theMadStatter
July 3, 2015

DerbyJSON

version 0.2

DerbyJSON is a file format designed to support storing derby game data in an external file. This includes not just data specific to a single game, but also supports multi-game data (and includes the ability to identify skaters across multiple games). The format is designed to be flexible to allow for multiple uses - for example, besides storing the "stats book" for game, this file format could be used to generate a directory website for all leagues, or provide logo information for a scoreboard when traveling, or generating a program with a team's rosters and pictures.

Because of this flexibility, no single document will use all the features possible. However, this does provide the ability for bits and pieces of multiple items to be used together (so this data format could be used to store team rosters in one system, which then generates a blank game template, which can be used by a game data system, which then exports the game data to another system to analyze it).

Design Notes

We want to define a schema that captures everything that is currently captured by the StatsBook, with additional optional information that can also be captured:

1. Time stamps on events (from potential future real-time data collection tools)
2. Unique skater identification, to allow skaters who change names (or skate under different names on home vs travel teams)
3. Support rule variations (such as "20 minute periods, 5 penalty foul-outs" common amongst home team play), as well as different versions of rules (including older "no-minors"), and potential future versions (such as the way "30 second penalties" was added).
4. Allow for recording additional staff information, including bench coaches and officials, to allow for tracking participants other than just skaters
5. Import/Export to the stat-book
6. Become an standard interchange mechanism between different roller derby tools
7. History sensitive - skaters will transfer between teams/leagues, teams will come and go, skaters will change names. This schema needs to be able to handle that information
8. Make it easy to work with the simple and common place, allow for more complex exceptional handling for other (for example, an individual that is both a skater for one league, but an official for another league)
9. Support both formal sanctioned bouts, "home bouts", or even "mixers" (with skaters composed from multiple leagues), or practice scrimmages
10. Easily human readable, even to the point where a human could create a file manually.

To this end, DerbyJSON is designed to record a series of "events" that happen in game - the result being something that can be viewed manually in a linear fashion designed to mimic the action on the track. This is as opposed to either factual summary ("this action occurred 5 times in the period") or a "spread sheet" model (which separates the events of each team from the other). The goal is to allow for the interleaving of data as it is collected (if possible), and as such, all events have an optional timestamp element. When data is imported from a stat book worksheet, very little temporal information is provided (unless the bout timer paperwork is filled out, and some additional timing information can be derived from the penalty box paperwork when the box trip record spans across multiple jams).

DerbyJSON objects can be broken down into several categories:

- Static data objects - such as the venue, skaters, officials. All of this information should be determined before the game starts and should not be edited during game-time.
- Event data objects - this is a record of all things that happen during the game, from starting whistle to final whistle (or end of official review at the end of the game).
- Organizational objects - the game (root), team, period, and jam objects.
- Support objects - used by other objects to contain structured data (for example, "notes" objects)

References and Unique Identifiers

One of the big problems is how to handle skaters with multiple names that skate for multiple teams (or may be an official for another league, etc...) A similar problem crops up with venue names - there are a number of locations that have the same name (but are in different states). It may also happen that data collection tools end up assigning different unique ids to the same bout as well. Related to this is how to refer to these "not uniquely named" things.

Note that currently, this applies only to venues, persons, and bouts. It is assumed that full league names are unique, and that teams are unique within the league (or, in the case of non-league teams, such as "Team USA", unique period)

First, all such objects include zero or more "uuid" objects (sorted in the "uuid" property). Having multiple uuid's may seem counter-intuitive, but it will make it easier to combine data sets (so a skater in one dataset can have a different uuid from the skater in another dataset and those two uuids can then be "merged"). UUIDs are, however, designed to be unique, and with sufficient entropy so as to not create a collision.

Second, within a document, a skater is referenced via a local "skater reference" attribute (unique within the required scope). This string can be in the following formats:

- "team:skater", where *team* refers to:
  - team abbreviation
  - special team names of "home" and "away"
  - team name
  - team UID

  and *skater* refers to:

- skater number
- skater name
- special skater number “???" for unknown skaters

- A unique string such as UUID or other game data collection specific unique reference.

The first is the preferred format. The last option is the least desirable since it makes it difficult to manually read the events and figure out who it refers to. A compliant reader must be able to support all styles.

> **Implementation detail**: It is is recommended that a dictionary be used to map skater references to underlying person objects. This can be done by taking a team and iterating their roster, and forming all possible references (including “home" and "away" forms) with all possible names/numbers, as well as any provided person UUID. Those keys can then refer to the person object

NB: The unknown skater is never written to DerbyJSON files.

A team reference is a string that can be the team’s abbreviation (if unique within the scope), name, UID or the special abbreviations “home" or “away”. A league reference is similarly their name, abbreviation (if unique), or UID. An association reference is simply their abbreviation (such as “WFTDA" or “MRDA”), which are assumed to be unique.

Timestamps

Timestamps are used to record when events happen, and can be recorded in a number of different formats:

Absolute	An absolute time, as specified by a 24 wall clock, or in seconds since the start of the “epoch”
Period relative	Related to the start of the current period (NB: This needs to factor in timeouts and other stoppages of the period clock to property convert to an absolute time)
Jam relative	Relative to either the start of the end of the jam (this can be useful for recording when the skater sat in the box based on the penalty box paperwork)
Period Abosulte	Based on the current time displayed on the period clock. This does not require that the period clock counts in any specific order - the direction of the period clock is specified in the ruleset element (which defaults to counting down from the period duration if not specified). NB: This needs to factor in timeouts and other stoppages of the period clock to property convert to an absolute time
Estimated Range	When the exact time isn’t know, it can potentially be estimated as being between two (hopefully known) timestamps

Estimated times can occur if, for example, we lack an absolute timestamp but know that Skater 123 got a penalty in period 3, sat in the penalty box, and was still there when the jam ended after its natural conclusion and the penalty box timer stopwatch said "0:15". So from this, we can deduce that the skater sat 15 seconds before the end of the jam (this assumes we have bout times for an absolute value), so the "enter box" object timestamp would be 15 seconds before the end of the jam. Since the jam was 2:00 in duration, we know that the penalty occurred sometime between the start of the jam and when the skater sat, and can construct an estimated range accordingly.

Estimated times like these are designed to always be derived based on some sort of algorithm. An example algorithm would take the bout clock paperwork and combine it with the penalty box paperwork to get times as per above. Note that different algorithms could produced different specific times estimates. As a result, any application is free to ignore the estimated timestamps and provide its own.

Required Structure

For a DerbyJSON game data file, the following structure is required:

- "team" property *must* contain three team properties:
  - "home"
  - "away"
  - "officials"
- "period" property *must* contains properties for the number of periods as specified by the ruleset, where the property name is the period number as a string.
- Jam objects within the period object's "jams" property *must* be in the correct order (though they can be interspersed with "note" objects, for example).
- Jam objects *must* have a "number" property containing the jam's number
- Events within the jam object's "events" property *should* be in order where possible. More specifically:
  - Scoring passes that occur before the star pass of team *must* be stored before the star pass, and those after the star pass *must* be stored after the star pass.
  - Box exit objects *must* occur after their corresponding box enter objects (and both *should* occur after the corresponding penalty)

- If accurate time stamps are used, the events *should* be sorted in time stamp order
- Specific instances of instances in time (such as the "date" property of a note object), unless otherwise specified, are in ISO-8601 format (YYYY-MM-DDTHH:mm:ss.sssZ). This is the standard format that Date.prototype.toJSON() creates. This is not to be confused with time stamps, which are relative to a specific game (or part if a game).

Object Specifications

This section covers the various objects that are currently defined for DerbyJSON. Remember that is it not expected to have a file contain all of these objects, or for all properties of the object to be defined.

(Root) Object

The root object represent an entire game, or a collection of other useful data (such as rosters for an entire league, etc…). The root object can contain the following properties:

Property	Type	Details
version	string (recommended)	The current version of the file format
metadata	metadata object	A generic object containing meta data
type	string (default=game)	Defines what sort of data is contained in this. Options include: • game • rosters • stats • leagues
teams	object of team objects (required for type=game, type=rosters)	Contains the teams involved in the game/etc. The property names are the unique ids for the team, or the pseudo-unique ids "home", "away", "officials"
periods	object of period objects (required for type=game)	Property names are typically "1" and "2"
ruleset	ruleset object	Defaults to current WFTDA ruleset
venue	venue object	Venue at which the game is played
uuid	array of string	Unique identifying strings identifying this game
notes	array of note objects	Any notes associated with the file/game as a whole
date	string	String containing the date of the game, in ISO-8601 standard format
time	string	Start time of the game, in ISO-8601 standard format
end_time	string	End time of the game, in ISO-8601 standard format
leagues	array of league objects (required for type=league)	Information about multiple leagues
timers	object of timer objects	Used to define specific behavior of timer prototypes. The keys include: • countdown (used to count down "time until derby") • period • halftime • jam • lineup • penalty • timeout By default, timers are generated based on time values set by the rulesets, with timeout, penalty and lineup timers counting up, and the rest counting down.
tournament	string	Name of the tournament (if any) that this game corresponds to
host-league	league identifier	The hosting league (if a tournament, or different from "home")

Property	Type	Details
expulsions	array of expulsion objects	If an expulsion occurred during the game, list the suspensions here
suspensions	array of skater identifiers	If this game represents a suspension for a skater, listed here
signatures	array of signature objects	Used to hold the signatures of an IGRF
sanctioned	boolean (default: false)	Is the game sanctioned
association	string (default: "WFTDA")	What governing body/association the game is played for. Possible values include "WFTDA", "MRDA", "JFTDA"

action, error object

Record of action/error (as per "action and error" tracking).

Property	Type	Details
timestamp	timestamp object	When the event occurred
skater	skater reference	Reference to the skater committing the action/error
count	number	When a child object of a period (as opposed to a jam), a count of the number of actions/errors
name	string	The name of the action/error
involved	array of involved objects	If these are recorded individually, this can list the skaters involved in the action/error

Records one of the actions found in the action sheet or the jammer action section of the error sheet (which are actually actions), or error. Actions/errors can be recorded as they happen (and include other involved in the action via the involved child object) in which case they are child objects of the jam object. Those objects should also have a timestamp property (and any count properties should be ignored and treated as a single occurrence of the action). Otherwise, if used as a "summary" (as gathered from the stat-book), count properties refer to the total occurrence (and default to zero).

call object

Used to record when a jam is called off by a skater or official (as opposed to running to its natural conclusion at two minutes, or due to an injury)

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the lap occurs
skater	skater reference	Skater calling the jam
official	string	used only if the jam was called by the official for special cases. Possible values include: “malfunction”, “major”, “interference”, “debris”, “too many skaters”, “fighting”, “unknown”, “other”

If the skater/team reference property is missing, it is assumed that the jam has been called off by whomever was the lead. If the given skater/team isn't lead, then this would normally also result in an illegal procedure penalty (and those properties aren't optional). If the jam ends with an injury, the injury object should be used. A jam that runs to the natural two minute conclusion would not have either object. If the jam ended due to one of the exceptional cases where an official can end the jam prematurely, the official property would be set with the possible case, and a notes object should include additional explanatory text.

---

certification object

Records a certification for an official, bench coach, announcer, etc..

Property	Type	Details
association	string	The name of the association providing the certification
certification	string	The type of the certification
level	number	The certification level
endorsement	string	Endorsement

This can record both certifications and endorsements. For WFTDA certifications, the certification would be either “Skating Official" or “Non Skating Official” (with the level property containing the property). Endorsements would have an endorsement property with the position instead of a certification.

---

enter box object

Represents the event when a skater enters the penalty box (and sits)

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the skater sits
skater	A skater reference object	One of the various forms of skater references indicating the skater who sat in the penalty box

Property	Type	Details
duration	number	The number of seconds the skater should sit (defaults to "30" or some multiple of there of).
substitute	A substitute object	If a substitution needs to be made, this refers to the new skater (while the "skater" property refers to the person whose penalty is being served
notes	array of notes objects	Any additional notes about the box trip

As per lineup paperwork instructions, this refers to the moment that the skater has begun timing their penalty by being seated

---

exit box object

Represents the event when a skater exits the penalty box

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the skater is released/leaves
skater	skater reference object	One of the various forms of skater references indicating the skater who sat in the penalty box
duration	number	The amount of time the skater actually spent sitting in the box (default to the duration property of the corresponding enter-box object)
premature	string	If the skater left the box prematurely: • "official" to indicate if the official released them early by mistake; • "skater" if the skater left early (and thus should have a corresponding illegal procedure call with the same timestamp); • "rescinded" to indicate that the skater was released due to the underlying penalty being rescinded as a result of an official review or other official consultation; • "mistake" if the skater was sitting in the box by mistake (such as hearing a warning and assuming it was a penalty) - this can also be used even if the entire penalty duration was served
no-skater	boolean	If the skater who was serving the time fouled out, was expelled, or injured (or otherwise was unable to serve the time), this indicates that there was no skater that actually returned to play at the end of the penalty. This will happen if the penalty is timed entirely within the jam (otherwise the enter-box will have a substitute child object)

Exit box refers to the event when a skater leaves the box (for whatever reason), and should have a corresponding "enter box" object previously recorded. Note that for jammer swaps, the first jammer's enter box object will have a duration of 30, with the exit box object having a duration of (say) 13. The second jammer's enter box object will then have a duration of 13 (and an exit of 13 as well).

expulsion object

Represents a expulsion of a skater due to their actions during the game

Property	Type	Details
skater	skater reference	The skater that was expelled
suspension	boolean	The result of the suspension meeting
notes	array of notes objects	Any additional notes about the expulsion

(Additional details about the suspension meeting can be added)

ghost point object

Records a ghost point (point that is awarded without having physically passed a corresponding opposing skater)

Property	Type	Details
skater	skater reference	The skater upon whom the ghost point was scored
ghost_point	string	An optional record of ghost points - corresponds to the old stat-book ghost point codes: L - Jammer Lap Point J - Jammer in the Box B - Blocker in the Box P - Pivot in the Box N - Not on the track point O - Out of play points G - Unknown ghost point

Since ghost points are not normally tracked anymore, that property is primarily used for importing older versions of the WFTDA stat book. Assuming all the other information is collected, it is also possible to derive this property from various other objects (assuming they have time stamps). The jammer lap would be the most difficult to deduce, other than there being five points, without the jammer leaving the track (and probably also based on the pass record of the two different jammers). There are possibly cases, however, where there was a jammer lap and a no-pass/no-penalty that is recorded as four points and the lap isn't possible to deduce from pass completion timestamps.

Note that the object should have either a skater reference or type property - both aren't always required if all other objects are recorded, but should be used if possible.

injury object

Indicates that a jam was stopped due to injury

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the lap occurs
skater	skater reference	A reference to the injured skater

A leave-track object should be used for a skater who can leave the track on their own power without forcing the jam to be called prematurely. This object can also be used as a child of the time-out object to indicate that an official time out was called to deal with an injury that did not require the jam to be stopped

---

involved object

Indicates optional information about other skaters involved in a penalty

Property	Type	Details
skater	skater reference object (required)	A skater involved in an event
notes	array of note objects	Notes discussion this skater involvement with the event

Penalties and other events rarely involve a single skater - for example, for contact related penalties, there is the recipient of the impact. Assuming this information can be gathered (for example, during bout footage review), it can be recorded in the involved child object of the penalty object. This includes both team mates as well as opposing skaters, depending on the penalty. For example:

• Contact penalty - opposing skaters that impacted by the contact

• Cutting the Track - the opposing skaters as well as the team mates who were passed while out of bounds

• Stop Assist - the team mate assisted

• Multi-player Block - the team mate involved with the block, as well as the opposing skater that was blocked

This object exists primarily to provide a more structured note-like object information (and is unlikely to be created live)

---

jam object

This represents an entire jam and all events that happen during it

Property	Type	Details
number	integer (required)	The jam number
timestamp	timestamp object	When the jam started
duration	number	Length of jam, in seconds
events	array of objects jam events	An array of all the events that happened during the jam (or the moments immediately preceding or following the jam)
notes	array of notes objects	Any additional notes about the jam (for commentary not specifically related to a given event)

Events that occur between jams should be stored with the relevant jams (as per the standard practices regarding when to record penalties and recording penalty box entry between jams)

jam event object

Jam event objects are a "base class" for all the events that can happen during a jam

Property	Type	Details
event	string (usually required)	The actual type of event: line up, pack lap, penalty, pass, star pass, action, error, lead, lost lead, call, enter box, exit box, box time, leave track, return track, injury, note
skater	A skater reference string	The skater that was involved in this event, if any
team	A team reference string	The team involved in this event, if any. Note that a skater reference can be sufficient to derive what the team is, but in some cases there is a need for the team information
timestamp	A timestamp object	When the event occurred, if know and applicable

Note that the "event" property normally defines what kind of event this object represents. There are a few cases where this can be omitted (such as action events within the "actions" property of a period). Also note that team is optional if the skater is specified

lead object

Corresponds to a jammer obtaining lead jammer status

Property	Type	Details
timestamp	timestamp object	When the jammer was declared lead
skater	skater reference	The jammer declared lead

While in theory one could simply use the a team reference, it is more efficient for latter analysis to explicit indicate the skater who gains lead. Note that this object is independent of the (initial) pass object, since lead jammer status occurs before the initial pass is completed.

league object

Represents a league object

Property	Type	Details
name	string (required)	The user visible full name of the league (this is assumed to be globally unique)
abbreviation	string (recommended)	The abbreviation for the league name (not globally unique, or even assumed to be unique within the file)
uuid	array of strings	Unique identifiers for this league. It is assumed that the name is also usable as a unique identifier
venue	venue object	"Home venue" object
teams	array of team objects	The various teams of a league
logo	logo object	Array of logos for a league (note that this will probably be the same as the logo for the "all star" team of the league)

Leagues are optional objects inside a data game file, but could be used in a repository of leagues/teams/skaters

leave track object

Records the moment a skater leaves the track to be counted as a "not on the track" point

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the leaving occurs
skater	skater reference object	One of the various forms of skater references indicating the skater who left the track
reason	string	The reason for leaving the track: "penalty" (default if not specified), "injury", "malfunction", "other"
opposing-pass	number	The number of the pass the opposing jammer is on (as per "old" lineup tracking)

This event doesn't actually correspond to anything that is currently tracked.

line up object

Line up object records who is lining up and in what position

Property	Type	Details
skater	Skater reference object	One of the various forms of skater references
start_in_box	boolean	boolean indicating that the given skater started in the box
position	string	One of the following possible values: “jammer”, “pivot”, “blocker”

(Note that this event doesn’t normally have a timestamp).

Line ups could be tracked in a variety of ways (one object for each team, and then five objects, one per position, for example), or the blockers could be enumerated (“blocker-1”,”blocker-2”, etc... unless the pivot is “blocker-1” in which case the first blocker would be “blocker-2”). By having one entry, despite having factorable team IDs, there are the fewest assumptions on how blocker information is processed (i.e., if a pivot isn’t fielded, is the skater “blocker-1” or “blocker-5”). This also slightly flattens out the structure (which could lead to a slight obfuscation), and also removes any ordering constraints of where line up information appears within the jam information. The team/skater properties then mirror the same properties in other jam events.

The start-in-box flag should be optional (there should be information that can be derived from the box trip events), but it is an imperfect world, and this flag will correspond directly to the “S” or “|” entry in the lineup sheet.

logo object

Used to represent logos of various sizes and styles

Property	Type	Details
small	url to the “small” logo	Ideally should be designed to fit in a 32x32 area
medium	url to the “medium” logo	Ideally should be designed to fit in a 128x128 area
large	url to the “large” logo	Ideally should be designed to fit in a 512x512 area
url	url to the base logo	If there is only one logo, this size should be used and it will be resized up or down as needed

Note that all logo properties can come in three variants: “light”, “dark”, “grayscale”. These are signified by appending an underscore and variant name (e.g., “medium_dark”. Light logos are used when the logo has is designed to appear on a light background. Dark logos are used when

the logo has been designed to appear on a dark background. Grayscale logos are for grayscale print display (and assumed to appear on a white background).

Note also that the logo sizes are approximate, and that, if needed, larger logos (of the appropriate style) will be rescaled down as needed, or up if absolutely required. If size variants are present, the single "url" property should not be present.

---

lost lead object

Corresponds to a jammer losing lead jammer status (or the ability to gain lead jammer status).

Property	Type	Details
timestamp	timestamp object	When the jammer lost lead
skater	skater reference	The jammer losing lead

While in theory one could simply use the a team reference (or if a lead object is present in the jam, omitted all together), it is more efficient for latter analysis to explicit indicate the skater who lost lead.

---

metadata object

Metadata contains additional information about the file that isn't relevant to the game data itself

Property	Type	Details
producer	string	The name of the application that created the game data
date	string	A date string of the time the file was created (which may not be the time of the game)
author	string	Name of person who created the game file
comments	string	Comments about the game data (not to be confused with notes about the game)

---

note object

Notes contain general information to describe data that is outside of the normal

Property	Type	Details
note	string (required)	The actual comment
author	string	Who write the note

Property	Type	Details
date	string	The time/date that the note was written, in ISO-8601 format (note that this is not a time stamp, since the note is often written independent of when the actual even the note refers to is took place
notes	array of note objects	Notes can contain notes that are in reference to themselves, allowing for threaded tree-like commentary

Note objects are normally stored in a “notes" property (as an array) when commenting on a single piece of data, but can also be found mixed in with other heterogeneous data (and can be identified by the “note" property).

---

pack lap object

Records a lap of the pack (such as is recorded on the bout time sheet). There can be multiple pack-lap objects for each lap, or optionally a single object with the total

Property	Type	Details
timestamp	timestamp object	When the lap happened (if counted individually)
count	number	Optional number of laps (or fraction of a lap) - if missing, the default value is “1”. The total pack laps for jam are the sum of all pack-lap count properties.

If only the total number of pack laps is recorded, time stamps do not provide a useful metric.

---

pass object

Corresponds to a completed pass (either initial or scoring), or a pass that wasn’t completed when the jam ended

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the lap occurs
number	number	Indicates the pass number of a completed pass
points	number	Number of points scored in the pass
skater	skater reference	The skater who scored the points for this pass
completed	boolean	A boolean indicating if the pass was completed (defaulting to “yes”)
ghost_points	array of ghost point objects	Optional array of ghost points that were included in this pass

pass objects are used to record both initial passes as well as subsequent scoring passes, and correspond to the point in time when the pass is completed (the skater exists the front of the

engagement zone). Non-completed passes would be recorded by having a completed property set to "no". A "no pass" can either be recorded as having no pass objects at all, or a pass object with number set to "1" and completed set to "no" (due the the way that no pass is handled with star passes, it is possible to have two such objects - one for the old jammer, one for the new jammer).

It is possible to record a (jammer lap) point in the initial pass by having a pass object with the number property set to "1" and the score property set to "1". Note that this event is independent of the lead (or loss-lead) objects, since it is possible to get lead but not complete an initial pass. If the skater clear the pack before the other skater gains lead and is not lead (due to a no-pass/no-penalty) there would be both a loss-lead and a pass object.

In the extremely rare case of getting a jammer lap point but not completing the initial lap, using a pass object with a number property set to "1", completed set to "no", and a score property set to "1".

---

penalty object

Records a penalty

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the lap occurs
skater	skater reference object	One of the various forms of skater references
penalty	string (required)	Letter code corresponding to the penalty
severity	string	Optional severity indication: • "no" for no-pass/no-penalty or no-impact; • "minor" for old minor penalties; • "major" for major penalties (default if not specified); • "expulsion" for expulsion penalties
rescinded	boolean	A boolean value that indicates that the penalty was rescinded due to either an official review or other official consultations (and ideally there should be a note or official review)
involved	array of involved objects	Array of other skaters that were involved in the penalty
cue	string	The verbal cue used for the penalty (to differentiate the various "illegal procedure" penalties, for example) as per the standard practice verbal cue. Additional details can be placed in a note objects if needed

The skater reference is the person for whom the penalty is recorded (so for some illegal procedures, this may be a captain who actually was not involved in the incident). In such cases,

additional information can be presented in note objects. Record of other skaters directly involved in the penalty is presented in the involved property.

period object

This represents everything that happens from shortly before a period starts to shorting after it concludes.

Property	Type	Details
timestamp	timestamp object	The start of the period
end	timestamp object	The end of the period
jams	array of jam, clock stoppage and note objects	Despite the name "jams" this contains an array of jams as well as inter-jam events such as timeouts. The actual type of the object is determined based on the presence of specific keys: • jam objects contain the "number" property • notes contain the "note" property • time out (clock stoppage) events contain "timeout" properties
actions	array of action objects	The actions for this period
errors	array of error objects	The errors for this period

If actions and errors can be gathered "real time" they belong with their respective jam events. If this data is gathered from a StatsBook which doesn't contain that information, the actions and errors are stored in their respective period

person object

Represents a single individual (regardless if player or official, on skates or off)

Property	Type	Details
name	string (required)	The (derby) name of the person
number	string (required for skaters in a game)	The number used by the skater (for a player). This is the value that is used when populating paperwork.
league	string	For officials in a game, this is their league affiliation
certifications	array of certification objects	Official's certification and endorsements
legal	string	The legal name of the person (if different from the derby name)

Property	Type	Details
roles	array of strings	What role that person plays on the team/game. For a player on a team in a game, this is usually omitted, though "captain" and "alt" are possible. For officials, this should be the standard abbreviation for their role(s). For general roster information, other possibilities such as "coach" or even "mascot" could be used. Also note that this only refers to the roles that they are performing either by default or in this one specific game
skated	boolean	Indicates that the person actually skated in the game (default to "true" - used to indicate an alt not skating)
uuid	array of strings	Other unique identifiers that can be use to link this person on this team in this game to the same person in another game (even if it is a different team).
insurance	string	Insurance number

Person represents a reference to an individual in a given context (such as "on this team in this game"). Note that a given individual can play multiple roles across multiple teams/leagues over time - it is assumed that any and all "true" uuid values that are shared are actually referring to the same person (and are thus actually unique). Values that are not formatted as a uuid (e.g., not of the format "29F68444-9CBC-4BBB-89C6-97264AD48674") are assumed to be locally unique only.

Because this refers to not just playing skaters but others as well, it is not referred to as a "skater" object.

---

return track object

Records the moment a skater returns to the track

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the leaving occurs
skater	skater reference object	One of the various forms of skater references indicating the skater who left the track
opposing-pass	number	The number of the pass the opposing jammer is on (as per "old" lineup tracking)

This should pair with a preceding "leave track" object

---

ruleset object

Represents the rules that the game is played under

Property	Type	Details
version	string	Textual identifier of name of the rule set in the format: MMMYYYY
period-count	number	Number of periods (default "2")
period	string	Length of period (default "30:00')
jam	string	Length of jam (default "2:00")
lineup	string	Length of lineup (default "0:30")
timeout	string	Length of team timeout (default "1:00")
timeout-count	number	Number of team timeouts, total in bout (default: "3")
official-review-count	number	Number of official reviews per period (default: "1")
official-review-retained	boolean	Is the official review retained if valid (default: "true")
official-review-maximum	number	Limit on number of official reviews per period (default: "2")
penalty	string	Length of penalty (default: "0:30")
minors	boolean	Are there minor penalties? (default: "false")
minors-per-major	number	How many minors are a major (default: "4")
foul-out	number	The number of penalties that a skater can accrue before fouling out (default: "7")

The ruleset property defaults to the current (January 2015) ruleset, with 30 second periods, and retaining official reviews (if the team has a valid objection).

star pass object

Records an attempted or successful star pass.

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the star pass occurs (ideally when the helmet cover is removed from the jammer's head)
skater	skater reference	Reference to the jammer from whose head the helmet cover was removed
completed	boolean (default: true)	Indicates if the star pass was completed

Property	Type	Details
failure	string	Present if the star pass fails, with either a value of "jammer" or "pivot" to indicate who was at fault for failing to perform the star pass (with a note describing more detail).

There should be a corresponding loss-lead event object for this if the jammer was lead. This object doesn't quantify all the possible details of the star pass, but explicitly recording the reason for the failure is a first attempt. Obviously a failed star-pass can't be easily determined by recreating a stat-book.

A "star stash" would be recorded with the completed property set to false (and no "failure" object recorded)

---

substitute object

Indicates that a skater was substituted for another skater in the penalty box

Property	Type	Details
skater	skater reference	One of the various forms of skater references indicating the skater who actually will serve the penalty time in the box for the skater indicated in the parent enter-box object
reason	string	If a substitute is needed, this should explain why. Possible values include: • injury • foul out • expulsion

If a skater fouls out, or is injured with a penalty due, this object indicates that another skater was substituted for this skater the start of the next jam (If a penalty finishes within the same jam without a skater substituting, this object will not be present). The reason property can normally be derived by looking at the most recent penalty object of the corresponding parent enter-box skater reference, but this explicitly indicates that they left the box due to an injury.

[NB: This scheme still causes a few problems and will need to be addressed later]

---

team object

Represents a single group within a league/game

Property	Type	Details
name	string (required)	The name of the team - required to be unique within the league (when combined with the date field, if any). Note that this potentially is an empty string for leagues that only have a single (travel) team.

Property	Type	Details
league	league identifier string (required for teams in a game data file, not for teams within a league object)	The name of the league that the team is a part of. This can either be a full name or some sort of UUID. Note that for a "mixer" game, this key isn't actually required)
abbreviation	string (recommended)	The abbreviation, if possible, for team's name. Note that teams such as "All Stars" do not have an abbreviation
persons	array of person objects	The members of a team
level	string (recommended)	What sort of team this represents: • All Star (primary rostered travel team) • B (secondary travel team) • C (tertiary travel team) • Rec (recreational team) • Officials (a group of officials) • Home ("home" team) • Adhoc - mixer type team
date	string (DDMMMYYYY)	When this roster was last know to be correct. For a game this field probably isn't used (since the value can be derived from the game's play date), but for a "master roster" document, it supports having multiple seasons of teams.
color	coloring object	The colors for the team, both for the scoreboard as well as for jersey information
logo	logo object	Referencers to logos of various appropriate sizes

Teams represent any collection of people representing the league - both fixed rostered teams as well as ad-hoc teams and officials. Teams persons property can include both skaters and non skaters

---

time out object

Time out objects are used to represent any time the period clock is stopped

Property	Type	Details
timeout	team reference (required)	This is the team that requested the stoppage of the period clock ("home", "away", or "official")
notes	array of note objects	Any extra descriptions of the clock stoppage (does not include the challenge or resolution of an official review)
injury	skater reference (injury only)	Indicates that the clock was stopped for the injury of this skater. Note that if the jam was stopped, there will also be a corresponding injury event in the jam. The "timeout" property should reflect the team that the skater belongs to
duration	number (in seconds)	How long the clock was stopped (including additional "lineup" time afterwards)

Property	Type	Details
timestamp	timestamp object	When the timeout occurred
review	string (OR only)	Description of what was to be reviewed
resolution	string (OR only)	Description of resolution of the review
retained	boolean (required for OR only)	If the official review was retained by the team

The various clock stoppage events are handled thusly:

- Team time out: "team" is the appropriate team, "duration" is some value between 65 and 90 (65 being a 1:00 time out plus 5 second warning before starting a jam - after a team time out the maximum duration for a lineup is 30 seconds)
- Official time out: "team" is "official"
- Official review: "team" is appropriate team, "retained" filled in accordingly, ideally "review" and "resolution" filled in
- Injury: "injury" skater reference required

---

timer object

Timer objects provide definitions for

Property	Type	Details
duration	number	How long, in seconds, the timer will run before it stops
counts-down	boolean	If 'yes', then the timer counts down from the duration value, otherwise it counts up
running	boolean	Is the timer currently running

While most timing information can be derived from other object information (such as rule set values, for example), this object can be used to define the direction of time shown (count up vs count down).

---

timestamp object

Timestamps are used to mark time - either absolute or estimated, and based on either the wall clock, period clock, or jam clock.

Property	Type	Details
wall	string	Time on the wall clock, in 24-hour "HH:MM:SS" (preferred)
epoch	number	Time in seconds, since the start of the unix epoch

Property	Type	Details
period	string	"MM:SS" in the period clock. Note that this must factor in both the period clock direction and various time outs to be able to convert to an absolute time
seconds	number	The number of seconds relative to the start of the period
jam	number	A signed number indicating the number of seconds from the start of the jam, or, if negative, from the end of the jam
approximate	boolean	Indicates that the time range is a "best guess" - used for reconstructing game data that doesn't have accurate time stamps
range	array of 2 timestamp objects	Indicates that the event happened somewhere between two time periods - used for reconstructing game data that doesn't have accurate time stamps

Except for "approximate", a timestamp object should only have one property. If multiple properties are provided, the must be consistent (it is implementation dependent as to which one is used).

---

venue object

venue is used to describe a location where roller derby is played. It is a superset of the location information (venue name, city, state/provence). venue can be a top level object. It is based on "address" objects of DocBook

Property	Type	Details
name	string (required)	Name of the venue (assumed to be globally unique, but unfortunately not guaranteed)
city	string (required)	The city the venue is in
state	string (required)	The state the venue is in
url	string	A url of the website of the venue
country	string	The country the venue is in
email	string	An email address for the venue
fax	string	The fax number of the venue
otheraddr	string	A secondary address for the venue
phone	string	The phone number of the venue
pob	string	The post office box number of the venue
postcode	string	Post code for the venue
street	string	The primary street address of the venue

Property	Type	Details
notes	array of notes objects	Any additional notes about the venue
uuid	array of strings	Array of unique identifiers of the venue
logo	logo object	A visual branding object used to represent the venue

XXX object

xxx

Property	Type	Details
timestamp	timestamp object	a timestamp indicating when the lap occurs

xxx