SoftwareX 10 (2019) 100271

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

jDSSAT: A JavaScript Module for DSSAT-CSM integration

Jonas de Abreu Resenes [a,*], Willingthon Pavan [a], Carlos Amaral Hölbig [a], José Maurício Cunha Fernandes [a], Vakhtang Shelia [b], Cheryl Porter [b], Gerrit Hoogenboom [b]

[a] Graduate Program in Applied Computing, University of Passo Fundo, Brazil
[b] Agricultural and Biological Engineering, University of Florida, Gainesville, FL, USA

ARTICLE INFO

Article history:
Received 23 April 2019
Accepted 24 June 2019

Keywords:
DSSAT
Crop simulation
JavaScript

ABSTRACT

The DSSAT is a collection of computer programs and tools integrated into a single software package in order to facilitate the application of crop simulation models in research and decision making. The DSSAT Shell, an user interface program, enables users to easily select and use any of the DSSAT components. It reads text files, both input and output with fixed width format, to provide information to the users and to be able to run the models. The logic to read DSSAT files and process the information to display to the user relies on the Shell itself and cannot be reusable by any other system, which makes it harder to implement alternatives for the DSSAT Shell since there are no frameworks available that implements the complexity of processing DSSAT files. The DSSAT tools were built using old programming technologies such Visual Basic which is in end-of-life support and Delphi, these technologies should be replaced for a modern and standardized software development approach for a better maintainability. Besides, these tools are stand-alone and they do not share code which increases the effort to maintain them. This work presents the jDSSAT, a multiplatform JavaScript module. The jDSSAT provides a standard and reusable approach for reading and processing DSSAT files. Through this approach, we isolate the complexity of processing DSSAT files to allow DSSAT integration on any environment. It also integrates with DSSAT-CSM to make it easier to run DSSAT models in Linux, Windows and MacOS. As a result, we present a multiplatform user interface prototype created to run DSSAT crop models using the main features of the jDSSAT. Also, the integration with the R environment that expands the possibilities of the DSSAT integration.

Code metadata

Current code version	2.0.0-rc.24
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_156
Legal Code License	List one of the approved licenses
Code versioning system used	git
Software code languages, tools, and services used	JavaScript
Compilation requirements, operating environments & dependencies	Git and Node
If available Link to developer documentation/manual	https://github.com/jabreuar/jdssat/
Support email for questions	jabreu.ar@gmail.com

Software metadata

Current software version	https://www.npmjs.com/package/jdssat
Permanent link to executables of this version	https://www.npmjs.com/package/jdssat
Legal Software License	List one of the approved licenses
Computing platforms/Operating Systems	distributed/web based
Installation requirements & dependencies	Git and Node https://www.npmjs.com/package/jdssat
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://www.npmjs.com/package/jdssat
Support email for questions	jabreu.ar@gmail.com

* Corresponding author.
E-mail addresses: jabreu.ar@gmail.com (J. de Abreu Resenes), pavan@upf.br (W. Pavan), holbig@upf.br (C.A. Hölbig), mauricio.fernandes@embrapa.br (J.M.C. Fernandes), vakhtang.shelia@ufl.edu (V. Shelia), cporter@ufl.edu (C. Porter), gerrit@ufl.edu (G. Hoogenboom).

https://doi.org/10.1016/j.softx.2019.100271

1. Introduction

Simulation of crop systems has significantly advanced over the past 40 years [1]. Crop modeling can facilitate researchers' ability to understand and interpret experimental results, and to diagnose yield gaps [2]. It can also be useful as a means to help the scientist define research priorities. Using a model to estimate the importance and the effect of certain parameters, a researcher can observe which factors should be more studied in future research, thus increasing the understanding of the system [3].

Decision support tools are designed to help users make more effective decisions by leading them through clear decision stages and presenting the likelihood of various outcomes resulting from different options [4]. These can be dynamic software tools, whose recommendations vary according to the user's inputs, and they may suggest an optimal decision path.

Decision Support Systems for Agrotechnology Transfer (DSSAT) were developed by the International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT) scientists' group [5]. DSSAT are the combination of crop simulation models (CSMs), database management programs and decision support systems (DSSs). CSMs are the computer programs that simulate crop growth and yield based on previously established calculations and input data of soil, weather and crop management practices [6]. The DSSAT-CSM can simulate yield on a range of crops and has been used by many scientists, decision-makers, and researchers all over the world for more than two decades. It is designed to facilitate the application of crop models in a systematic approach to agronomic research composed of more than 42 models. Prior to the development of the DSSAT, crop models were available, but these were used mostly in labs where they were created [7] and there were no sets of programs and a model suite that could be used as a tool for decision-making as well as predicting one or more crops within a cropping system [8].

The DSSAT Shell program provides a user-friendly working environment in which various stand-alone tools and applications are seamlessly integrated with the DSSAT crop models. Within the shell, the user can launch applications for creating and modifying data files, running the crop models, and analyzing the results [9]. However, the DSSAT Shell and other applications of the DSSAT system, which are installed separately, were built using technologies under end-of-life support such Visual Basic. Another point is that the DSSAT tools, including the Shell, are available for Windows OS only, which requires users from Linux and MacOS to deal with the command line to run model in DSSAT-CSM. This approach requires a very good understanding of DSSAT-CSM commands that might be painful for some users.

Other tools like *pyDSSAT* [10], python library to execute original Fortran program on Linux, were built to improve DSSAT's users experience over the terminal command line execution in Linux OS, including the ability of controlling input files, analyzing output files with Matplotlib tools, and providing a GUI toolkit for efficient interactive work integrate with DSSAT-CSM. There is also DASST, which is a R package for reading, processing and writing DSSAT files [11]. It uses tools available in R for statistical and graphical analyses. This package tends to simplify the post processing of DSSAT simulated values stored in .OUT, files offering methods that expose these data as belonging to a collection of data.frame objects that can be thought like tables. However, either pyDSSAT or DASST are not able to provide a full experience as DSSAT Shell does. They also do not provide a multiplatform approach and they were not built to provide alternatives to create a modern DSSAT Shell.

This study is motivated by the need to apply software engineering techniques to build a reusable approach for DSSAT integration. We have implemented common functionalities in a JavaScript module to allow any user interface easily read and processes DSSAT files. Tools like DSSAT Shell reads files only what is needed to provide info to the user and to be able to run the model. The implementation of the files processing relies on DSSAT Shell code itself, so it cannot be used anywhere else. Any code that does anything with a user interface should only involve user interface code [12]. There is also a need to implement software engineering best practices such cross-platform principle, to allow DSSAT users easily run and visualize simulations from Windows, Linux and MacOs.

The jDSSAT is a JavaScript Module created to be standard approach for DSSAT integration. The module is capable to process different DSSAT files such CDE, OUTPUT, EXPERIMENTS and TREATMENTS. It is designed as a module to either run on a client[1] or backend side[2] and facilitates developers from DSSAT community to create alternatives for DSSAT Shell, without having to worry about interoperability and extra tools installation. The jDSSAT module is cross-platform[3] and provides a series of JavaScript functions for DSSAT-CSM integration that allow any developer to build their own user interface to run DSSAT model in Linux, Windows and MacOS.

The remainder of this paper is structured as follows. Section 2 presents the key technical requirements that guided the jDSSAT development. Section 3 presents the jDSSAT design and architecture. Section 4 covers the main jDSSAT features and their implementation details. Section 5 describes usage patterns for jDSSAT functionalities. Finally, Section 6 presents our conclusions and directions for future research.

2. Requirements

There are four primary functional requirements that have guided jDSSAT implementation:

1. **Portability:** The jDSSAT should run the same code base in different operational systems, it is the crucial issue for development cost reduction. The operational systems supported are Linux, MacOS, and Windows. Also, it should integrate with DSSAT-CSM to run models on the operating system supported without the developer having to specify it.
2. **Reconfigurability:** To avoid having to rewrite pieces of code when a new DSSAT version is released, we would design a configurable system that can be applied in various scenarios. Below is a summarized list of the settings options for jDSSAT:
   - DSSAT work directory
   - DSSAT versions supported
   - Output files supported
3. **Web-based:** The jDSSAT module should be accessed over HTTP where processing is done over the internet on an external server. A REST interface should be provided to allow the integration with different programming languages.
4. **Reusability:** Reuse of jDSSAT artifacts in various formats. It should be used as "plug-and-play" in the client side using web technologies through NPM and in the server side through REST API.

The jDSSAT is designed to return data in data structures that are well-organized and ready to use, such as objects, lists and vectors. We describe the data structures for jDSSAT as follows:

[1] Requests pages from the Server, and displays them to the user. In most cases, the client is a web browser or an offline application.

[2] Responsible for serving pages.

[3] Is computer software that is implemented on multiple computing platforms.

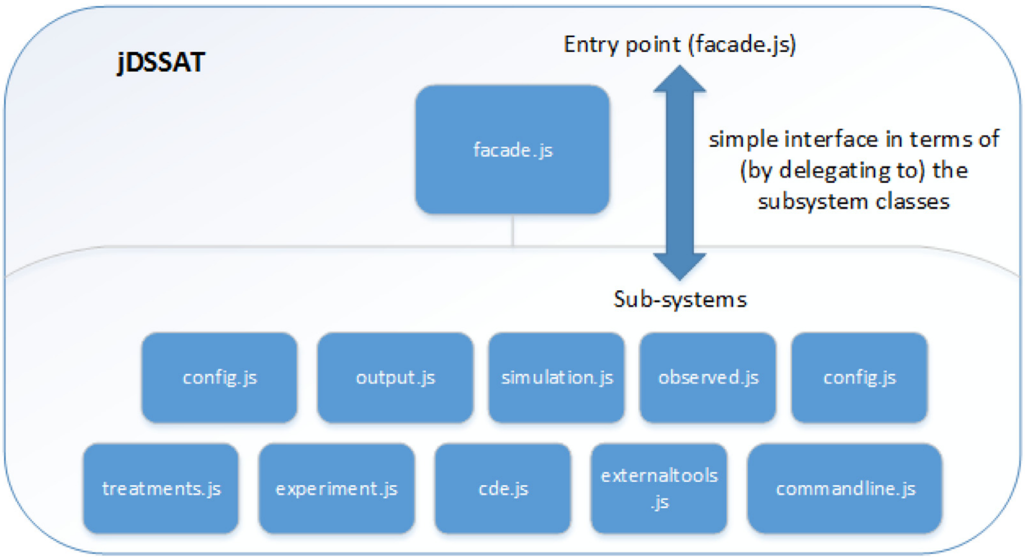


Fig. 1. Overview of jDSSAT using Facade design pattern. These entry point access the system on behalf of the facade client and hide the implementation details.

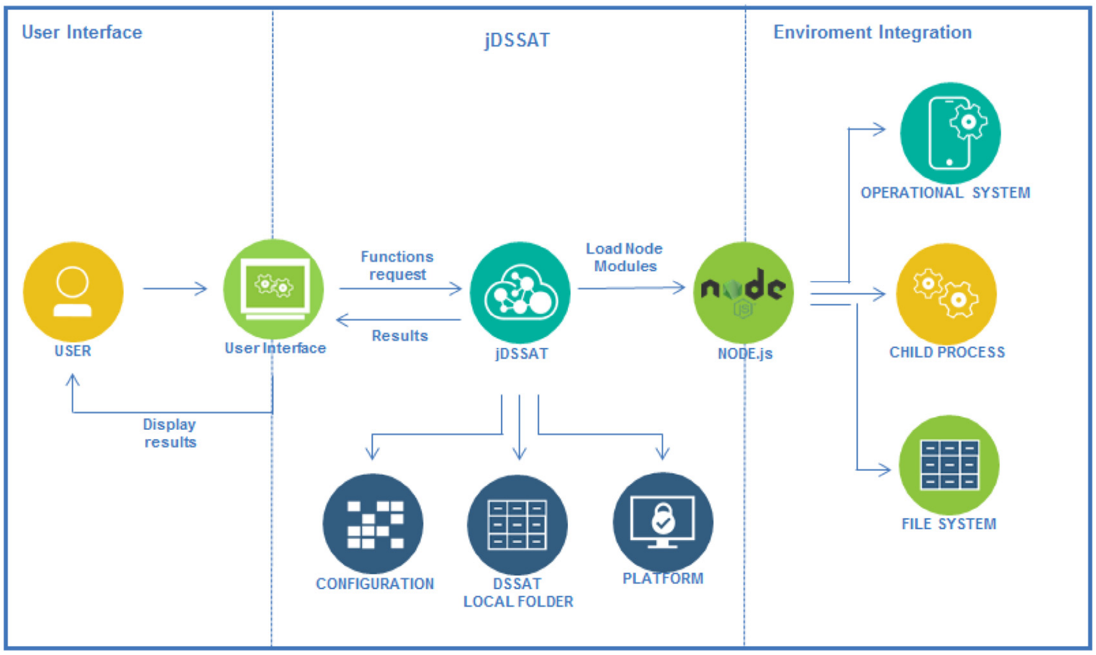


Fig. 2. The jDSSAT components architecture and operational system integration. The user interface sends a request to jDSSAT to retrieve and display the information such as crops, experiments and treatments. The node modules are used by jDSSAT to run operations such as process FILEX and run commands in DSSAT-CSM.

- Vectors to represent lists with single values such crops available in DSSAT.
- Complex objects containing property name and property value.
- Array of objects as lists of complex objects such output and experiment files.

3. The jDSSAT module design

We have implemented jDSSAT to make the architecture as much optimized and straightforward as possible. The module follows facade software-design pattern, Fig. 1, to provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use [13]. The facade pattern provides a more isolated functionalities implementation and reduce the risk of adding issues to the sub-systems that are not being changed. This pattern is particularly used when a system is very complex or difficult to understand because the system has a large number of interdependent logic.

The jDSSAT runs in a Node environment. In Fig. 2, we show the architecture components in a user interface context. The user interface loads jDSSAT as dependency and it provides info to the user through jDSSAT. Then, jDSSAT loads NodeJS [15] modules to access the File System, Child Process and Operational System. For instance, if the user interface needs to display the experiments available for a given crop, jDSSAT will process the EXPERIMENT file within crop folder and return a formatted object to the user interface.

Before jDSSAT start any integration with DSSAT is must be initialized. It first identifies the platform is running (Windows, Linux or MacOS), it does that by using the Operational System

Table 1
CSM modes of operation adapted from Overview of DSSAT4 Cropping System Model (CSM) [14].

Run mode	Description	Command line arguments
Batch	Experiments and treatments are listed in a batch file	B batchfilename
Sensitivity analysis	Screen user interface to interactively modify input parameters	E FileX Treatment#
Sequence analysis	Soil processes are continuous, crop sequence is listed in batch file	Q batchfilename
Spatial analysis	Simulates one or more crops over space.	S batchfilename
Seasonal analysis	Multiple years run with the same initial conditions	N batchfilename
Interactive	Screen user interface for interactive selection of experiment and treatment	(none)
Run all treatments	All treatments of specified experiment are run	A FileX
Debug	The input module is not called to read FILEX, soils files or cultivar file. Instead, the temporary input file (inpfile) is read.	D inpfile

Table 2
Auxiliary jDSSAT functions and its description.

Function	Description
jdssat.path()	returns the local DSSAT installation path
jdssat.version()	returns the DSSAT version in use by jDSSAT
jdssat.platform()	returns the platform (Win32, Darwin or Linux) where jDSSAT is running
jdssat.tree()	returns an array of folder model objects. The folders will be those who has experiments file inside
jdssat.experimentDescription(experiment, filePath)	returns a string with the experiment description
jdssat.createBashFile(crop, experiments[])	creates a batch file using a template
jdssat.outFiles(crop)	returns an array of output file names
jdssat.cde()	returns an array of objects containing cde, label and description fields
jdssat.openExternalTool(name)	opens a DSSAT Shell external tool
jdssat.openDssatFolder()	opens the current DSSAT folder
jdssat.openFileInEditor(crop,fileName)	opens a file in editor. The editor used depends on the platform
jdssat.getDataFiles(crop)	returns all files in a crop that ending with .*A or .*T extension
jdssat.folders()	returns all folders within DSSAT installation path
jdssat.filePreview(crop, fileName)	returns a file content in html tags
jdssat.batchCommand	creates a batch command
jdssat.runBatchFile(crop)	runs a batch command using child process node module

module that provides an object containing the platform name (win32, darwin or linux). Then, the configurations such DSSAT CSM executable file name, file system pattern and folder name for external tools. These configurations are stored in a configuration file to avoid code changes in case some path changes in futures DSSAT releases. Finally, jDSSAT finds what is the latest DSSAT version installed on the user's computer and make this version as default to use during the simulations. As a result of jDSSAT initialization, we can now make use of jDSSAT public functions (Table 2) for reading, processing and writing DSSAT-CSM files. When a function that requires a folder or file content read, jDSSAT makes a file system call using File System (fs) module from Node. The fs module provides an API for interacting with the file system in a manner closely modeled around standard POSIX function [16].

The DSSAT-CSM provides a mechanism to run a model simulation through command line, which is used by jDSSAT. The command line integration in jDSSAT is made by child process module, that enables access to the Operating System functionalities. So, once jDSSAT simulation function is called, a command is executed on the operational system.

Other components and implementation details are discussed in later sections.

4. Implementation and testing

The jDSSAT implementation is divided into two main components, DSSAT files processing and DSSAT-CSM integration. This section discusses the implementation of the current release of jDSSAT and also how we tested its functions.

4.1. DSSAT files processing

The jDSSAT is able to process FILEX for a specific experiment. This files contains data on treatments, field conditions, crop management and simulation controls [14] and they are found within crop folders with *.*X* extension. Also, jDSSAT can process *OUTPUT* file generated by the crop model that contains results of a simulation. The General DSSAT Profile, named as *DSSATPRO*, is processed to provide information such extension, command line and path for each crop.

4.1.1. Crop folders

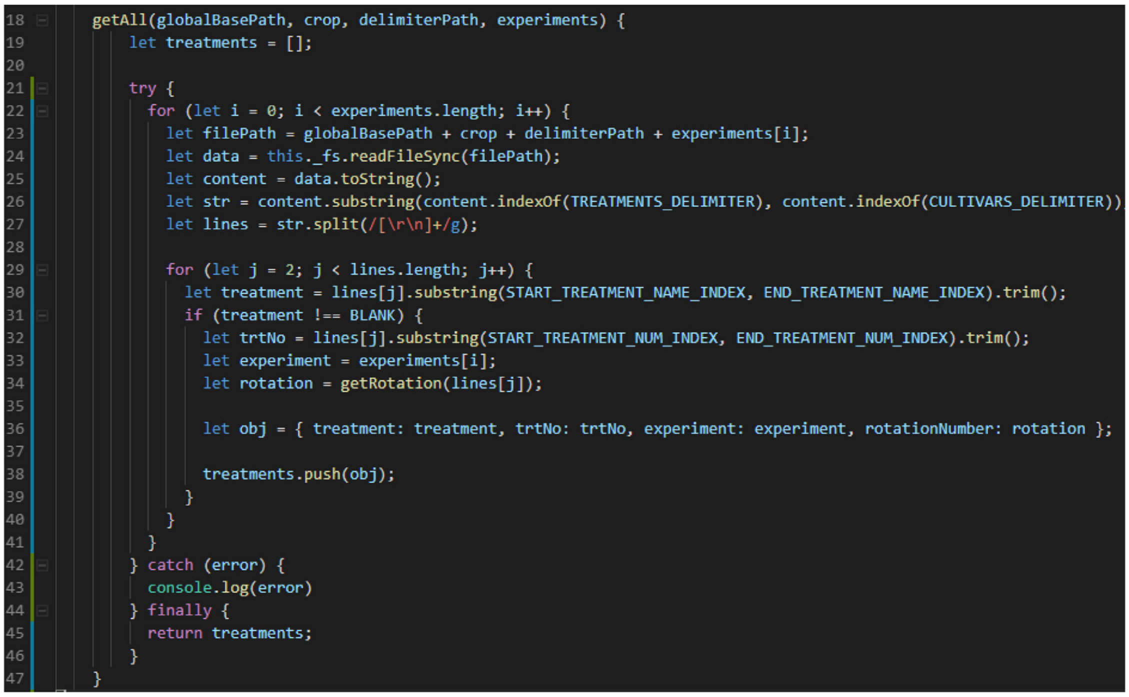
The jDSSAT uses DSSAT installation directory content to find *DSSATPRO* file, which designates the locations of all programs and data files used in DSSAT [9]. The DSSATPRO extension file depends on the DSSAT version installed, on Windows, Linux and MacOS, the latest DSSAT version released is **v47**. In other to provide a list of available crops in DSSAT, the jSSAT must read and parse the `DSSATPRO.v47` file, on Windows, to build an array of objects containing crop name, location folder and crop extension. The folders that do not have FILEX in its content will not be returned.

4.1.2. Reading experiments

The read experiment function will look by all FILEX within folder's content. The function will look by files ending with *.*X extension in the crop folder. For each FILEX file the algorithm 1 creates an array of experiments with an object containing the description, modified date, name and number as object fields.

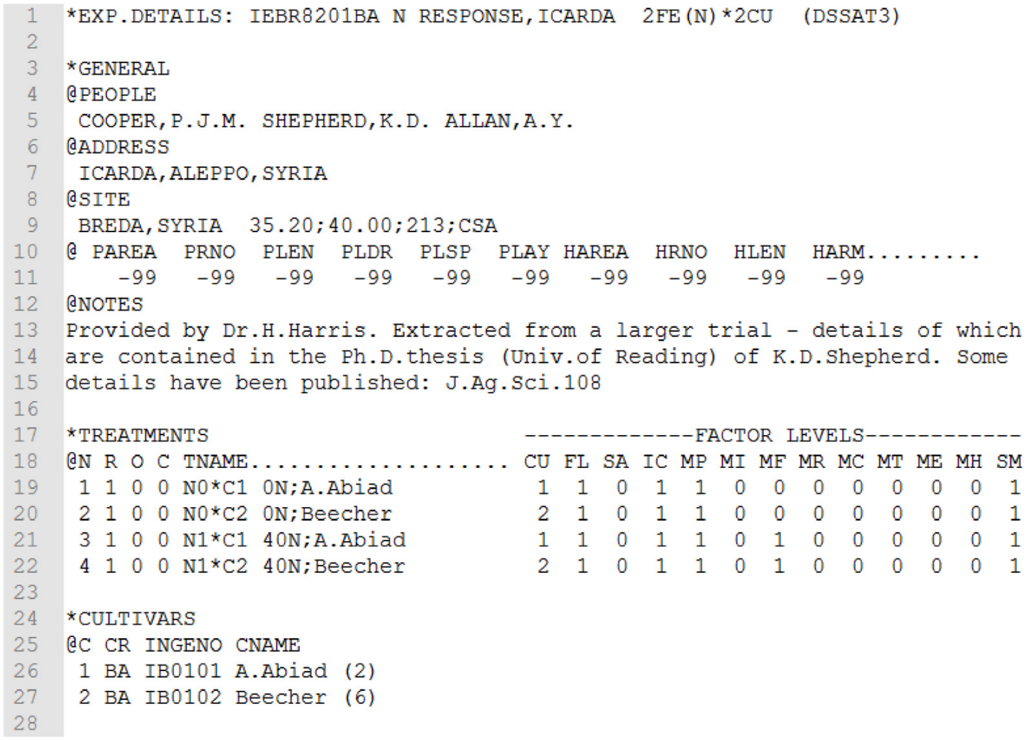
4.1.3. Reading treatments

The treatments function, Fig. 3, will receive an array of FILEX names as input. Each FILEX contains a tratments section, as shown in the Fig. 4 lines 17 to 22, that will be used to retrieve tratments information. The jDSSAT first loads the FILEX content, then it will parse the content by doing a substring of the index of **TREATMENTS* until the index of **CULTIVARS*. After that, a loop is made through lines of the substring to get the treatment, treatment number and experiment to format an object to send as part of the function response.



```
getAll(globalBasePath, crop, delimiterPath, experiments) {
  let treatments = [];

  try {
    for (let i = 0; i < experiments.length; i++) {
      let filePath = globalBasePath + crop + delimiterPath + experiments[i];
      let data = this._fs.readFileSync(filePath);
      let content = data.toString();
      let str = content.substring(content.indexOf(TREATMENTS_DELIMITER), content.indexOf(CULTIVARS_DELIMITER))
      let lines = str.split(/[\r\n]+/g);

      for (let j = 2; j < lines.length; j++) {
        let treatment = lines[j].substring(START_TREATMENT_NAME_INDEX, END_TREATMENT_NAME_INDEX).trim();
        if (treatment !== BLANK) {
          let trtNo = lines[j].substring(START_TREATMENT_NUM_INDEX, END_TREATMENT_NUM_INDEX).trim();
          let experiment = experiments[i];
          let rotation = getRotation(lines[j]);

          let obj = { treatment: treatment, trtNo: trtNo, experiment: experiment, rotationNumber: rotation };

          treatments.push(obj);
        }
      }
    }
  } catch (error) {
    console.log(error)
  } finally {
    return treatments;
  }
}
```

Fig. 3. Treatment function implementation. For each FILEX received as input the function load its content and finds the tratments section in the file and load a treatments object array as result.



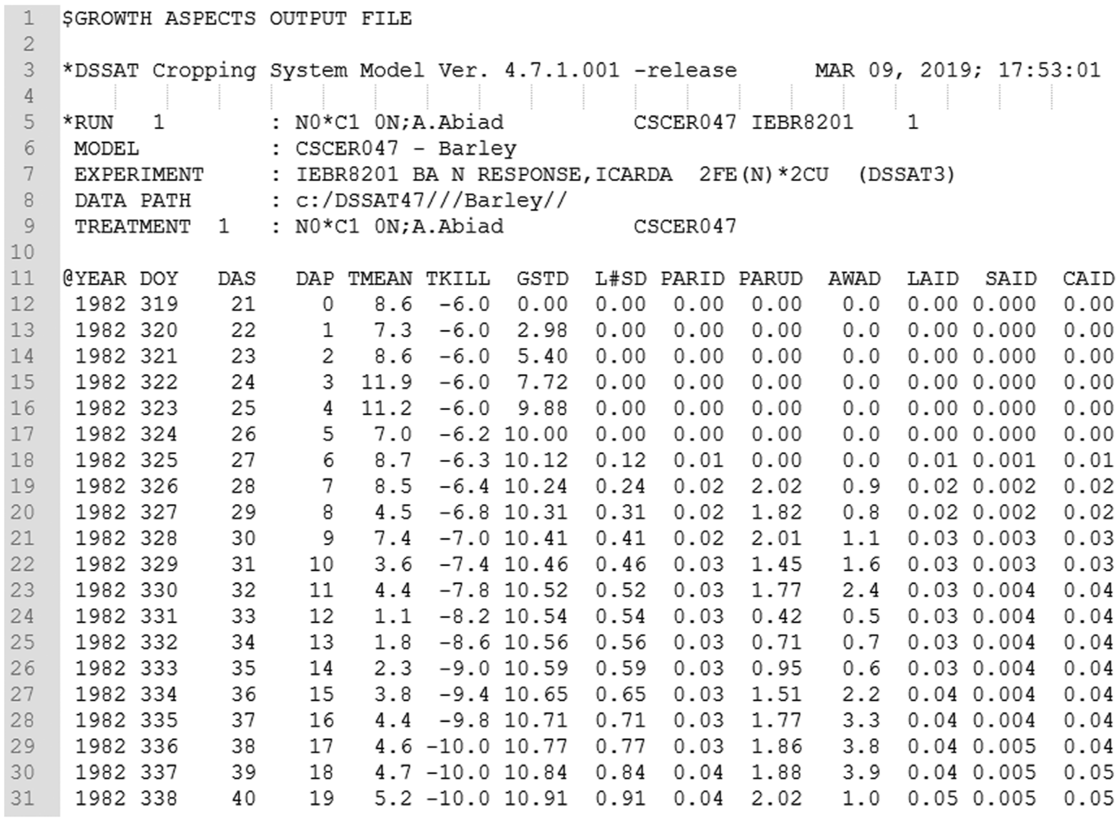
```
 1  *EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)
 2
 3  *GENERAL
 4  @PEOPLE
 5   COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
 6  @ADDRESS
 7   ICARDA,ALEPPO,SYRIA
 8  @SITE
 9   BREDA,SYRIA  35.20;40.00;213;CSA
10  @ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
11      -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
12  @NOTES
13  Provided by Dr.H.Harris. Extracted from a larger trial - details of which
14  are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
15  details have been published: J.Ag.Sci.108
16
17  *TREATMENTS                        -------------FACTOR LEVELS------------
18  @N R O C TNAME.................... CU FL SA IC MP MI MF MR MC MT ME MH SM
19   1 1 0 0 N0*C1 0N;A.Abiad           1  1  0  1  1  0  0  0  0  0  0  0  1
20   2 1 0 0 N0*C2 0N;Beecher           2  1  0  1  1  0  0  0  0  0  0  0  1
21   3 1 0 0 N1*C1 40N;A.Abiad          1  1  0  1  1  0  1  0  0  0  0  0  1
22   4 1 0 0 N1*C2 40N;Beecher          2  1  0  1  1  0  1  0  0  0  0  0  1
23
24  *CULTIVARS
25  @C CR INGENO CNAME
26   1 BA IB0101 A.Abiad (2)
27   2 BA IB0102 Beecher (6)
28
```

Fig. 4. Partial content of FILEX (IEBR8201.BAX) from Barley crop. The jDSSAT reads the tratments, lines 17 to 22, from this files and returns a formatted response with treatments properties as number and (N) and name (TNAME).

4.1.4. Reading .OUT files

Crop simulation models can provide very detailed outputs of the simulated crop. However, analyzing the outputs is challenging. The jDSSAT provides an easy way to read and process these files. The result of a simulation are stored in `.OUT` extension files, these files contains a fixed width format. As seen in Fig. 5, the line 11 are headers that represents the variable names, each row after the header are the variable values, if a simulation ran with more than one experiment (line 7) or treatment (line 9), this file will have more headers and variable values on its content.

The read output file function receives the crop name and output file name as input. The crop name variable value will be used to identify where the output file is located in the user's

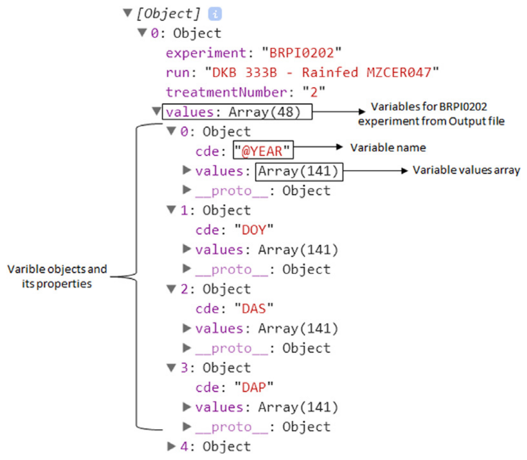


```
 1 $GROWTH ASPECTS OUTPUT FILE
 2
 3 *DSSAT Cropping System Model Ver. 4.7.1.001 -release      MAR 09, 2019; 17:53:01
 4
 5 *RUN   1          : N0*C1 0N;A.Abiad          CSCER047 IEBR8201    1
 6  MODEL            : CSCER047 - Barley
 7  EXPERIMENT       : IEBR8201 BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)
 8  DATA PATH        : c:/DSSAT47///Barley//
 9  TREATMENT  1     : N0*C1 0N;A.Abiad          CSCER047
10
11 @YEAR DOY   DAS   DAP TMEAN TKILL  GSTD  L#SD PARID PARUD  AWAD  LAID  SAID  CAID
12  1982 319    21     0   8.6  -6.0  0.00  0.00  0.00  0.00   0.0  0.00 0.000  0.00
13  1982 320    22     1   7.3  -6.0  2.98  0.00  0.00  0.00   0.0  0.00 0.000  0.00
14  1982 321    23     2   8.6  -6.0  5.40  0.00  0.00  0.00   0.0  0.00 0.000  0.00
15  1982 322    24     3  11.9  -6.0  7.72  0.00  0.00  0.00   0.0  0.00 0.000  0.00
16  1982 323    25     4  11.2  -6.0  9.88  0.00  0.00  0.00   0.0  0.00 0.000  0.00
17  1982 324    26     5   7.0  -6.2 10.00  0.00  0.00  0.00   0.0  0.00 0.000  0.00
18  1982 325    27     6   8.7  -6.3 10.12  0.12  0.01  0.00   0.0  0.01 0.001  0.01
19  1982 326    28     7   8.5  -6.4 10.24  0.24  0.02  2.02   0.9  0.02 0.002  0.02
20  1982 327    29     8   4.5  -6.8 10.31  0.31  0.02  1.82   0.8  0.02 0.002  0.02
21  1982 328    30     9   7.4  -7.0 10.41  0.41  0.02  2.01   1.1  0.03 0.003  0.03
22  1982 329    31    10   3.6  -7.4 10.46  0.46  0.03  1.45   1.6  0.03 0.003  0.03
23  1982 330    32    11   4.4  -7.8 10.52  0.52  0.03  1.77   2.4  0.03 0.004  0.04
24  1982 331    33    12   1.1  -8.2 10.54  0.54  0.03  0.42   0.5  0.03 0.004  0.04
25  1982 332    34    13   1.8  -8.6 10.56  0.56  0.03  0.71   0.7  0.03 0.004  0.04
26  1982 333    35    14   2.3  -9.0 10.59  0.59  0.03  0.95   0.6  0.03 0.004  0.04
27  1982 334    36    15   3.8  -9.4 10.65  0.65  0.03  1.51   2.2  0.04 0.004  0.04
28  1982 335    37    16   4.4  -9.8 10.71  0.71  0.03  1.77   3.3  0.04 0.004  0.04
29  1982 336    38    17   4.6 -10.0 10.77  0.77  0.03  1.86   3.8  0.04 0.005  0.04
30  1982 337    39    18   4.7 -10.0 10.84  0.84  0.04  1.88   3.9  0.04 0.005  0.05
31  1982 338    40    19   5.2 -10.0 10.91  0.91  0.04  2.02   1.0  0.05 0.005  0.05
```

Fig. 5. Partial content of PlatGro.OUT file, the read output file function from jDSSAT will process the lines 11 to 31 and as result will return an array of objects with the variable name and its values, so the number of columns in the output file will be the number of objects within the result array.

Algorithm 1: Get experiments for a given crop algorithm

```
  Input  : crop
  Output: experiments array
1 while crop content is not null do
2     instructions;
3     if IS FILEX then
4         add object in the experiments array;
5     else
6         continue;
7     end
8 end
```



```
▼ [Object]
  ▼ 0: Object
      experiment: "BRPI0202"
      run: "DKB 333B - Rainfed MZCER047"
      treatmentNumber: "2"
    ▼ values: Array(48)
      ▼ 0: Object
          cde: "@YEAR"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 1: Object
          cde: "DOY"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 2: Object
          cde: "DAS"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 3: Object
          cde: "DAP"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▶ 4: Object
```

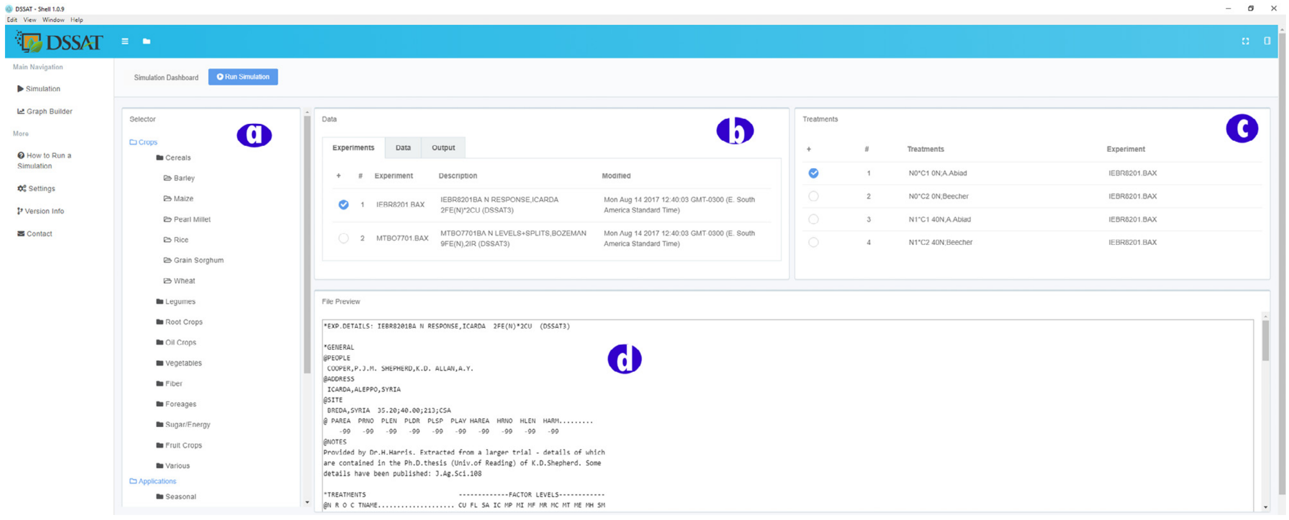
Variables for BRPI0202 experiment from Output file
Variable name
Variable values array
Varible objects and its properties

Fig. 6. Read output file function object result. It shows the total variables found and partial variables and its values content.

machine. Once the output file location is found, jDSSAT reads the outfile file content as follows:

1. Initializes a header object with index, length, variable name.
2. Initialize an array of objects to store experiment, treatment number, treatment and its values.
3. Loop through the lines.
4. Finds what is the run number, experiment and treatment. Each of these information has one prefix. This step is performed by looking at RUN, MODEL and EXPERIMENT respectively in the begging of the line. If one of this identifier is found, jDSSAT reads the content line that cames after.
5. Reads simulation values. The @ symbol followed by YEAR variable is the first character of the data-headers line in DSSAT output files. When jDSSAT finds this symbol in the beginning of the line, it initializes an auxiliary array to store the header. Each value between spaces will be one position on the header array.
6. Gets values for each header variable. The algorithm should consider that next lines in the loop will contain the data, similar to *EasyGrapher*' approach [17]. This process should execute until the algorithm does not find another run.
7. Repeat all the steps if there are more than one run in the output file.

The result of reading a DSSAT output file will be an array, we seen in Fig. 6, containing an object for each run. The jDSSAT also reads `Summary.OUT` and `Overview.OUT` to provide more information about simulation results.



+	#	Experiment	Description	Modified
✓	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	Mon Aug 14 2017 12:40:03 GMT-0300 (E. South America Standard Time)
	2	MTBO7701.BAX	MTBO7701BA N LEVELS+SPLITS,BOZEMAN 9FE(N),2IR (DSSAT3)	Mon Aug 14 2017 12:40:03 GMT-0300 (E. South America Standard Time)

+	#	Treatments	Experiment
✓	1	N0*C1 0N;A.Abiad	IEBR8201.BAX
	2	N0*C2 0N;Beecher	IEBR8201.BAX
	3	N1*C1 40N;A.Abiad	IEBR8201.BAX
	4	N1*C2 40N;Beecher	IEBR8201.BAX

```
*EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)

*GENERAL
@PEOPLE
 COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
@ADDRESS
 ICARDA,ALEPPO,SYRIA
@SITE
 BREDA,SYRIA  35.20;40.00;213;CSA
@ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
    -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
@NOTES
Provided by Dr.H.Harris. Extracted from a larger trial - details of which
are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
details have been published: J.Ag.Sci.108

*TREATMENTS                        -------------FACTOR LEVELS------------
@N R O C TNAME.................... CU FL SA IC MP MI MF MR MC MT ME MH SM
```

Fig. 7. An user interface to run DSSAT model built using jDSSAT functions responses. The UI has four main components: (a) selector, for the user select the crop; (b) data, displays the experiments available for the crop select; (c) treatments available for the experiment selected; and (d) file preview, display FILEX preview. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

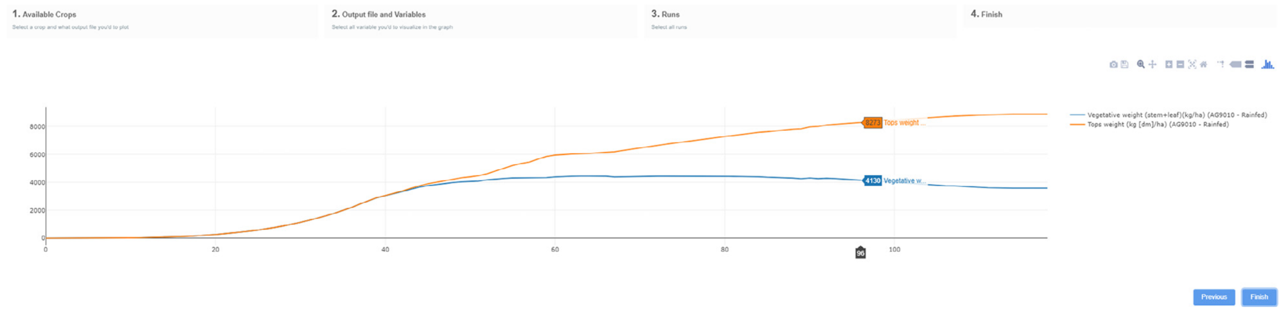


Fig. 8. Results of a Maize simulation experiment. The read output function from jDSSAT was used to retrieve the vegetative weight (orange line) and tops weight (blue line) variables value. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.2. DSSAT-CSM integration

The DSSAT-CSM incorporates models of all crops within a single set of code. The run mode is specified by the command line arguments when the model is called. If the model is run using the DSSAT shell, these command lines are transparent to the user [14]. There are set of commands available for CSM, Table 1, the run modes batch and sensitivity are available in the current jDSSAT version. The child process Node module within jDSSAT enables access Operating System functionalities by running any system command.

4.2.1. Running a simulation

To run a CSM model first we need to select the experiments and treatments to be simulated. A group of experiments and treatments can be run in a single simulation or in batch mode. A single experiment/treatment can be run either as a batch simulation or in sensitivity mode [18]. The jDSSAT approach is a batch simulation. The simulation is divided into two steps.

1. **Creating a batch file:** The batch file is a set of configurations to run a model such experiments and treatments list. There is a pre-built template of this file within jDSSAT. The function uses this template to fill up the configurations to run a model in batch mode.
2. **Running a command:** After the batch file is created, jDSSAT uses child process node module [19] to execute the command created in the command-line interpreter.

4.3. Auxiliary functions

There are also auxiliary functions in other to help users to know additional information, such as the DSSAT version and path being used. The complete list of auxiliary functions can be seen at Table 2.

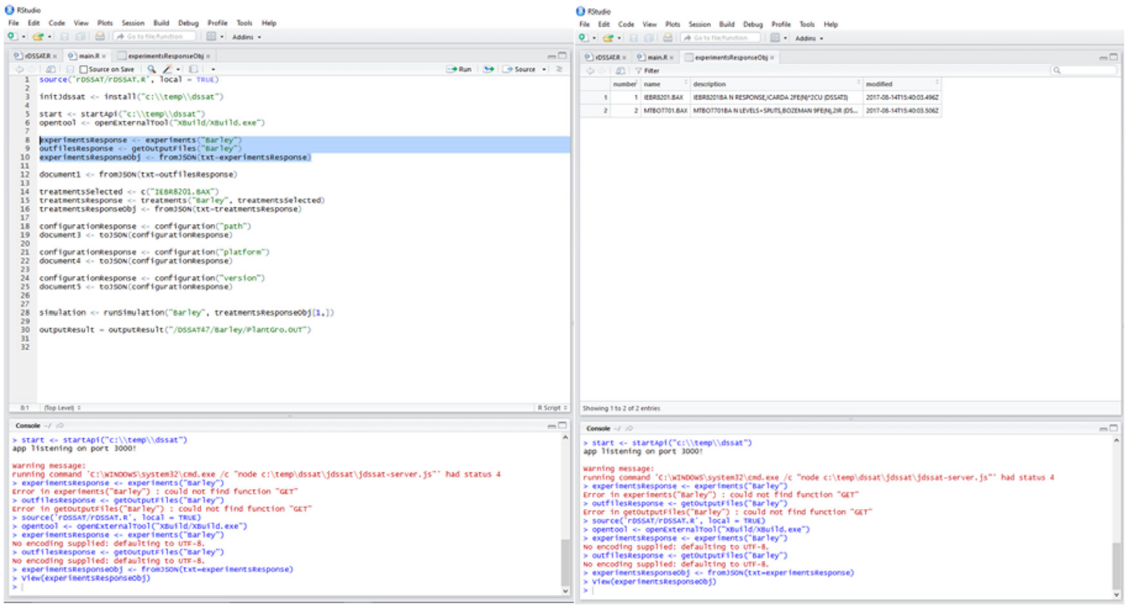
4.4. Module testing

We have used jDSSAT functions to create an user interface, shown in Fig. 7, to run DSSAT models. This user interface provides functionalities to the user to choose a crop, experiment, and treatments. Also, it displays a preview of the file that is being processed. The option to run a simulation is available in the blue button on the screen. The user interface also provides a Graph builder, Figure Fig. 8, for the user to visualize the simulation results, it provides options for select the crop, what output file and what variables to plot in a graph.

A simple R package was created to test the jDSSAT over HTTP. This package download jDSSAT in the local machine and start a API to send request from R to jDSSAT. In Fig. 9 we show a experiment function response in R.

5. Practical usage

This section discusses the practical usage of jDSSAT and its functions. There is a skeleton project with the basic configuration that needs to be used to run the jDSSAT functions according next sessions' instructions.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Addins
rDSSAT.R
main.R
experimentsResponseObj
Source on Save
Run
Source
Filter
Showing 1 to 2 of 2 entries
Console

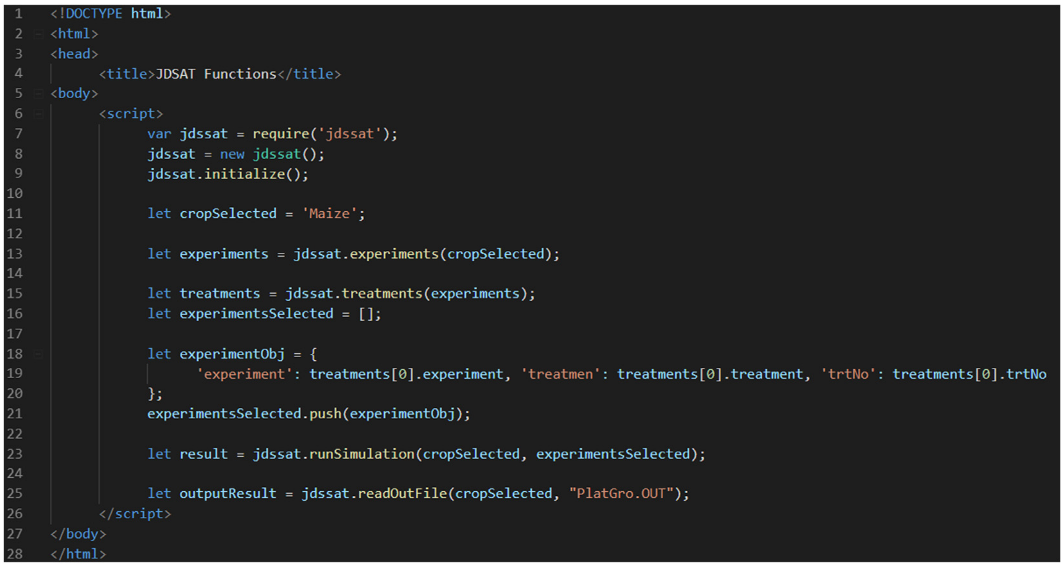
```r
source('rDSSAT/rDSSAT.R', local = TRUE)

initJdssat <- install("c:\\temp\\dssat")

start <- startApi("c:\\temp\\dssat")
opentool <- openExternalTool("XBuild/XBuild.exe")

experimentsResponse <- experiments("Barley")
outfilesResponse <- getOutputFiles("Barley")
experimentsResponseObj <- fromJSON(txt=experimentsResponse)

document1 <- fromJSON(txt=outfilesResponse)

treatmentsSelected <- c("IEBR8201.BAX")
treatmentsResponse <- treatments("Barley", treatmentsSelected)
treatmentsResponseObj <- fromJSON(txt=treatmentsResponse)

configurationResponse <- configuration("path")
document3 <- toJSON(configurationResponse)

configurationResponse <- configuration("platform")
document4 <- toJSON(configurationResponse)

configurationResponse <- configuration("version")
document5 <- toJSON(configurationResponse)


simulation <- runSimulation("Barley", treatmentsResponseObj[1,])

outputResult = outputResult("/DSSAT47/Barley/PlantGro.OUT")
```

```
> start <- startApi("c:\\temp\\dssat")
app listening on port 3000!

Warning message:
running command 'C:\WINDOWS\system32\cmd.exe /c "node c:\temp\dssat\jdssat\jdssat-server.js"' had status 4
> experimentsResponse <- experiments("Barley")
Error in experiments("Barley") : could not find function "GET"
> outfilesResponse <- getOutputFiles("Barley")
Error in getOutputFiles("Barley") : could not find function "GET"
> source('rDSSAT/rDSSAT.R', local = TRUE)
> opentool <- openExternalTool("XBuild/XBuild.exe")
> experimentsResponse <- experiments("Barley")
No encoding supplied: defaulting to UTF-8.
> outfilesResponse <- getOutputFiles("Barley")
No encoding supplied: defaulting to UTF-8.
> experimentsResponseObj <- fromJSON(txt=experimentsResponse)
> View(experimentsResponseObj)
> |
```

	number	name	description	modified
1	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	2017-08-14T15:40:03.496Z
2	2	MTBO7701.BAX	MTBO7701BA N LEVELS+SPLITS,BOZEMAN 9FE(N),2IR (DS...	2017-08-14T15:40:03.506Z

```
> start <- startApi("c:\\temp\\dssat")
app listening on port 3000!

Warning message:
running command 'C:\WINDOWS\system32\cmd.exe /c "node c:\temp\dssat\jdssat\jdssat-server.js"' had status 4
> experimentsResponse <- experiments("Barley")
Error in experiments("Barley") : could not find function "GET"
> outfilesResponse <- getOutputFiles("Barley")
Error in getOutputFiles("Barley") : could not find function "GET"
> source('rDSSAT/rDSSAT.R', local = TRUE)
> opentool <- openExternalTool("XBuild/XBuild.exe")
> experimentsResponse <- experiments("Barley")
No encoding supplied: defaulting to UTF-8.
> outfilesResponse <- getOutputFiles("Barley")
No encoding supplied: defaulting to UTF-8.
> experimentsResponseObj <- fromJSON(txt=experimentsResponse)
> View(experimentsResponseObj)
> |
```

Fig. 9. Experiment function running in R through jDSSAT API. The right side shows a code sample to retrieve the experiments from Barley crop and the left side shows a frame if the result of the experiments function.



```html
<!DOCTYPE html>
<html>
<head>
    <title>JDSAT Functions</title>
<body>
    <script>
        var jdssat = require('jdssat');
        jdssat = new jdssat();
        jdssat.initialize();

        let cropSelected = 'Maize';

        let experiments = jdssat.experiments(cropSelected);

        let treatments = jdssat.treatments(experiments);
        let experimentsSelected = [];

        let experimentObj = {
            'experiment': treatments[0].experiment, 'treatmen': treatments[0].treatment, 'trtNo': treatments[0].trtNo
        };
        experimentsSelected.push(experimentObj);

        let result = jdssat.runSimulation(cropSelected, experimentsSelected);

        let outputResult = jdssat.readOutFile(cropSelected, "PlatGro.OUT");
    </script>
</body>
</html>
```

Fig. 10. Sample of jDSSAT functions to initialize, read experiments and treatments, run simulation and retrieve results.

5.1. Installation

The jDSSAT is available through Node package manager (npm), the package manager for JavaScript [20]. The download can be done by running *npm install jdssat* command. Once the jDSSAT download is completed, a folder is automatically created under node modules.

```
project
├── app
└── node_module
    └── jdssat
        └── source
```

5.2. Functions

The code seen in Fig. 10 shows how to initialize and use experiments, treatments, simulation and read output files function. The initialize function requires jdssat module, the default initialization function identifies the platform windows (win32), linux and MacOS(darwin) then it finds the latest DSSAT version installed on user's machine by looking at "C:/" on Win32 or "/" on Darwin and Linux platform. Also, it loads a series of configuration, such base path for the operating system, the version of DSSATPRO [7] and other extra data for managing the fixed width format properties. The experiments function input is a crop, it reads all FILEX from the crop selected. Also, the experiments are used as input for the treatments function. To run a DSSAT model simulation, the jDSSAT run simulation function expects the crop and an array of objects with the experiments selected. As seen in Fig. 10 (lines 18, 19 and 20), the objects within the array are composed by experiment, treatment and treatment number. Finally, the read output function input is a crop selected and output file name.

6. Conclusions and future work

In this paper, we have presented jDSSAT, a flexible and powerful JavaScript module that abstracts the complexity of reading and processing DSSAT files. The module integrates with DSSAT-CSM in different operational system for running models. The implementation of jDSSAT also highlights interoperability benefits, which have been an issue in the DSSAT tools. Also, the jDSSAT is designed to be a standard module for DSSAT integration which reduces the effort to build integrations with DSSAT in other programming languages such as R.

The jDSSAT is an ongoing project. We intend to reuse its code to have a version to run DSSAT models on the web. There is also a need of develop an end-to-end user interface as an alternative for the current DSSAT Shell.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.softx.2019.100271.

Acknowledgments

Many thanks for my Advisor Prof. Dr. Willingthon Pavan and my Co-Advisor, Prof. Dr. Carlos Amaral Holbig for all the support and feedbacks. Also, thanks to Prof. Dr. José Maurício Cunha Fernandes for the feedbacks and partnership during this work. Special thanks to Dr. Gerrit Hoogenboom for all the support and also to be open and accept this work on the DSSAT community.

References

[1] Boote KJ, Jones JW, Hoogenboom G, White JW. The role of crop systems simulation in agriculture and environment. Int J Agric Environ Inform Syst 2010;1(1):41–54. http://dx.doi.org/10.4018/jaeis.2010101303.

[2] Liu H, Liu H, Lei Q, Zhai L, Wang H, Zhang J, Zhu Y, Liu S, Li S, Zhang J, Liu X. Using the DSSAT model to simulate wheat yield and soil organic carbon under a wheat-maize cropping system in the North China Plain. J Integrative Agric 2017;16(10):2300–7. http://dx.doi.org/10.1016/S2095-3119(17)61678-2.

[3] Dourado-Neto D, Teruel D, Reichardt K, Nielsen D, Frizzone JA, Bacchi OOS. Principles of crop modeling and simulation: I. Uses of mathematical models in agricultural science. Sci Agric 1998;55(SPE):46–50.

[4] Rose DC, Sutherland WJ, Parker C, Lobley M, Winter M, Morris C, Twining S, Ffoulkes C, Amano T, Dicks LV. Decision support tools for agriculture: Towards effective design and delivery. Agric Syst 2016;149:165–74.

[5] Sarkar R, et al. Use of DSSAT to model cropping systems. CAB Rev 2009;4(025):1–12. http://dx.doi.org/10.1079/PAVSNNR20094025.

[6] Sarkar R, Kar S. Evaluation of management strategies for sustainable rice–wheat cropping system, using DSSAT seasonal analysis. J Agric Sci 2006;144(5):421–34.

[7] Jones J, Hoogenboom G, Porter C, Boote K, Batchelor W, Hunt L, Wilkens P, Singh U, Gijsman A, Ritchie J. The DSSAT cropping system model. Eur J Agron 2003;18(3):235–65. http://dx.doi.org/10.1016/S1161-0301(02)00107-7.

[8] Sarkar R. Decision support systems for agrotechnology transfer. In: Lichtfouse E, editor. Organic fertilisation, soil quality and human health. Dordrecht: Springer Netherlands; 2012, p. 263–99. http://dx.doi.org/10.1007/978-94-007-4113-3_10.

[9] Wilkens PW, Hoogenboom G, Porter CH, Jones JW, Uryasev O. DSSAT v4 data management and analysis tools: Vol. 2. International Consortium for Agricultural Systems Applications; 2004, p. 177.

[10] He X, Peng L, Sun H. pyDSSAT Documentation release 1.0. 2015, http://xiaoganghe.github.io/pyDSSAT/doc/index.html. [Accessed 16 September 2018].

[11] Lozza H. Dasst: tools for reading, processing and writing DSSAT files, R package version 0.3.3, Buenos Aires, Argentina. 2017, https://github.com/hlozza/Dasst. [Accessed 10 July 2018].

[12] Fowler M. Separating user interface code. IEEE Softw 2001;18(2):96–7.

[13] Gamma E. Design patterns: elements of reusable object-oriented software. Pearson Education India; 1995.

[14] Porter CH, Wilkens PW. DSSAT v4.5 overview: Vol. 1.

[15] Nodejs Foundation. Node.js. 2018, https://nodejs.org/en/. [Accessed 24 September 2018].

[16] Herron D. Node web development. Packt Publishing Ltd; 2013.

[17] Yang J, Huffman ET. EasyGrapher: software for graphical and statistical validation of DSSAT outputs. Comput Electron Agric 2004;45(1):125–32. http://dx.doi.org/10.1016/j.compag.2004.06.006.

[18] Jones JW, Tsuji GY, Hoogenboom G, Hunt LA, Thornton PK, Wilkens PW, Imamura DT, Bowen WT, Singh U. Decision support system for agrotechnology transfer: DSSAT v3. In: Tsuji GY, Hoogenboom G, Thornton PK, editors. Understanding options for agricultural production. Dordrecht: Springer Netherlands; 1998, p. 157–77. http://dx.doi.org/10.1007/978-94-017-3624-4_8.

[19] Wilson J. Node.js 8 the right way: practical, server-side javascript that scales. Pragmatic Bookshelf; 2018.

[20] npm Inc. Node package manager. 2018, https://www.npmjs.com/. [Accessed 10 July 2018].