A JavaScript Module for DSSAT-CSM integration

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.

# 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 client1 or backend side2 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-platform3 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.

# 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:

• 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.

# 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 issuesdato 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 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.

# 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.

## DSSAT files processing

### 4.1.1 Design of Central Server

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.

### 4.1.2 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.

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

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.

## 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.

# 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 DSSATCSM 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.