Last update Apr 5, 2024

Overview

The interface is based on the exchange of JSON-encoded files in a folder on a computer (Mac or PC)

The interface consists of 3 different categories of files:

1) Timing system status file
2) Meet program
3) Results files

The file folder has to be selected by the user in both the timing system and the meet management system separately. The interface will only work if the same folder is selected in both programs (e.g. “C:\TimeDrops\Invitational-2023-07-20\”)

Timing System status file

Upon initialization, the Time Drops system will generate a file “time_drops_status.json” with the following content:

```
{
   "currentEvent": "1",
   "currentHeat": 1,
   "currentSessionNumber": 123,
   "currentSessionId": "67251",
   "protocolVersion": "1.0.0",
   "timingSystemType": "Time-Drops",
   "timingSystemVersion": "2023.04.787.debug",
   "lastMeetProgramDate": "2023-07-29T16:33:41.922",
   "updatedAt": "2023-07-31T17:32:31.725-07:00"
}
```

This file will be updated whenever any of these events occur:

- The meet program is updated from the meet management system

- The system starts a new heat
- The timing system application is started or re-started

Meet program:

The meet management system writes a file to the folder with the name “meet_program.json”. This file contains JSON per the following Kotlin class definition:

```
@Keep
class MeetProgramJson {
   @Keep
   data class Meet(
       val meetName: String,    // Name of the meet as assigned by the user
       val meetProgramVersion: Int = 1,  // Data format version, must be 1
       val meetProgramDateTime: String, // Date/Time the program was last updated, in ISO 8601
UTC format
       val meetHostTeamName: String, // name of the team or organization hosting the meet
       val meetStartDate: String,  // First day of the meet, in ISO 8061 DAY ONLY format (e.g.
"2023-07-31")
       val meetEndDate: String?, // Last day of the meet, in ISO 8061 DAY ONLY format (e.g.
"2023-07-31")
       val meetEvents: List<Event>, // list of events for the entire meet (see class Event
below)
       val meetSessions: List<Session>,  // list of Sessions for the meet, must contain at least
one active session, see class Session below
       val meetTeams: List<Team>,   // list of teams participating in the meet see class Team
below
       val meetSwimmers: List<Swimmer>, // list of swimmers participating in the meet, see class
Swimmer below
   )

   @Keep
   data class Pool (
       val poolNumberOfLanes: Int, // number of lanes used for this meet
       val poolCourse: String, // LCM, SCM, or SCY
       val poolFirstLaneNumber: Int, // index of the fist lane, usually 0 or 1
   )

   @Keep
   data class Team (
       val teamId: String, // unique identifier for the team
       val teamAbbreviation: String, // 3-5 character abbreviation for the team (for display on
the scoreboard) - required
       val teamShortName: String, // Short name of the team (required)
       val teamFullName: String?, // Long name of the team (optional)
       val teamMascot: String?, // team mascot, e.g. "Penguins", optional
   )

   @Keep
   data class Session(
       val sessionNumber: Int,  // number of the session within the meet
       val sessionId: String,   // identifier of the session - results files will be tagged with
this identifier
       val sessionName: String?,  // name of the session, e.g. "Saturday Prelims" - required
       val sessionBeginAt: String, // DateTime of the expected start of the session in ISO 8601
zoned time, e.g. "2023-08-01T05:32:29-07:00"
       val sessionEndAt: String, // DateTime of the anticipated end of the session in ISO 8601
zoned time
       val sessionRaces: List<Race>, // List of Races scheduled for this session, see class Race
below
       val sessionPool: Pool, // specification for the pool this session is run in
       val sessionIsCurrent: Boolean, // Each meet must have exactly one current session
   )
   @Keep
   data class Event (
       val eventNumber: String,  // number of the event, e.g. "3" or "4A"
       val eventStrokeCode: Int, //    1 -> Free  2 -> Back  3 -> Breast 4 -> Fly  5 -> Medley
       val eventStroke: String?, // Freeform description of the stroke, can be used for
localization
       val eventIsRelay: Boolean, // true for relays
       val eventRelaylegs: Int?, // the number of swimmers in a relay, typically 4
       val eventDistance: Int, // in meter or yard depending on pool course. For relays, specify
total distance,
       val eventGender: String, // "M", "F", or "X" or any other category designation
       val eventMinAge: Int,  // minimum age for event, specify 0 for "&under" or open events
       val eventMaxAge: Int,  // maximum age for event, specify 0 for "&over" or open events
       val eventDescription: String?, // description is basically a duplication of all the
previous information - might still be useful
       val eventShortLabel: String?,
       val eventFullLabel: String?,
       val eventRecords: List<Record>, // list of records applicable to this event. can be empty
       val eventStandards: List<Standard>, // list of time standards applicable to this event.
can be empty
   )

   @Keep
   data class Record(
       val recordCompletedDate: String?, // Date current record was set in ISO 8601  Date Only
format (e.g. "2012-03-16")
       val recordSetName: String, // name of record, e.g. "Team Record"
       val recordSetCode: String, // 1-character tag for the record, e.g. "T"
       val recordTimeInt: Int, // all times are encoded in 1/100th of seconds,
       val recordHolderName: String?,
       val recordHolderTeam: String?,
       val recordEligibleTeamIds: List<String>, // list of teams which are eligible for this
record
   )

   @Keep
   data class StandardTier(
       val tierName: String, // name of the tier, e.g. "Gold", "Silver", etc.
       val tierCode: String, // 1-character tag for the standard on the Scoreboard, e.g. "G", S"
       val tierTime: Int, // in 1/100 of a second
       val tierEligibleTeamIds: List<String>,
   )

   @Keep
   data class Standard(
       val standardName: String, // name of the Standard, e.g. "Motivational"
       val standardTiers: List<StandardTier>,
   )
```

```
  @Keep
  data class Race(  // commonly called a heat - Race is slightly different as it defines the
specific order of races in the program
      // IMPORTANT: the race number defines the order of races at the meet irrespective of the
event number and heat number
      // therefore it is possible to have preliminaries and finals out of order
      val raceEventNumber: String?,
      val raceNumber: Int, // race number as generated by the program - actual race number for
the results can be different due to false starts etc. The List of races in the session should be
strictly monotonic with respect to the race number
      val raceHeatType: String, // for Prelims, Finals, Swimoffs etc.
      val raceHeatNumber: Int, // number of the heat
      val raceTotalHeats: Int, // this is strictly for display, e.g. "Heat 1 of 3"
      val raceLanes: List<Lane>,
  )

   @Keep
  data class Lane(
      val laneNumber: Int, // number of the lane, starting at 0 or 1 depending on the pool
setup
      val isEmpty: Boolean, // lane is seeded empty, all other values should be null
      val laneEventNumber: String? = null, // for the purpose of combining events, each lane
has it's own event number
      val laneSeedTime: Int? = null, // seed time in 1/100 of a second
      val laneSwimmerId: String? = null, // nullable type, will be null in relay events and
have a value in individual events
      val laneRelayTeamName: String? = null, // used in relay events - something like "RSM A" -
OR - just "A" and then combine with the team name/abbreviation
      val laneRelaySwimmerIds: List<String?>? = null, // for relay events - list of swimmers in
this relay - null for individual events
      val laneTeamId: String? = null, // Team association of this entry
  )


  @Keep
  data class Swimmer(
      val swimmerId: String?, // unique id of each swimmer (not necessarily globally unique,
but must be unique within the meet
      val swimmerName: String?, // name of swimmer in the preferred order (first last or last,
first)
      val swimmerGender: String?,
      val swimmerAge: Int?,  // per age up date
      val swimmerTeamId: String?, // references list of teams
  )
}
```

The program file should be updated in any of the following conditions:

- At the start of the meet
- Whenever there have been any changes to the events/heats/swimmers, such as:
    - Scratches
    - Substitutions
    - Lane changes
    - Addition or removal of heats

- Adding of events (swim-offs etc.)

After each update of the meet_program.json, the time_drops_status.json will be updated to reflect the current session number and session id which matches the ‘active’ session in the program

Result files

After each heat is completed, Time Drops will write a result file with the name

“SessionSSSS_Event_EEEE_HeatHHHH_RaceRRRR_XXX.json

where SSSS is the session ID, EEEE the heat number, HHHH the heat number, RRRR the race number and XXXX the revision number

The numbers/IDs are not padded with leading 0. Note that Time Drops has the internal ability for the user to make adjustments and corrections to the results. A result that has already been written can be update later. For each update, the XXX revision number will be increased. The Meet Management system should only consider the highest revision number to be the currently valid result.

The results .json files will have the following content:

```
{
  "createdAt": "2023-07-31T17:53:00.042-07:00",
  "protocolVersion": "1.1.0",
  "sessionNumber": 123, // from program
  "sessionId": "67251", // from program
  "eventNumber": "1",
  "heatNumber": 2,
  "raceNumber": 2,
  "startTime": "2023-07-31T17:52:11.234-07:00",
  "lanes": [
    {
      "lane": 1,
      “finalTime”: 4883, // combination of the pad and button times. Meet Management program
may prefer to calculate this time themselves
```

```
      "padTime": 4883, // only present when using pads
      "timer1": 4942, // in 1/100th of seconds
      "timer2": 4925,
      "timer3": 4904,
      "isEmpty": false, // lane declared empty by timing operator, may still have times in case of
operator error
      "isDq": false, // for relay judging platform
      "splits": [
        {
          "distance": 25,
          "time": 1686
        }
      ]
    },
   …… more lanes to follow
  ]
}
```

APPENDIX A

Example of a (minimalistic) meet_program.json with a single heat of 200 backstroke:

```
{
    "meetName": "My Meet Title",
    "meetProgramVersion": 1,
    "meetProgramDateTime": null,
    "meetHostTeamName": "My Team",
    "meetStartDate": "2019-02-03",
    "meetEndDate": null,
    "meetEvents": [
        {
            "eventNumber": "10",
            "eventStrokeCode": 2,
            "eventStroke": "BACK",
            "eventIsRelay": false,
            "eventRelaylegs": 1,
            "eventDistance": 200,
            "eventGender": "F",
            "eventMinAge": 10,
            "eventMaxAge": 0,
                "eventDescription": null,
                "eventShortLabel": null,
                "eventFullLabel": null,
                "eventStartTime": null
            }
        ],
        "meetSessions": [
            {
                "sessionNumber": 4,
                "sessionId": "4",
                "sessionName": "Session 4",
                "sessionBeginAt": "2019-02-03T00:00:00-08:00",
                "sessionEndAt": null,
                "sessionRaces": [
                    {
                        "raceEventNumber": "10",
                        "raceNumber": 1,
                        "raceHeatType": "F",
                        "raceHeatNumber": 1,
                        "raceTotalHeats": 1,
                        "raceLanes": [
                            {
                                "laneNumber": 2,
                                "isEmpty": false,
                                "laneEventNumber": "10",
                                "laneSeedTime": 21500,
                                "laneSwimmerId": "8",
                                "laneTeamId": "4409"
                            },
                            {
                                "laneNumber": 3,
                                "isEmpty": false,
                                "laneEventNumber": "10",
                                "laneSeedTime": 20000,
                                "laneSwimmerId": "29",
                                "laneTeamId": "4561"
                            },
                            {
                                "laneNumber": 4,
                                "isEmpty": false,
                            "laneEventNumber": "10",
                            "laneSeedTime": 21000,
                            "laneSwimmerId": "24",
                            "laneTeamId": "4561"
                        }
                    ]
                }
            ],
            "sessionPool": null,
            "sessionIsCurrent": true
        }
    ],
    "meetTeams": [
        {
            "teamId": "4409",
            "teamAbbreviation": "MDW",
            "teamShortName": "Meadow",
            "teamFullName": "Meadow",
            "teamMascot": null
        },
        {
            "teamId": "4561",
            "teamAbbreviation": "FOR",
            "teamShortName": "Forest",
            "teamFullName": "Forest",
            "teamMascot": null
        }
    ],
    "meetSwimmers": [
        {
            "swimmerId": "8",
            "swimmerName": "Doe, Jane",
            "swimmerGender": "F",
            "swimmerAge": 11,
            "swimmerTeamId": "4409"
        },
        {
            "swimmerId": "24",
            "swimmerName": "Montana, Hanna",
            "swimmerGender": "F",
            "swimmerAge": 15,
            "swimmerTeamId": "4561"
        },
        {
            "swimmerId": "29",
            "swimmerName": "Jackson, Luisa",
            "swimmerGender": "F",
            "swimmerAge": 12,
            "swimmerTeamId": "4561"
        }
    ]
}
```