|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Size** | **Initial Value** | **Must Have?** | **Comments** |
| // metadata |  |  |  |  |  |
| compName | Str | 100 chars | “” | y |  |
| matchType | str | <50 chars | “” | y | Options(“Test/Practice”, “Qualifications”, “Quarters”, “Semis”, “Finals”) |
| matchNumber | Int | 1000 (+-) | 0 | y | 0 is an illegitimate value (practice rounds and such things can be negative numbers) |
| teamNumber | int | 6 digits decimal (+-) | 0 | y | O is an illegitimate team number. Generally team numbers should be 4 digits unless there are very special circumstances |
| scouterName | str | 30 chars | “” | y |  |
| whenCaptured | datetime | Size of date0 | “1989-03-20 00:00:00” |  | initial value is first founding date (time is 0…) |
| scouterTeamNumber | int | 6 digits decimal (+-) | 0 | y | See “teamNumber” |
| driverStation | int | Small int (< 1000) | -1 |  |  |
| // autonomous |  |  |  |  |  |
| startPlace | Int? | Small int (< 1000) | -1 |  | From list |
| ballsInUpperAuto | int | Small int (< 1000) | 0 |  |  |
| ballsInLowerAuto | Int | Small int (< 1000) | 0 |  |  |
| ballsMissedAuto | int | Small int (< 1000) | 0 |  |  |
| passedLine | bool |  | false |  |  |
| ballsHumanShotAuto | bool |  | false |  |  |
| ballsHumanScoredAuto | bool |  | false |  |  |
| whichBallsCollected | List of Int? | Small int (< 1000), list size by size of options? | [] |  | From list |
| autoMalfunction | bool |  | false |  |  |
| autoFreeText | str | <500 | “” |  |  |
| // tele-op |  |  |  |  |  |
| ballsInUpperTele | int | Small int (< 1000) | 0 |  |  |
| ballsInLowerTele | Int | Small int (< 1000) | 0 |  |  |
| ballsMissedTele | int | Small int (< 1000) | 0 |  |  |
| // end game |  |  |  |  |  |
| levelClimbed | int | Small int (< 1000) | 0 |  |  |
| climbSuccessful | bool |  | true |  |  |
| climbTime | float | Small int (< 1000) | 0 |  | Units = sec |
| // post game |  |  |  |  |  |
| defensiveDefenseLevel | Int | Small int (< 1000) | -1 |  | 1 - 7 |
| offensiveDefenseLevel | Int | Small int (< 1000) | -1 |  | 1 - 7 |
| wasDefendedLevel | Int | Small int (< 1000) | -1 |  | 1 - 7 |
| shootingLocations | List of Int? | Small int (< 1000), list size by size of options? | [] |  | From list |
| collectingLocations | List of Int? | Small int (< 1000), list size by size of options? | [] |  | From list |
| goodTeamMateLevel | Int | Small int (< 1000) | 4 |  | 1 - 7 |
| wasBroken | Int | Small int (< 1000) | -1 |  | 1-3 (by seriousness of damge) |
| freeText | str | <500 | “” |  |  |
| generalImpression | Int | Small int (< 1000) | 4 |  | 1 - 7 |
| // post game flags |  |  |  |  |  |
| robotNoFunction | Bool |  | false |  |  |
| systemNoFunction | bool |  | false |  |  |

Use python and the above table to help create js object with code below:

names = """names from word""".splitlines()  
  
def\_vals = """vals from word""".splitlines()  
  
for name, val in zip(names, def\_vals):  
 if name[0:2] != "//":  
 print(name + ": " + val + ",")  
 else:  
 print(name)

use python and js reducer initial state to create action types using code below

lines = """js initial state from reducer""".splitlines()  
extracted\_names = []  
for line in lines:  
 if line[4:6] != "//" and line != "":  
 name = line[4:line.find(":")]  
 extracted\_names.append(name)  
  
for name in extracted\_names:  
 print(f"export const {name}\_SET = '{name}\_set';")

create actions for reducer (SET) using python and initial state from reducer

lines = """initial state from reducer""".splitlines()  
extracted\_names = []  
for line in lines:  
 if line[4:6] != "//" and line != "":  
 name = line[4:line.find(":")]  
 extracted\_names.append(name)  
  
for name in extracted\_names:  
 if len(name) >= 2:  
 upperName = name[0].upper() + name[1:]  
 print("export function set" + upperName + "(new" + upperName + "){")  
 print("return {type: actionTypes." + name + "\_SET, payload: {" + name + ": new" + upperName + "}};")  
 print("}\n")

create cases from reducer using python and initial state from reducer

lines = """  
 initial state from reducer""".splitlines()  
extracted\_names = []  
for line in lines:  
 if line[4:6] != "//" and line != "":  
 name = line[4:line.find(":")]  
 extracted\_names.append(name)  
  
for name in extracted\_names:  
 if len(name) >= 2:  
 print("case actionTypes." + name + "\_SET:")  
 print(" return produce(state, stateCopy => {")  
 print(" stateCopy." + name + " = action.payload." + name + ";")  
 print(" return stateCopy;")  
 print(" });")

create INC and DEC action types using python and SET actionTypes

lines = """  
SET actionTypes  
""".splitlines()  
  
for line in lines:  
 print(line.replace("SET", "INC"))  
 print(line.replace("SET", "DEC"))  
 print("")

create inc and dec actions using python and set actions

lines = """set actions  
""".splitlines()  
  
  
def find\_2nd(string, substring):  
 return string.find(substring, string.find(substring) + 1)  
  
  
# must be in correct format with one line empty between actions  
for first\_line, second\_line, third\_line in zip(lines[0::4], lines[1::4], lines[2::4]):  
 t = first\_line.replace("set", "inc")  
 t = t[:t.find("(") + 1] + " numToIncrement = 1 " + t[t.find(")"):]  
  
 l = second\_line.replace("SET", "INC")  
 l = l[:find\_2nd(l, ":") + 1] + " { numToIncrement } }" + l[l.find(";"):]  
  
 x = third\_line  
 print(t)  
 print(l)  
 print(x)  
  
 t = first\_line.replace("set", "dec")  
 t = t[:t.find("(") + 1] + " numToDecrement = 1 " + t[t.find(")"):]  
  
 l = second\_line.replace("SET", "DEC")  
 l = l[:find\_2nd(l, ":") + 1] + " { numToDecrement } }" + l[l.find(";"):]  
  
 x = third\_line  
 print(t)  
 print(l)  
 print(x)  
  
 print("\n")