**Synching the CAT with the web:**

Here’s what can get mismatched:

1. Participants have changed; a debate on the web doesn’t have the same competitors or judges as the uploaded file
2. Debate no longer exists; debate on the web does not appear in the uploaded file
3. Debate in uploaded file does not exist on the website
4. Results differ between website and the upload file (do we want to shut out judges; you vote once, and after that you have to contact the tab room to change your decision—we did at the NDT)?
5. Entry names have changed; spellings of judge or team names are different, although IDs match

Is this list complete? What else can get garbled?

Possible approach -- does this work?

For all pairings issues, over-write the web with the upload file (will there be a unique identifier mess – every time a round is uploaded the old one is deleted and new unique identifiers are generated?). (Will there be times when the web pairings might be more accurate than the offline ones?)

For all ballot/results issues, we either (a) lock the judge out after they vote, or (b) make sure we NEVER write a blank result over an existing result in either direction, and/or (c) use timestamps, keep the newest decision by default, and notify the director of the discrepancy.

For all name/spelling issues, automatically process any changes if the account isn’t actively managed by a competitor or judge, and otherwise notify the active user of the name change and let them approve it.

OR

Timestamp the upload file, ONLY upload changes, always assume the upload info is correct and can simply over-write the web, with 2 exceptions: (a) NEVER put a blank result over an existing result, and (b) DON’T make name/spelling changes to active accounts (notify users instead). Will that handle the 5 situations above?

CHRIS: Over-write all rounds, lock judges out, no timestamps, use the local name.

**Basic data structure and examples**

All the data are stored in a file called “TourneyData.xml”. It is a 2-deep structure that contains tables (with columns/fields) and records. The data are all loaded into a dataset named “ds” on every form when the form loads. All changes made to the dataset are automatically saved whenever the form closes, and the program will enforce that only one form can be open at a time.

The program will automatically create a folder called “CAT” in your documents folder, and that’s where all the data will be stored. The program itself can run from anywhere. For a test run, click the “DOWNLOAD A TOURNAMENT FROM THE INTERNET” button and it will automatically download the TourneyData.xml file for the 2012 Fullerton tournament and the TourneyData.xsd file. To use a different file, just replace the TourneyData.xml file in the CAT folder in your documents library.

Pairings are contained in 4 tables: ROUND, PANEL, BALLOT, and BALLOT\_SCORE. Rounds occur in timeslots, and each division (called an “event”) has a different round. ROUND is the parent for all other tables. For example, in timeslot 2 there might be a round for JV, a round for open, and a round for LD. A “panel” is where information about each individual debate is stored; ROUND is its parent. A round can contain several panels; in a 10-team division there would be 5 debates and hence 5 panels (assuming 2 teams per debate). A BALLOT exists for each judge and each team on each panel; PANEL is the parent table of a BALLOT. Ballots contain “ballot\_score”s, but those don’t have to be accessed for pairing or judge placement purposes.

The structure of the entire data file is contained in a file “TourneyData.xsd.” Here’s an example of the “judge” table structure:

<xs:element name="JUDGE">

<xs:complexType>

<xs:sequence>

<xs:element name = "ID" type="xs:integer" />

<xs:element name = "DOWNLOADRECORD" type="xs:integer" nillable="true" />

<xs:element name = "SCHOOL" type="xs:integer" nillable="true" />

<xs:element name = "FIRST" type="xs:string"/>

<xs:element name = "LAST" type="xs:string"/>

<xs:element name = "OBLIGATION" type="xs:integer"/>

<xs:element name = "HIRED" type="xs:integer"/>

<xs:element name = "TABRATING" type="xs:integer" nillable="true"/>

<xs:element name = "AVGPREF" type="xs:decimal" nillable="true"/>

<xs:element name = "NOTES" type="xs:string" nillable="true"/>

<!-- Timeslot and division columns are added at run time to mark eligibility/availability for different -->

<!-- divisions and timeslots; columns are named either TIMESLOT or EVENT followed by ID for timeslot or event -->

</xs:sequence>

</xs:complexType>

</xs:element>

This means there is a table called “JUDGE” with 10 fields, starting with “ID” and ending with “Notes.” Here’s how you would iterate through the table and access the values (AvgPref is calculated during setup, and recalculated when teams change their prefs).

Dim JudgeID As Integer

Dim AvgJudgePrefRating As Integer

For x = 0 To ds.Tables("Judge").Rows.Count - 1

JudgeID = ds.Tables("Judge").Rows(x).Item("ID")

AvgJudgePrefRating = ds.Tables("Judge").Rows(x).Item("AvgPref")

Next x

There are many functions that are stored in modules and can be called from any form. Usually, you have to pass the dataset to the module. Here are some useful functions for judge placement:

GetJudgeRating: Takes a judge and team number, a specification for ordpct or rating, and returns the relevant rating or ordinal percentile.

GetRoundsJudges: Takes a judge number and returns the number of rounds the judge has already heard

MaxLeft: Takes a judge number and returns the maximum number of remaining prelim rounds the judge can hear, accounting for timeslots where they are unavailable.

ActiveInTimeSlot: Takes a judge number and identifies whether they have already been scheduled to judge in the timeslot.

CanJudge: Takes a judge number and panel, and identifies whether they judge can hear the teams on the panel. Does NOT check for non-school conflicts or strikes, which are stored in the JUDGEPREF table. A judge pref of 999 (which can be accessed via the GetJudgeRating function) means there’s a non-school conflict or strike.

**Tiebreakers and scores**

There are 3 things we need; **tiebreakers**, **scores**, and **score settings**.

A **tiebreaker** is ALWAYS a function. Even when we say things like “wins” we mean win totals. Some are fairly standard functions, like summed totals or high-low totals, but some require custom figuring, like judge variance or opposition wins.

Tiebreakers are linked to “rounds” via “tiebreak\_set” locations. Each round uses a tiebreak\_set to determine the seedings. Tiebreak\_set might be things like “prelim teams” or “prelim speakers” or “elim teams.” Each tiebreaker needs to indicate the raw score that needs to be collected in order for the function to calculate.

A **score** is an individual value that a judge gives a team or speaker. From what I can tell, there are only 5 in use anywhere (win/vote, speaker points, speaker ranks, team points, and team ranks) and I am confident enough of that I’m almost willing to hard-code them. Tiebreakers are calculated on the basis of raw scores.

Scores require **score\_setting**s. Scores can vary based on increment (do you allow whole points, tenth points, or half points, etc.). So, there need to be score\_settings that are keyed to “tiebreak\_set”s, so that different divisions can use the same tiebreakers (like speaker points) with different settings (PoFo might use whole points, while policy uses decimals).

There are 5 classes of tiebreakers. (1) Those that involve summing points or ranks by speaker or team, with or without high-low drops. They require 2 functions, one for speakers and one for teams. Ballot counts can be processed in these functions. (2) Wins, which require separate processing because they may involve counting ballots. (3) Judge variance. (4) Opposition wins. (5) Opposition point or rank totals. (6) Random.

All tiebreakers that involve high low drops and totaling are processed with function SUMSCORE, which works for both teams and speakers.

There is a separate function for JUDGEVARIANCE, which works for both teams and speakers.

SUMSPRKBYTEAM calculates team points and ranks based on individual speaker points and ranks.

SUMOPPSCORE will total opposition points or ranks of individual speakers.

GETTEAMPOINTS retrieves team speaker points. Team ranks can be processed through SUMSCORE.

GETWINS processes wins, as does GETOPPWINS. GETOPPWINSFORPULLUPS is not used when creating the tiebreaker tables, but is used by some pairing routines.

HADBYE returns the number of byes, and is also used by pairing routines and not the tiebreaker table.

**BYES**

Byes are indicated by -1 in the judge field for the ballot for byes involving only 1 team; if there are 2 teams and one forfeits side information is retained, and if there are 2 teams users can optionally indicate that the judge should get credit for hearing the debate. Note that a -99 is blank judge field, while a -1 in the judge field indicates a bye.

Entering a score of 0 for the ballot\_score will halt averaging; otherwise, the functions will all average the scores. The exception is ballots (Score\_ID=1), where a 0 or 1 is saved at ballot entry. Entering a -1 for the judge or side will automatically count the ballot as a win, otherwise, it will count the 0 or 1 for the ballot score.

When computing tiebreakers, you STILL need a “1” for the ballot to indicate a win, and -1 for speakers for your points to be averaged. This means you must store the “1” for the ballot at the time the bye is assigned. When pairing, the judge/side thing will be used to determine the side due.

1. If only one team, enter -1 as judge and -1 as side, -1 for all speaker points, and a 1 for ballot.
2. If no debate occurred, enter -1 as judge for both ballots, enter a ballot for each team, and average or don’t average points.
3. If a debate did occur, keep judge field, enter ballot for each team, and points or average for each speaker.

This will take care of most of the bye processing, but separate processing must be built into these functions: JUDGEVARIANCE, GETWINS, and GETOPPWINS.

**The big project:**

1. Everything must be coded in the general case, to include any number of teams, judges, and balloting options per round.
2. Everything must be expandable, so if a new format comes out we might need to add a setting or a tag here or there, and maybe some new code to process it, but the data format should not change.
3. Everything must be scalable, so that really big tournaments won’t grind to a halt.
4. We must be able to accommodate multiple divisions of disparate events, so that we should be able, for example, to run parli, world’s, LD, and policy out of the same computer.

**Big Ideas**

Define all the datatables in the .xsd file. Whenever you load the table structures will be there even if there’s no data in them. But, when you save, it will delete any field that’s empty, so be careful.

On each page, add a check that will look for the tables that are necessary for the forms to run.

On each datagrid or page, include instructions on how to do core functions.

Each datagrid can pop up a message box by column. You can then suppress the help. There’s a tag in the tournament settings that will allow you to suppress all the help things.

**Functions in modGlobals:**

SaveFile

LoadFile

MakePrimaryKey (only for LoadFile)

FullNameMaker: Makes either a full team name (i.e., “Arizona State Symonds and Symonds)or an acronyms depending on whether you send you send “Full” or “Code” as the type.

FixMissingSchools: enforces referential integrity between users in the TourneyDataMaster file and the schools table in the TourneyData file. Assumes that school 11 is for hireds; need to figure out with Chris an unaffiliated field.

MakeUnboundComboBox: Makes a combobox populated with the array and default value you send it.

GetTagValue: For any table with a tag field (usually the settings tables), it finds the tag for the specified string you send it and then returns the value for the tag.

GetEventSetting: Receives an event and Tag value, and returns the value for the specified tag for the specified event/division

GetName: Receives and ID and a field, returns the value for that field as a string

GetTeamBySpeaker: receives a speaker ID, returns the team the speaker is on

GetSpeakersByTeam: takes team ID, returns an sql string with student numbers

MakeResultsTable: Creates a table of all pairings in a given round, with or without results

GetTeamN?

GetSpkrN?

GetTBList?

GetRowInfo: Receives the name of a table and an ID value, and returns the specified field

GetElimName: Returns a string value based on a RD\_NAME value

GetSideString: Receives a numeric value, and assigns a string based on the SideDesignation value stored in the event\_Setting table

ValidatePanel: Confirms that a panel has the right number of ballots based on the number of teams and judges per debate specified in the settings. Returns a string value of “OK” if everything is fine and an error message otherwise.

**Functions in modPairing**

PullBallotsByRound: Pulls all the ballots assigned in a given round

AddJudgeToPanel: Stores the judge on the panel; returns “OK” if everything is fine and otherwise returns an error message

AddTeamToPanel: Stores a team on a panel; returns “OK” if everything is fine and otherwise returns an error message

AddPanel: Creates a new debate, sets the fields based on the number of teams and judges based on event and round settings

ActiveInTimeSlot: Takes a team or judge number and returns a Boolean value indicating whether they have been assigned to a pairing in a given round.

ELIMS

An elim seeds table is created that, for every elim round, just stores this information:

dr.Item("Event")

dr.Item("Round")

dr.Item("Entry")

dr.Item("Seed")

The teams stored in the entry field are used for pairings.

**Room/Panel thing:**

When the .xsd file has the ROOM field on the PANEL table it throws a “input string in wrong format” error. If you change the field type to string it works fine, and you can then change back the type to integer, but obviously that’s too awkward for general use.

**Ballot Codes**

Ballot code is Tournament-Round-Judge. An issue will be how to sync up the round number. Presumably, tournament and judge numbers will be synched through the API.

**Tournament Setup**

Need to populate:

First step: TOURN table, TOURN\_SETTING, and timeslots. Set defaults if they aren’t there.

Second step: Set up EVENT and EVENT\_SETTING (divisions). You can probably do this before step one, but there’s no good reason to.

Third step: Set up tiebreakers. You MUST complete the 2nd step first; tiebreakers are specific to divisions. In most cases, all that is necessary is clicking on the default tiebreakers.

Fourth: Set up rounds. You must complete both the 2nd and 3rd step before completing this step. Note that rounds are linked to divisions and tiebreaker sets.

Outstanding issues:

1. Reading in the TourneyDataResult.xml file – if tiebreakers are specified, it uses them. If not, it supplies the defaults. It IS possible to infer defaults from the FINALRANK table, but that appears to be more headache than it’s worth for now because 99% of the results will come from TRPC and that will specify tiebreakers.

TABLE NAMES

1. TOURN
2. ENTRY
3. ENTRY\_STUDENT
4. EVENT
5. EVENT\_SETTING
6. JUDGE
7. ROOM
8. SCHOOL
9. ROUND
10. PANEL
11. BALLOT
12. BALLOT\_SCORE
13. SCORES
14. SCORE\_SETTING
15. TIEBREAK
16. TIEBREAK\_SET
17. ELIMSEED
18. TIMESLOT
19. BRACKET
20. JUDGEPREF
21. TOURN\_SETTING