

**Spectrum Rules**

**Client Side**

*General (applies to windows, forms, grids, and any other components/objects)*

* Never access any controller for any reason
* Should not access external components directly (internal components are fine)
* Functions used by the component should always be part of the object itself, not defined outside the object or in global scope (on the same level as constructor and initComponent)
  + Public: if other components will be using it (refresh, reset, save)
  + Private: for internal only, such as setting up a menu during construction (i.e. many grids have a season filter dropdown that is created in the constructor after callParent)
    - Private functions should have a “\_” prefix

*Window:*

* Extend ‘Ext.spectrumwindow’
* Buttons nested inside footer (bbar or dockedItems).
  + Example: a Save button that will call the public save() method of the internal form(s)

*Form:*

* Extend ‘Ext.spectrumforms’
* Save function: most if not all forms require this. Simply get all form data and send ajax
  + Previously, we have been adding the save button to the form, but new components will have the buttons in the window (eventually we will have to update old components as well)

*Grid:*

* Extend ‘Ext.spectrumgrids’
* Subcomponent locations:
  + Top Left: filters visible records
  + Top Right: search
  + Body: list of records that can be interacted with
  + High Bottom Left: buttons that do not interact with a selected row
  + High Bottom Right: buttons that interact with a selected row
  + Low Bottom Left: pagination (optional if more than 100 records is common)
* Model for data Store defined in /js/models/
* Refresh function: Optional but recommended. Reload the data in the grid store

**Server Side**

*Model:*

* Access to database, session data, files
* Can load and access other models, but never a controller
* Can use any library
  + By default, it has loaded all libraries that the controller loaded
  + Can also load additional libraries (i.e., curl and ftp libraries are intended for use more for the model than the controller)
* File storage (creating temp files, and uploading from user)
* Session / cookie data
* Business logic

*View:*

* Displays data it is given
  + Formatting with PHP is allowed and recommended. The controller does not care about the format. For example: handling null, formatting dates, formatting decimal places
* No calculations or ‘heavy’ php logic.
* Should not access session data
* Use HTML whenever possible, instead of echo a string containing html
* Relevant CSS should be loaded from another file, not defined here in a block or inline.
* CSS classes are preferred to ids

*Controller:*

* Input validation (always required)
  + Example: use input library instead of $\_REQUEST array
  + Reject invalid or incomplete data before sending to model
* Data filtering / sorting.
* Can access all models, libraries, and views
* No heavy computations or utility functions (use libraries instead)
* No direct access to database, session
* No control or concern on how data is displayed or formatted (no HTML)

*Library:*

* Context free
* Able to be used by multiple controllers
* Algorithms / computations go here, so transforming input to output
* No access to database, controllers, models, views
* Can access other libraries
* Easy to drop into different projects / applications

*Database*

* **All database changes must be saved as an executable .sql script, and tracked by git.**
  + Save script files in (root) /sql , prefixed by the task number
    - Example: /sql/1318\_delete\_link.sql
  + Most common actions
    - Alter
    - create new tables
    - Inserting into lookup tables
    - New or altered stored procedures / functions
  + This is the only way to affect the live database (which is only done by project manager)
  + In postgres, use the right hand side tab “SQL Preview” and copy that script
    - Tables and Functions: You can get this script BEFORE the save is finished. Also, remember to change ‘newTable’ or ‘newFunction’ to whatever name you gave it
    - For functions, if you forget you can always edit the function, add one space, and grab the script again
    - For new or altered tables: if you forget you have to drop and re-create the table/column (or rewrite the script manually with the Query button)
    - For inserting records, you must write the insert yourself using the Query functions
  + If SQL fails to execute when the codebase is merged, or if there is SQL missing and live break, issue will be sent to the originating developer to re-write the script in whatever means possible WITHOUT touching the live database at all.
    - NO EXCEPTIONS.

**Tournament Scheduler Glossary**

*Convener:*

Organization that manages tournaments and only tournaments (in contrast to a league)

*Tournament:*

An event that plans for teams to play each other through a single, or a series of, block(s)

*Event (Tournament Type):*

A tournament that has no seeding predetermined by a season.

*Playoff (Tournament Type):*

A tournament that has team starting positions set by a season.

*Combined (Tournament Type):*

A mix between an event and a playoff. This usually comes into play when a playoff tournament allows invitations of additional teams.

*Format:*

A type of tournament structure. Currently only two

* Round Robin
* Knockout (also sometimes called ‘Elimination’)

A tournament can consist of one or more different formats (see blocks)

*Block:*

A section of a tournament, with teams, games, and exactly one format. Can be considered an instance of a format with a specified type, and all other parameters and context determined.

*Group Block:*

A self-contained group of blocks. A tournament can have several groups.

*Template:*

A preset saved group of blocks.

*Placeholder:*

A team type that is not treated as a regular team in the system, except for scheduling games. Regular league teams, when registering for a tournament, will have an option to register as the placeholder team and then replace them in all schedules.

*Active Registration:*

A tournament with active registration allows a team to select the placeholder team they wish to replace.

**Outline of New Elements, and changes to Existing Components**

*(New) Screen: Manage > Organizations > Manage Conveners*

*Conveners Grid*

- Identical to Manage Leagues except for Conveners

- Use lookup table for league types

- Cannot create seasons or schedules of type ‘League’

*(Existing) Screen: Seasonal Operations > Seasons*

*Seasons Form & Grid*

- A season type will be added with the following options: 'League', 'Tournament'.

*Season Details Window*

- A new option will be added allowing 'Active Registration'.

*Registration Grid*

- A Placeholder Team field will be added to signify the placeholder team selected

*(Existing) Game Results and Standings*

* Add the option to specify one team as the winner of a game, even if the score is tied
* Form Changes anywhere score is entered (game results component)
  + Three Radio Options
    - Tie (default selected)
    - Home wins tiebreaker
    - Away wins tiebreaker
  + Tiebreaker Description (if tie not selected)
* This tiebreaker needs to be saved in the database along with game results / scores
* Internal computation of statistics & standings needs to check this when a tie-breaker is detected, and adjust win/loss/tie, etc., accordingly.

*(Existing) Screen: Seasonal Operations > Teams*

*Teams Grid*

- Add 'type' field to team. Types will be:

- Active (Team is a real team, registration approved)

- Registered (A team has registered into this placeholder, but not approved)

- Placeholder (Theoretical team, no team registered into this space yet)

*(Existing) Screen: Create Schedule*

*Welcome to the Scheduler Wizard (Form)*

- Season Selection should determine rules below (See rules section)

- Hide rules to start, show only once season option selected (this will depend on the season

type)

* If tournament type is selected use a different save action:
* playoff/ post\_create\_playoff

*Schedules in Progress (Grid)*

* New Columns:
  + Season Name
  + Season Type
  + Tournament Type (blank on League Schedule)

*(New) Screen: Create Tournament Schedule – sub screen of ‘Create Schedule’*

*Tournament Scheduler Sections (Toolbar)*

- Global Rules *(Existing)*

- Timeslots *(Existing)*

- Structure *(New)*

- Audit Reports *(Existing)*

- Finalize Schedule *(Existing)*

**New Components**

NOTE: Unless otherwise specified, forms will also require a Window component (not specifically listed here), which has a save button that will call the forms defined Save Action. It will be the job of the toolbar or grid button to create the window and form, put them together, and handle show/hide events.

*Grid:* Tournament Structure (Groups and blocks) – See Figure 1

* Type: Ext Treeview
  + Two levels maximum
  + Roots are groups
    - Leaves are blocks
* No filters, search, or pagination
* Load method
  + json\_block\_groups
* Drag and drop to change / save :
  + The order of blocks within a group
  + The order of groups
  + Move blocks between different groups
    - Blocks cannot be outside of a group
    - Groups cannot be in another group
* Columns:
  + Name
  + Type
  + Format
  + Teams
  + Games
  + Requires
* Lower Left Toolbar Buttons (in order):
  + Create group / block
  + Create from Template
* Lower Right Toolbar Buttons (in order):
* Depending on whether a group or block is selected, different buttons will show:
* If Group Selected:
  + Group Rules
  + Remove selected Group
  + Save Group as a Template
* If block selected:
  + Select timeslots
  + Block editor
  + Rules
  + Remove selected block

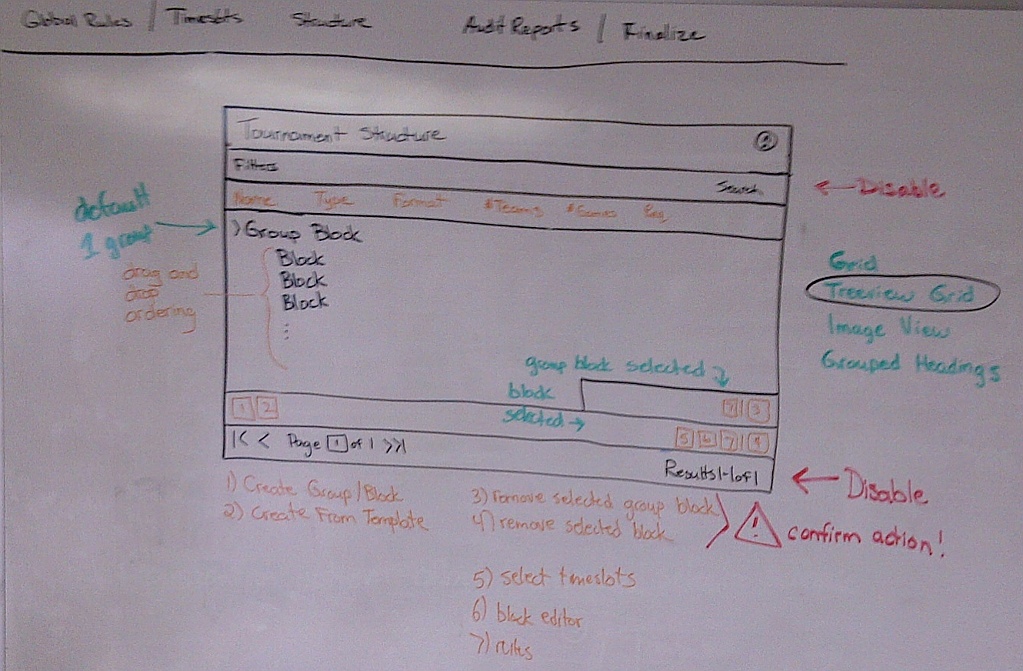
****

Figure : Tournament Structure Grid

*Form:* Rules

* Load / Extend from League Scheduler
* Hide ‘# games per team’
* Add ‘Maintenance Time’ (time)
* Add ‘Maintenance Interval’ (numeric)
* Used by Global rules, Group Rules, Block rules, so a hidden input is needed to identify the type and object associated with it, if any
* Load methods (if not global):
  + json\_block\_rules
  + json\_block\_group\_rules
* Save methods (if not global) , create and update use the same method
  + post\_block\_rules
  + post\_block\_group\_rules

*Grid:* Create Group from Template

* Show list of templates available, in each of the following types
  + Spectrum templates (built in)
  + My templates (owned by active org, ignore user created\_by)
  + Saved templates (all orgs)
* Load method:
  + json\_block\_templates
* Preview button
* Select button
* Save (select) action
  + post\_block\_group\_from\_template

*Form:* Block Editor

* Two horizontal panels (columns)
  + Left is an accordion menu
  + Right panel changes to a different view depending on user selections
* Accordion Menu Items
  + Format Type
    - new form “Format Type” (see below)
  + Format Details
    - Two new forms for “Format Details” (see below)
  + Timeslots
    - Load or extend existing timeslot grid but hide all the buttons
    - Replace selection model with multiple select checkbox model
    - Button to save assignments (optional: zero is allowed)
  + Block Structure
    - Two new forms for “Block Structure” (see below)
  + Team Sources
    - new form “Block Structure” (see below)
* Save actions for assigning timeslots
  + post\_assign\_block\_timeslot
  + post\_remove\_block\_timeslot

*Window:* Block Editor

* View is the ‘Block Editor’ form
* Buttons (lower right) move accordion for you, and save if needed
  + Back
    - Navigate between views / menu items
  + Save & Continue
    - Navigate, or call form.save

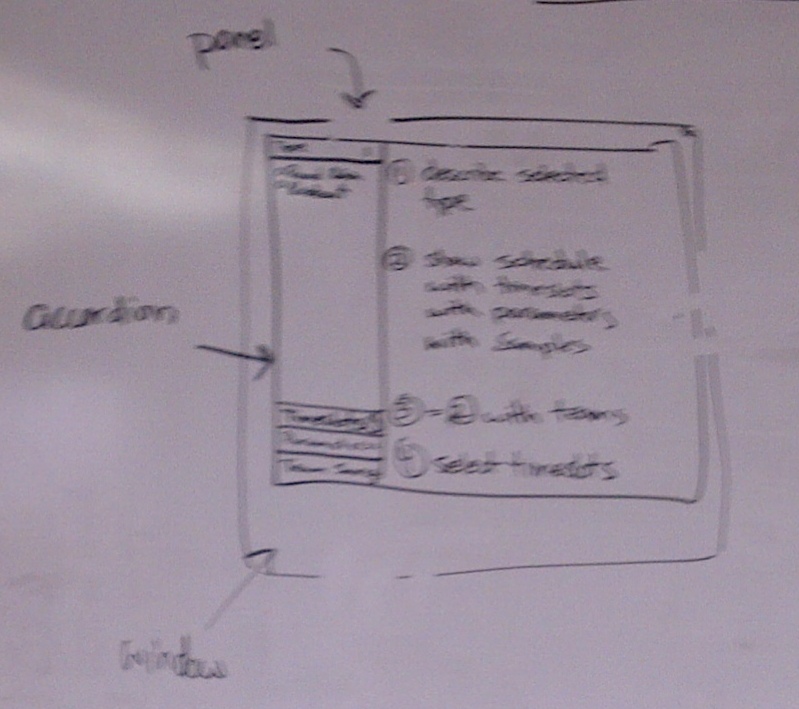


Figure : Block Panel

*Form:* Format Type

* Select one of two options (from a new lu\_ table)
  + Round Robin
  + Knock-out
* Describe each type
* Example of type (image)
* Cannot continue without one option selected
* Save action
  + Nothing to post yet, just save the type and load the appropriate form

*Form:* Format Details - Round Robin

* Number of team slots (numeric > 0)
* Number of Rounds (numeric > 0)
* Save action
  + post\_block

*Form:* Format Details - Knockout

* Number of team slots (numeric > 0)
* Games per match (numeric > 0)
* Knockout number (numeric > 0)
* Dead End number (numeric > 0 and < ‘knockout number’)
* Is Modified? (bool)
* Start Type (drop down, lu\_ enumerated). Different options will be available depending on the number of team slots
  + Balanced Start
  + Staggered Start
  + Low-Seed Elimination
* Give Priority (loss not allowed if knockout number ==1)
  + Win bracket (default)
  + Loss bracket
* Save action
  + post\_block

*Form:* Block Structure - Round Robin

* This simply needs a grid of games (it can use the session\_games grid from the league scheduler)
* date time and venue : (leave blank for now)
* Pass to backend all the parameters from the previous forms, and
* Add in a drag and drop element, so dragging a team onto a row will assign to that placeholder team
* Save action
  + post\_block\_teams

*Form:* Block Structure - Elimination

* See Figure 3
* First, the technical details:

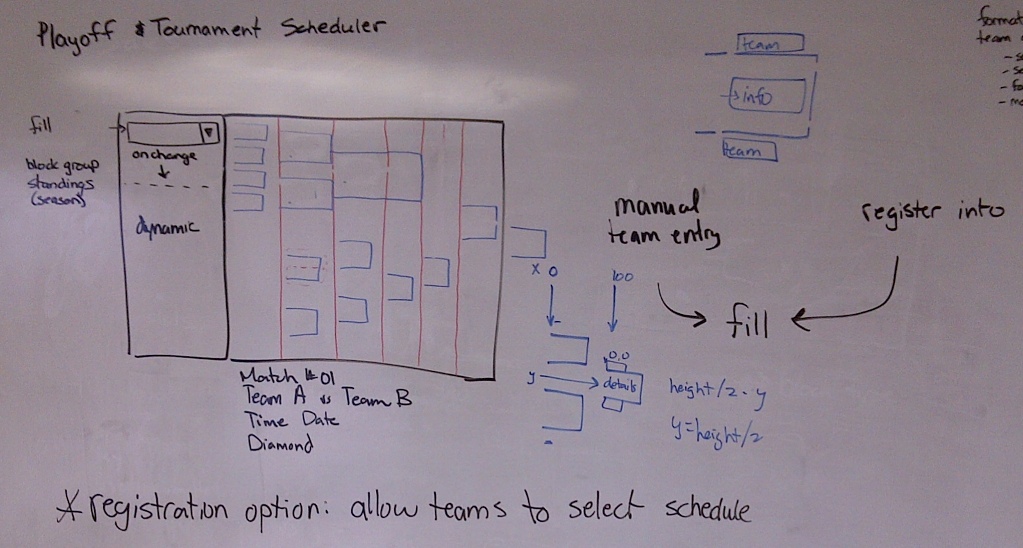


Figure : Block Editor

* Requires creation of new custom Ext.panel with layout “absolute” , which allows each element to be positioned with exact (x,y) coordinate system. Each element will be of type ‘html’ as they are just loading plain images and text
* <http://www.objis.com/formationextjs/lib/extjs-4.0.0/docs/api/Ext.layout.container.Absolute.html>
* Constructor will call the load action, which will transform all block variables and games into the array of brackets, which is ready to display based on all the (x,y) coordinates of every piece.
  + Minimal computation should be done in JS side.
  + This JS component will not care if blocks are displayed wrong, or overlap, or other issues. Its only job is to display what it is given.
  + If things are displaying wrong, that is the job of the load action to compute the locations of all the components correctly
* Block Panel Variables: (different from the block variables for the form)
  + Brackets[] : array of brackets. In Figure 3, each blue item within the panel is a bracket in this array
* See Figure 4 for a sample Bracket

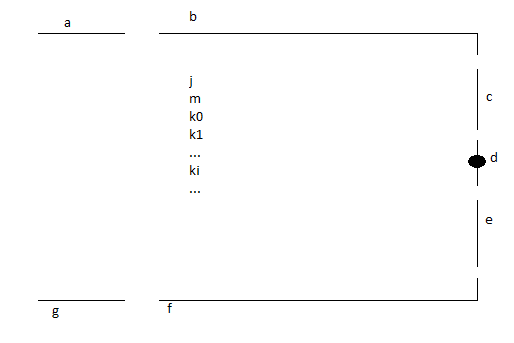


Figure : Bracket

* Bracket variables (called bracket Pieces):
  + a) ltExt
    - can be null
  + b) tPanel
  + c) rtExt
  + d) mConnector
  + e) rbExt
  + f) bPanel
  + g) lbExt
    - can be null
  + j) Details
  + k) Game[]
    - gDateLabel
    - gTimeLabel
    - gLocationLabel
  + m) matchNumberLabel
  + n) bracket
* Each Bracket Piece has
  + start[] – the top left corner, with indices start[‘x’] and start[‘y’]
  + end[] – the top left corner, with indices end[‘x’] and end[‘y’]
  + img: the image for this piece, or null if no image needed
  + lbl: label to show, or null if none is needed
* So, for example, we can reference brackets[i].rtExt.start.x , or brackets[j].bPanel.img
* Public Methods
  + reset() to erase all showing
  + drawBrackets()
    - Loop on all brackets, and call \_drawBracket on each one
* Private Methods
  + \_drawBracket(b)
    - Loop on all pieces in this bracket, and call drawPiece in each one
  + \_drawPiece(p)
    - Pull the data needed out of this piece and then call drawBox
  + \_drawBox(sx,sy,ex,ey,img,lbl)
    - This will add an xtype panel with the given start and end coordinates. It will have to compute the height and with, so as follows:
    - {x:sx,y:sy,width:ex-sx,height:ey-sy,html: html\_str }
    - Where the html string is generated from the img and lbl, so it adds the image src and label text if they exist as needed, with proper spacing, etc
* Load action:
  + json\_build\_block\_brackets

*Form:* Team Sources

* Team Source types / methods:
  + Seeded from the season
    - Places teams based on standings rankings
    - Allows manual editing after
    - Option to pull wildcard teams from multi-level standigns
  + From a different group
    - Placeholder teams that are assigned as W/L from other groups
  + Registration
    - Placeholder teams with a unique name that a registering team can select when entering the tournament

**Server Side:**

**Database Changes**

*New Table:*

Name : schedule.lu\_block\_format

Columns

* format\_id: int
* format\_name: varchar
* is\_enabled: bool (defaults true)

Data

* (1,’Round Robin’,true)
* (2,’Knock-Out’,true)

*Existing Table*

Name: public.lu\_team\_status

Data

* (3,’Placeholder’)

Notes:

Validate that all queries pertaining to teams are not gathering placeholder teams when they are not required to. Some SQL may not consider team status at all, and they will need to make sure that the status is !=3, for regular teams only. Also, a query looking for approved teams may simply check that team\_status !=2, which used to imply that the status must be 1 (approved). This is no longer valid, hence for getting approved or unapproved teams we must always inner join on this table and on the exact value desired

*Existing Table*

Name: public.lu\_schedule\_type

Old Data

* (1,’Finalized’)
* (2,’Tournament’)
* (3,’Incopmlete Wizard Save’)

New Data

* (1,’Season’)
* (2,’Tournament’)
* (3,’Season Incomplete’)
* (4,’Tournament Incomplete’)

*Existing Table*

Name: public.schedule\_session

New Column

* season\_id (int)

Fuctions: insert\_schedule\_session:

Update this function to also take season\_id, and save in this column. Ensure changes are reflected in the relevant schedule\_model function

**New Controller : tournament.php extends schedule.php**

*Public Functions*

\*Note: all public functions will have permissions assigned to league executive and scheduler roles. Also remember to grab the generated sql maintenance files in github by task number.

post\_create\_tournament

Validate input and create a new save file of tournament incomplete. Use the schedule\_model function that inserts into schedule\_session table

post\_block

insert or update a block

post\_block\_order

change the order of blocks

probably called by drag and drop events

post\_block\_teams

Assign teams to a block. Will handle differently depending on the block type (rr,ko) and the team selection type (placeholder, season, manual, etc). Still stored only in session

json\_blocks

get blocks for a given group

post\_block\_group

Create a new blockgroup with parameters

json\_block\_groups

get all block groups for the current schedule

post\_block\_group\_from\_template

take a template id and create a block for current schedule

json\_block\_templates

get all block templates. possibly filter options here

post\_create\_template\_from\_group

take a saved block and publish as a template

possibly check first to see if a similar/identical template already exists, if so reject.

Possibly use a ‘published’ flag

post\_assign\_block\_timeslot

assign the given timeslot to the given block

probably called by the selection model checked event, so save as soon its checked

post\_remove\_block\_timeslot

remove the given timeslots to the given block

probably called by the selection model checked event, so save as soon its checked

post\_block\_rules

save the rules attached to this block

json\_block\_rules

get the rules saved to this block

post\_block\_group\_rules

get the rules saved to this block group

json\_block\_group\_rules

get the rules saved to this block

json\_build\_block\_brackets

builds the brackets for the customized absolute layout component, complete with (x,y) cords for each block piece, and labels, images, etc. Requirements are that it fits all the parameters of the KO block, its ready to display, and that it displays correctly with the given coordinates

**New Model : tournament\_model.php extend schedule\_model.php**

**New Model : block\_model.php extend tournament \_model.php**

* Each method from tournament controller will most likely have a similar method in one of the \_model file to get/insert/delete/update the relevant data in the session.
* Important: if a \_model method is using session data instead of database, prefix the method with ‘s\_’
  + For example: ‘s\_insert\_block’ or ‘s\_delete\_block’ or ‘s\_get\_blocks’

**New Library : tournament.php extend scheduler.php**

Details of this will come after the design of everything else is finalized, as the processes here may change a bit. This is essentially where the implementation of all the above options is calculated. Its main tasks will be as follows:

* Computing blocks into block pieces, ready for display by the JS view
* Scheduling the tournament games itself, similar to the parent library

Rough Notes:

Use some inherited functions, possibly overwrite some, others will be ignored. For example create\_matches will be skipped completely, and find timeslot may use different logic. Instead of overriding methods, we may implement a ‘state’ or ‘mode’ flag in the base library that will instruct it on which logic type to use – this avoids any copy-paste of methods to make one minor change for tournaments.