Componentized-WEAP RESTful Framework Installation and User Guide*

Version 1.0

Mostafa D. Fard

Hessam S. Sarjoughian**

Arizona Center for Integrative Modeling and Simulation
School of Computing, Informatics, and Decision System Engineering
Arizona State University, Tempe, Arizona, USA
https://acims.asu.edu

July 2020

* Funding: United States National Science Foundation Grant #CNS-1639227, "INFEWS/T2: Flexible Model Compositions and Visual Representations for Planning and Policy Decisions at the Sub-regional level of food-energy-water nexus".

** Point-of-Contact

Table of Contents

List of figures

List of tables

1. Componentized-WEAP Software Application

The Componentized-WEAP (C-WEAP) is a RESTful framework application [1] written in **NodeJS** for the Water Evaluation and Planning (WEAP) system [2] [3].

2. WEAP Software System

This is a propriety system is a software supported for the Windows OS. Information on lincensing this software is available at WEAP Licensing.

3. Executable Componentized-WEAP Software Application Installation

The executable version of the Componentized-WEAP (C-WEAP) framework is a standalone application to be run on the Windows OS. You just need to download the executables version of the C-WEAP framework from https://acims.asu.edu/software/c-weap/, then unzip the downloaded file in a directory. It contains the "workspace" folder (which uses to manage the required flat files by the C-WEAP framework and WEAP system), the "config.json" file (to set the host and port number of the web-service), the "node-activex.node" (the required third-party packages), and the C-WEAP.exe file. All required libraries and frameworks (NodeJS, Express, etc.) are embedded in this executable file. The unzipped directory should not be installed shared disk drives such as Dropbox.

To run the C-WEAP double-click on the C-WEAP.exe file. Upon successful execution of C-WEAP application, the message "Componentized-WEAP is listening at http://{hostname}:{port}" appears in the first line of a Command Prompt console such as the one shown in Figure 1.

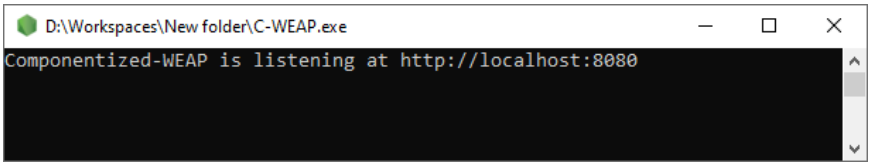


Figure 1. Windows console displaying successful execution of the C-WEAP.

The C-WEAP can invoke a defined set of WEAP system APIs following the procedure described and exemplified in Section 5.

4. Componentized-WEAP Source Code Installation

To execute the C-WEAP software application from source code, the following software frameworks and tools need to be installed.

4.1. NodeJS

Download the NodeJS framework for the Windows 64-bit OS (MSI or ZIP) from https://nodejs.org/en/download/ (see Figure 2). At the time of preparing this user guide, the latest version of the NodeJS framework is *12.18.1*. It also includes *npm 6.14.5* (Node Package Management). After downloading, use the default choices to install NodeJS.

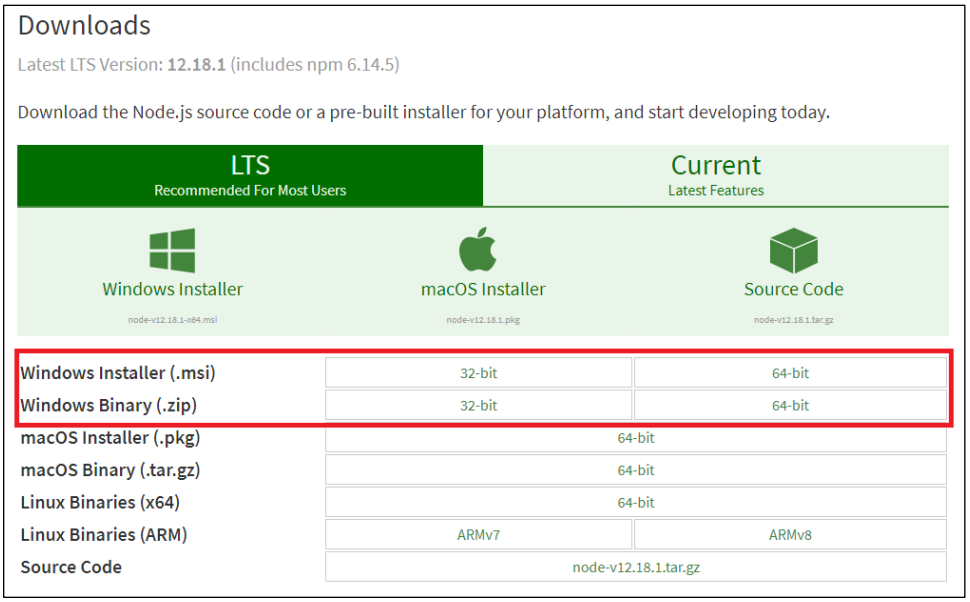


Figure 2. NodeJS download page (https://nodejs.org/en/download/).

4.2. Python

Installing Python requires multiple installations in the order provided below.

Step 1: Choose and click on **Python version *2.7.18*** from https://www.python.org/downloads/.

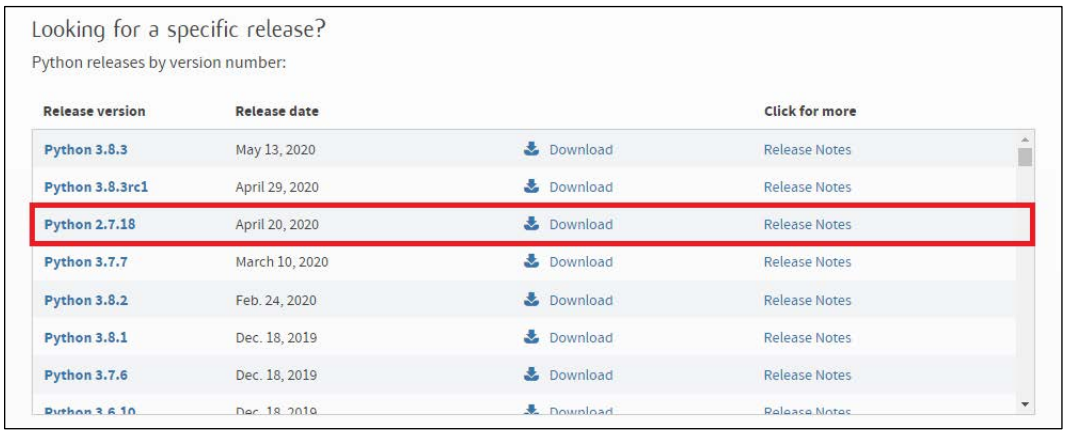
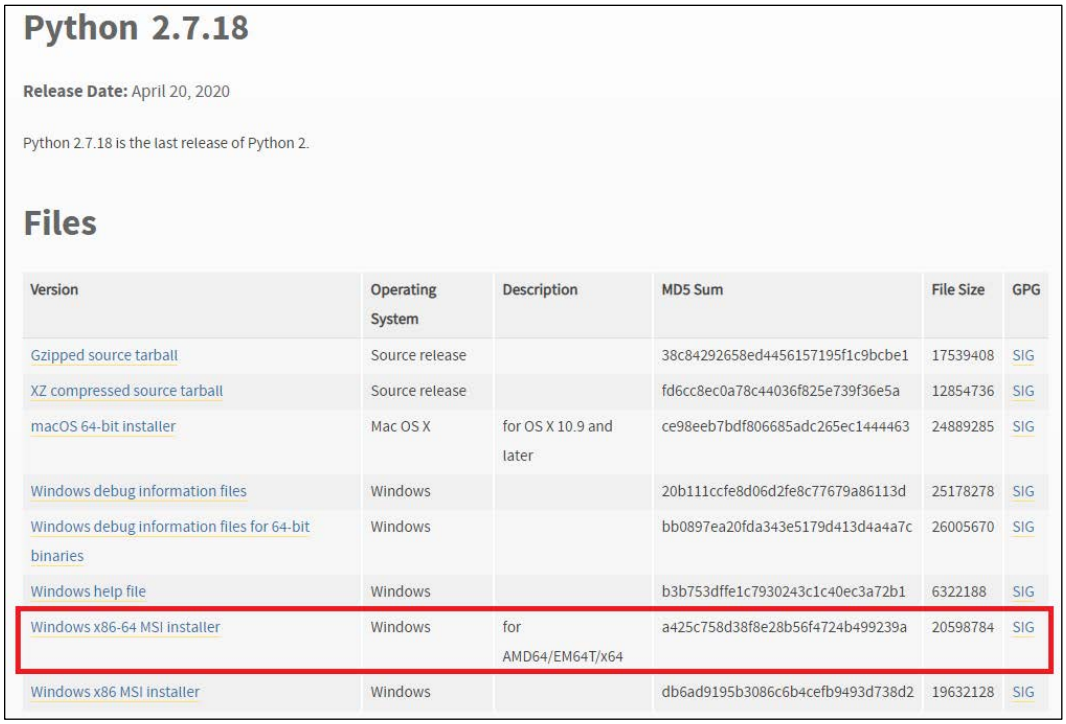


Figure 3. Python download page (https://www.python.org/downloads/).

Step 2: Download the **Windows x86-64 MSI installer** of the python 2.7 (see Figure 4) and install it using the default choices.



Python 2.7.18

Release Date: April 20, 2020

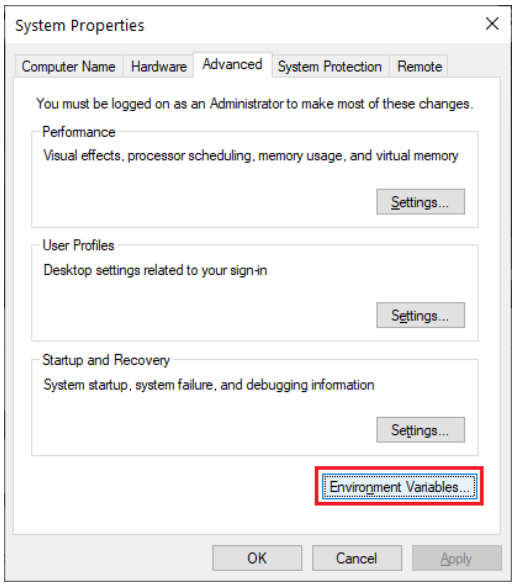
Python 2.7.18 is the last release of Python 2.

Files

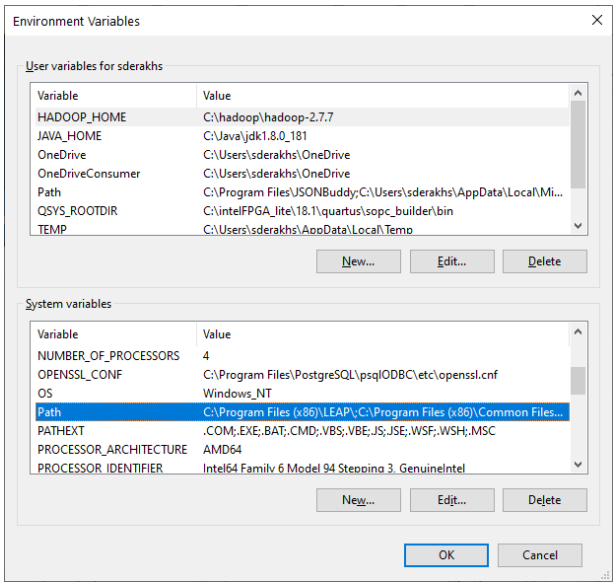
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		38c84292658ed4456157195f1c9bcbe1	17539408	SIG
XZ compressed source tarball	Source release		fd6cc8ec0a78c44036f825e739f36e5a	12854736	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	ce98eeb7bdf806685adc265ec1444463	24889285	SIG
Windows debug information files	Windows		20b111ccfe8d06d2fe8c77679a86113d	25178278	SIG
Windows debug information files for 64-bit binaries	Windows		bb0897ea20fda343e5179d413d4a4a7c	26005670	SIG
Windows help file	Windows		b3b753dffe1c7930243c1c40ec3a72b1	6322188	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	a425c758d38f8e28b56f4724b499239a	20598784	SIG
Windows x86 MSI installer	Windows		db6ad9195b3086c6b4cefb9493d738d2	19632128	SIG

Figure 4. Python 2.7.18 download page (https://www.python.org/downloads/release/python-2718/).

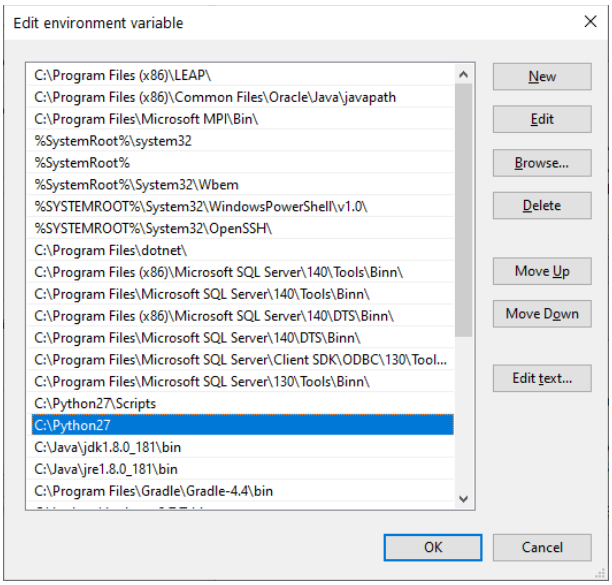
Step 3: Set the environment variable by following the steps shown in Figure 5.



(a) system properties



(b) environments variables



(c) edit environment variables

Figure 5. Add Python path to the Windows Environments Variables.

4.3. TypeScript

Run the following commands using the Windows Command Prompt (cmd) as follows:

Step 1: *npm install typescript --global*

Step 2: *npm install node-gyp --global*

Run the following command using Windows PowerShell (run as administrator)

Step 1: *npm install --global --production windows-build-tools*

Note: Download & Install Visual C++ from here if there is any error in executing the previous step. Also, this step may take a long time (e.g., ~15-30 minutes) and further require multiple runs.

4.4. Git

Download and install the **Git** version control from https://git-scm.com/downloads. At the time of preparing this user guide, the latest version is *2.27.0*. Use the default choices in the installation steps.

Figure 6. Git download page.

4.5. VS-Code

Different IDEs can be used for code development. We recommend the VS-Code editor, and it is used in the rest of this User Guide. Download the Windows version of the VS-Code from https://code.visualstudio.com/Download. Use the default choices in the installation steps.

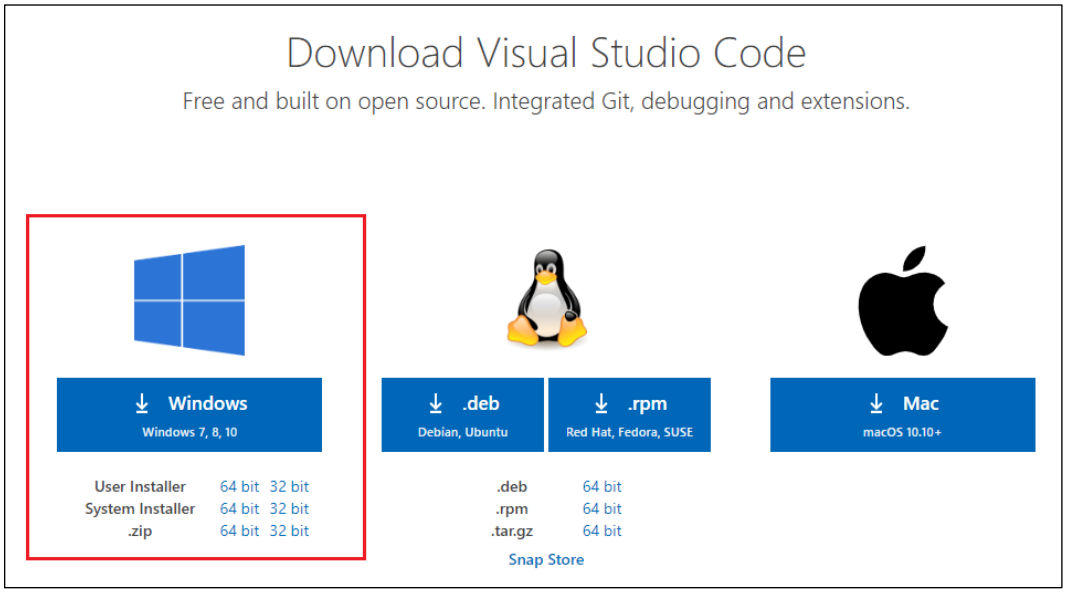


Figure 7. VS-Code Editor download page.

After installing the VS-Code, some extension must be installed (e.g., *TSLint*) and some are recommended to be installed (e.g., *Code Runner*). As shown in Figure 8, open the VS-Code editor, go to the extension page, type **TSLint**, and click on the install button. The same can be done for the **Code Runner** extension.

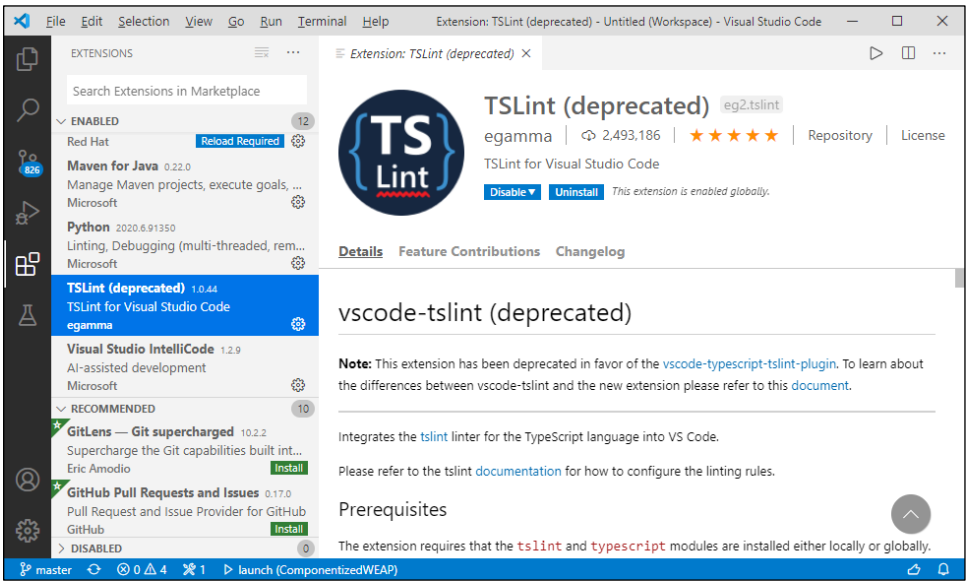


Figure 8. VS-Code editor, extensions page.

Note: The execution policy in the Windows OS Client must be changed to RemoteSigned to be able to run the script. For more information, see About Execution Policy and Set Execution Policy pages. So, open Windows PowerShell (run as administrator), and run:

- Set-ExecutionPolicy RemoteSigned

4.6. Download Source Code

To download the C-WEAP framework from the GitHub in the VS-Code editor, follow the following steps:

1) Open VS-Code
2) Press CTRL+SHIFT+P (View/Command Palette...) and type "*git:Clone*"
3) Enter the C-WEAP git URL (https://github.com/comses/ComponentizedWEAP.git); contact hss@asu.edu for access.
4) Select a folder for the project to be uploaded

After downloading the source code, you should see the folders similar to Figure 9 in the Explorer window of the VS-Code editor.

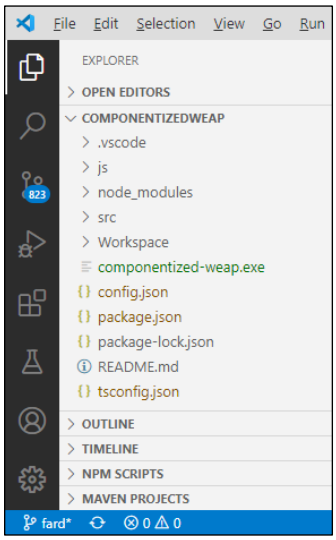


Figure 9. The C-WEAP project directory in the VS-Code editor.

4.7. Update the C-WEAP Packages

After downloading the C-WEAP framework for the first time, or after changing (new updates) any third-party packages (the *dependencies* section of the "**./package.json**" file, see Figure 10), updates to the framework as needed using the following steps:

Step 1: Open VS-Code

Step 2: Right-click on the project folder in the Explorer window (or right clock in the blank area of the project in the Explorer window), and select **Open in Terminal** (as shown in Figure 10).

Step 3: Run the following command in the Terminal (see Figure 10) to update all packages:

- *npm install*

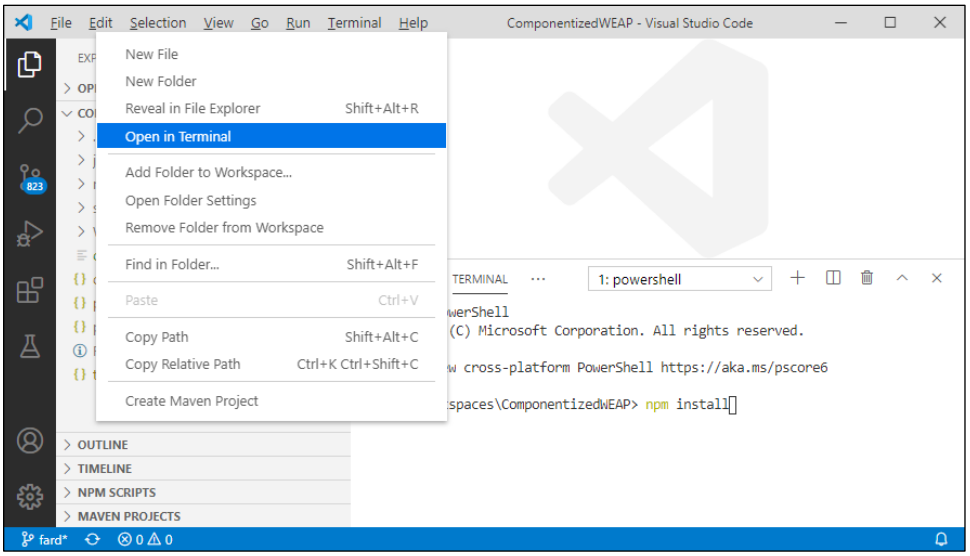


Figure 10. Open Terminal for a project in the VS-Code editor.

4.8. Building the C-WEAP RESTful Framework

The C-WEAP framework is written in TypeScript framework (to have extra facilities which are not available in JavaScript). Finally, the TypeScript files must be converted to the JavaScript files to be able to run on the server using the NodeJS framework. TypeScript does the conversion automatically (using the "**./tsconfig.json**" file) using the following command:

Step 1: Open VS-Code

Step 2: Press CTRL+SHIFT+P (View/Command Palette...) and type "*tasks:run build task*"

Step 3: Select "tsc: build – tsconfig.json" in the opened list

Note: In the C-WEAP RESTful framework, all the TypeScript files are organized under the "**./src**" folder, and all the generated JavaScript files are under the "**./js**" folder (based on the configuration in the *tsconfig.json* file). Also, the conversion is from TypeScript to ES6.

4.9. Running the C-WEAP RESTful Framework

Be sure that all changes in the TypeScript files are converted to JavaScript before running the C-WEAP framework. The configuration to run the project saves in the "**./.vscode/launch.json**" file. The Run page in the VS-Code editor has a run button to execute the launch file. As can be seen in Figure 11, all open projects in the VS-Code are listed. A project must be selected in the drop-down list (e.g., Componentized-WEAP), then click on the start debugging button (▷) to run it.

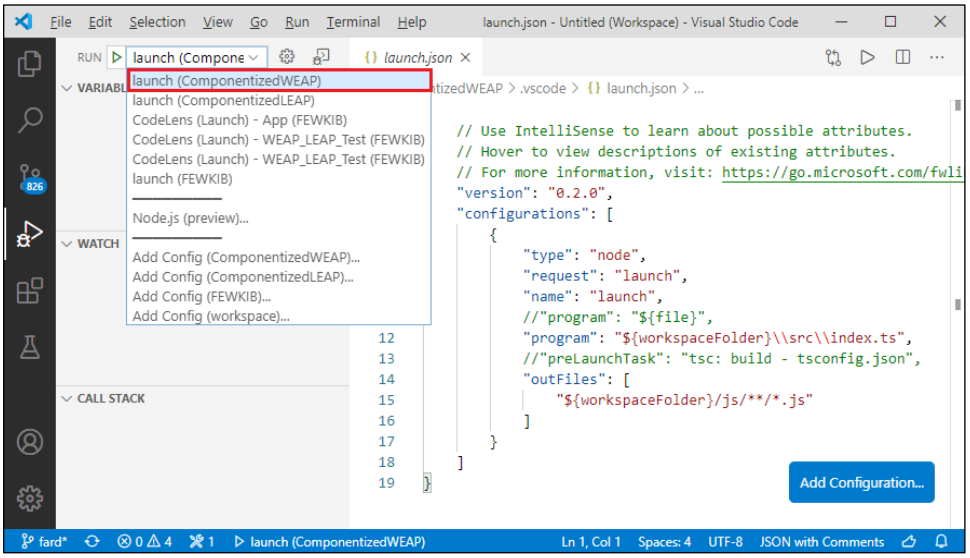


Figure 11. Running a project in the VS-Code editor.

As shown in Figure 12, after running the C-WEAP framework, the "Componentized-WEAP is listening at http://localhost:8080" must be displayed in the DEBUG CONSOLE tab of the editor.

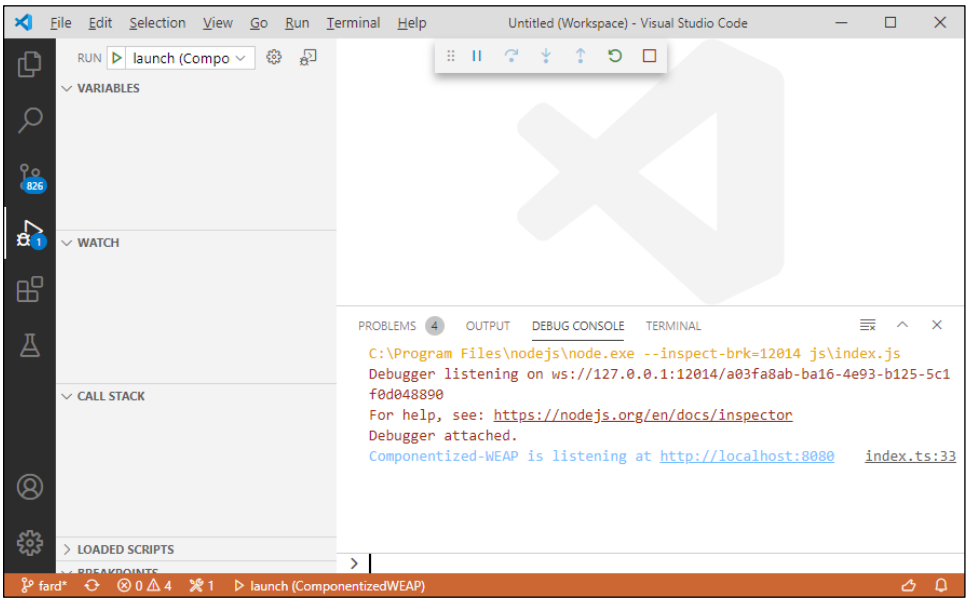


Figure 12. The C-WEAP framework after running in the VS-Code editor.

Note: The WEAP system must be running before running the C-WEAP framework. (Running the C-WEAP prior to running the WEAP system produces unexpected results).

5. Modules

The C-WEAP APIs are categorized into five modules related to different parts of the WEAP system or subset of WEAP's entities. The modules are *Project*, *Version*, *Node*, *Link*, and *Flow*. Each module has a set of APIs to read/write data from/to the WEAP system. The used WEAP's APIs to develop the C-WEAP RESTful framework is listed in Section 6.

The URL patterns for five API types are shown in Table 1. The pattern inside each open and close pair bracket is optional. In the pattern of the URLs, constants are written in *PascalCase* style; parameters start with colons and written in *camelCase* style; query parameters (to apply to some filters on returned data) written after the question mark by *Key=Value* (*camelCase* style for the *Key* part). All URLs start with constant "/Water". The NodeType, LinkType, FlowType, VariableType, and subNodeType (which are bold) in the patterns, must be replaced by a valid value from Table 2. In the Flow URLs, the subNodeType uses to access a specific collection of sub-nodes, and then use *:subNodeName* to select one. For example, the URL "/Water/demo/DemandSites/phoenix" returns the *phoenix* demand site's data of the *demo* project. The data of a variable can be retrieved by mentioning the name of the variable and intended scenario. Query parameters can be used to filter the returned data (the years and time-steps).

Table 1. URL pattern for different types of APIs.

Category	URL Patterns
Project	/Water[/:projectName[/Run]]
Version	/Water/:projectName/Versions[/:versionName/Revert]
Node	/Water/:projectName/**NodeType**[/:nodeName[/**VariableType**[/:variableName/:scenarioName[/Expression][?startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]
Link	/Water/:projectName/**LinkType**[/:sourceName/:targetName[/**VariableType**[/:variableName/:scenarioName[/Expression][?startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]
Flow	/Water/:projectName/**FlowType**[/:flowName[/**subNodeType**[/:subNodeName]][/**VariableType**[/:variableName/:scenarioName[/Expression][?&startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]

Table 2. Type-Values for the patterns of the APIs.

Type	Values
NodeType	Catchments, DemandSites, Groundwaters, Reservoirs, OtherSupplies, WastewaterTreatments
LinkType	Transmissions, Runoffs, ReturnFlows
FlowType	Rivers, Diversions
VariableType	Inputs, Outputs
subNodeType	Reaches, Reservoirs, RunOfRiverHydros, StreamflowGauges, FlowRequirements

All the URLs contain http://(hostname):(port). In our examples, the hostname is "*localhost*", and the port is "*8080*". To test the APIs, the WEAP system and the C-WEAP RESTful framework run, first. Then the APIs are called by the Postman tool. Also, the "*Weaping River Basin*" project is using as the WEAP's project to test the APIs. The Schematic view of this project is presented in Figure 13.

The C-WEAP framework always checks the existence of all parameters (e.g. :projectName, :variableName, etc.) in the URL. For example, the C-WEAP framework first checks the existence of the project "*Weaping River Basin*", then the river "*Nile*", then the input variable "*Headflow*", and finally the scenario "*Current Accounts*" in the URL "http://localhost:8080/Water/Weaping%20River%20Basin/Rivers/Nile/Inputs/Headflow/Current%20Accounts". The correspond error message (with status code 404) will return in the case of not existing a parameter.

Also, for updating APIs, the new values must be set in the body of the request for the URL with PUT methods.

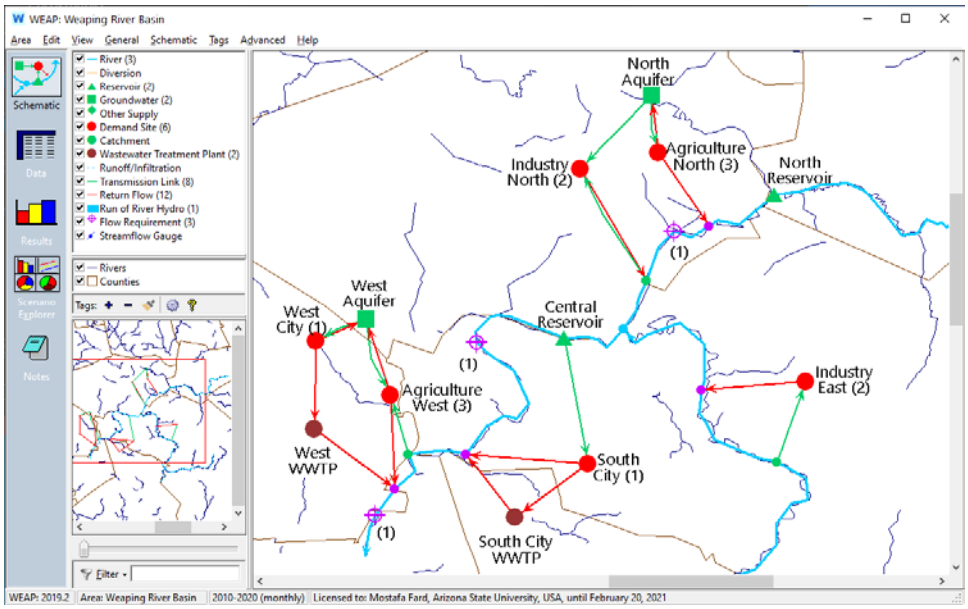


Figure 13. The "*Weaping River Basin*" project in the WEAP system.

Any application can be used to call the APIs (for example, typing a URL in the address bar of a web-browser and hitting the Enter for the GET type requests), but we use the Postman tool (see https://www.postman.com/). As shown in Figure 14, the API method, the URL, the parameters, and the body of the request (for PUT requests) can specify in the Postman (and some other features that we are not going to use them).

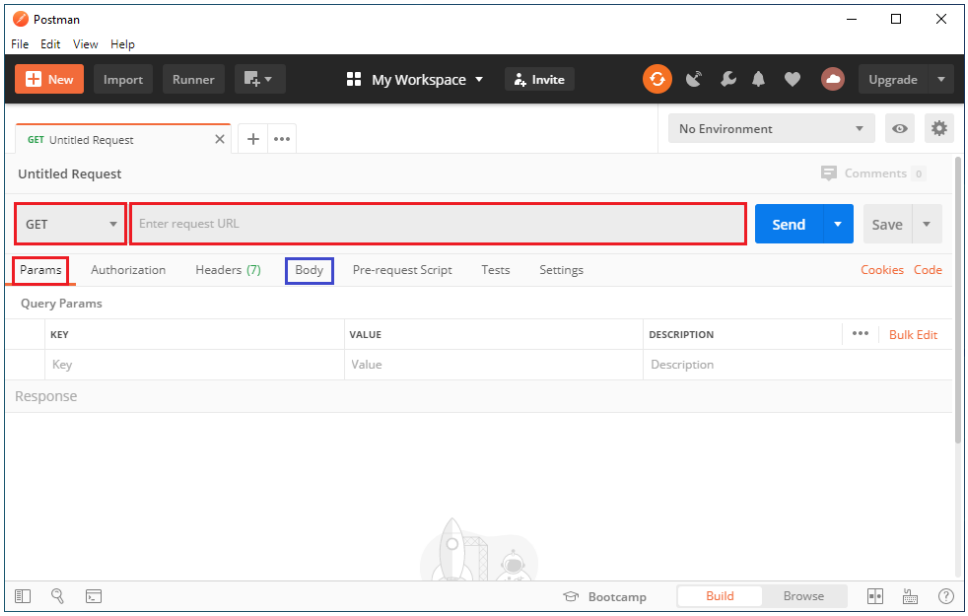


Figure 14. The Postman tool environment.

Note: Using an incorrect URL makes “404 Not Found” response (incorrect hostname, port, constant, etc. in the URL). For example, Figure 15 shows the situation that the URL entered “http://localhost:8080/Watter” by mistake. It shows the message “Cannot GET /Watter” in the web-browser. Indeed, it is requesting an API that is not defined in the webserver.

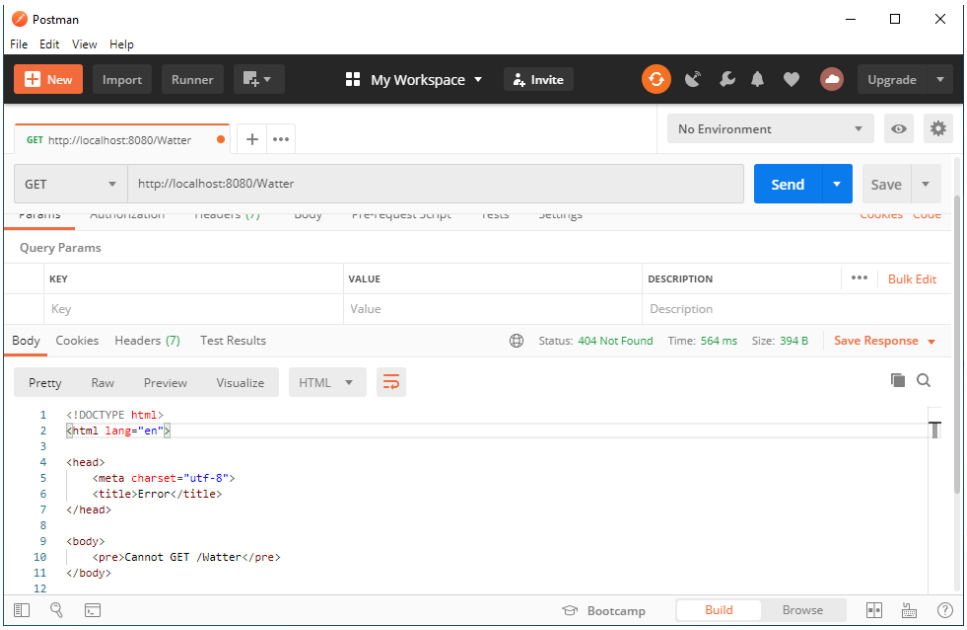


Figure 15. Calling an incorrect URL ("/Watter") in the Postman.

Figure 16 presents individual domain model classes defined in the C-WEAP framework for receiving/sending data from/to the API caller.

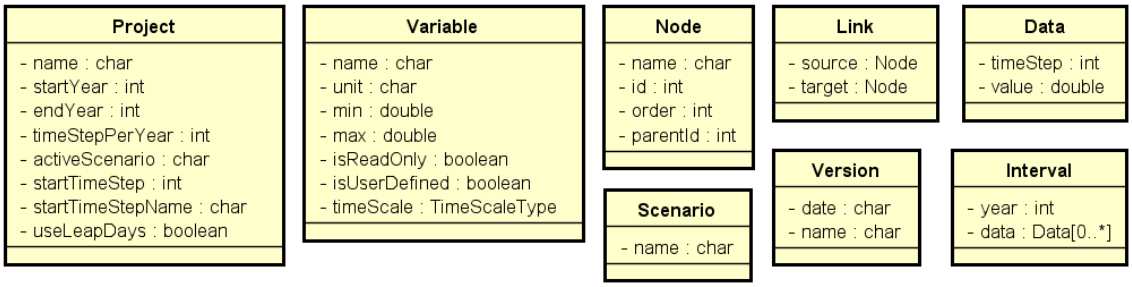


Figure 16. Domain Model classes in the C-WEAP framework.

5.1. Project

The C-WEAP's APIs related to the Project category are listed in Table 3.

Table 3. List of APIs for the Project module.

#	Method	URL	Return Value/s	Description
P1	GET	/Water	String[]	Get the name of all projects
P2	GET	/Water/:projectName	Project	Get properties of a project
P3	GET	/Water/:projectName/Run	Boolean	Run a project
P4	PUT	/Water/:projectName	Boolean	Update properties of a project, by setting new values for the Project object in the body of the request

Example: As an example, the API P1 from Table 3 is presented here. Figure 17 shows the available projects in the WEAP system. Calling the URL "http://localhost:8080/Water" in Postman (or web browser) returns the project names while the C-WEAP is running (see Figure 18). The list of projects shown in Figure 17 vary depending on the projects that are available in the WEAP system. It is shown in the Postman that the status of the request is "200 OK", the time to get data is "128 ms" (which can be different in different calls), and the size of the response is "361 Byte". The time and response time measurements can vary depending on the host computer and other factors.

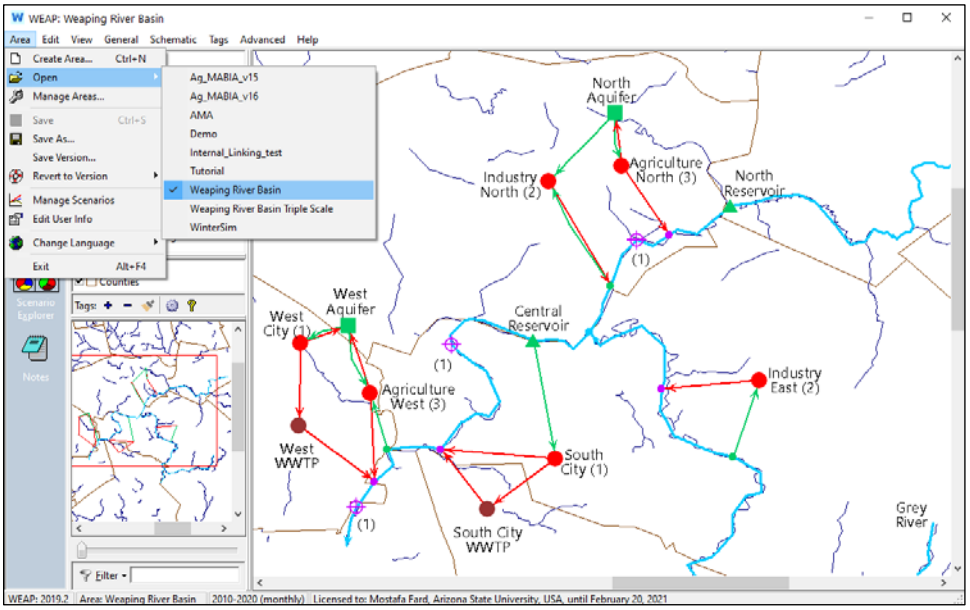


Figure 17. The projects in the WEAP system.

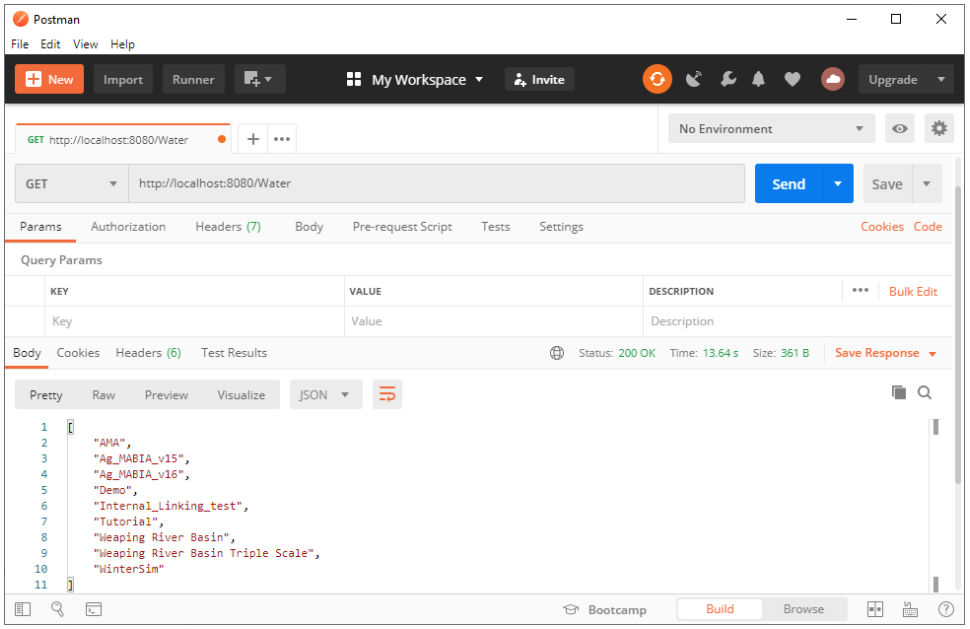


Figure 18. Calling the URL "/Water" in the Postman.

5.2. Version

The C-WEAP's APIs related to the Version category are listed in Table 4. The name of the version is the concatenated of the date and name properties of the Version class in Figure 16.

Table 4. List of APIs for the Version module.

#	Method	URL	Return Value/s	Description
V1	GET	/Water/:projectName/Versions	Version[]	Get the list of all versions of a project
V2	GET	/Water/:projectName/Versions/:versionName	Version	Get a version of a project
V3	PUT	/Water/:projectName/Versions/:versionName/Revert	Boolean	Revert to a version for a project

Example: As an example, the API V1 from Table 4 is presented here. Figure 19 shows the available versions defined in the "*Weaping River Basin*" project in the WEAP system. Calling the URL "http://localhost:8080/Water/Weaping%20River%20Basin/Versions" in Postman (or web browser) returns the list of versions for the project, ordered by ascending on date (see Figure 20).

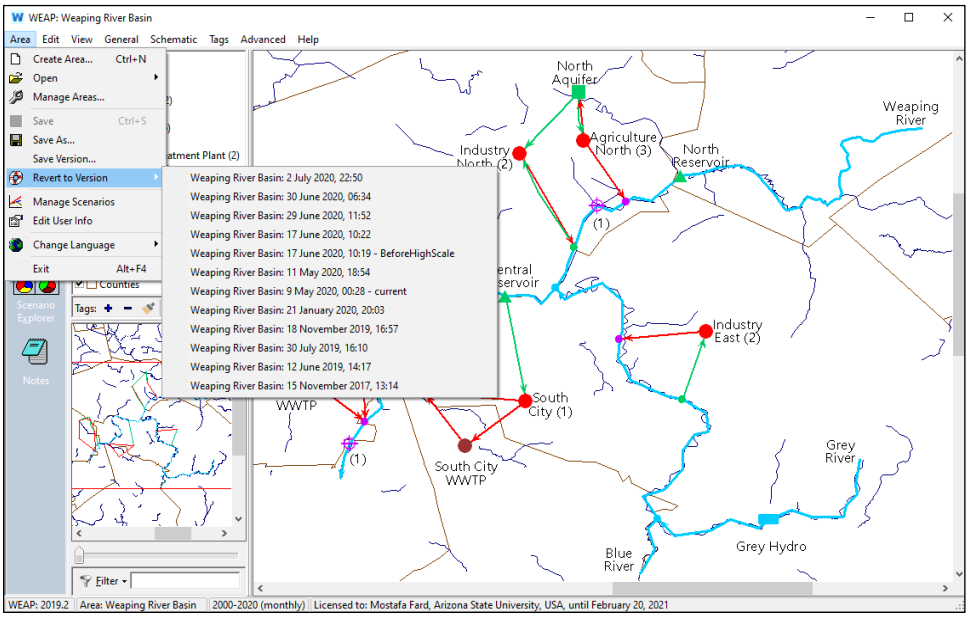


Figure 19. The versions of "*Weaping River Basin*" project in the WEAP system.

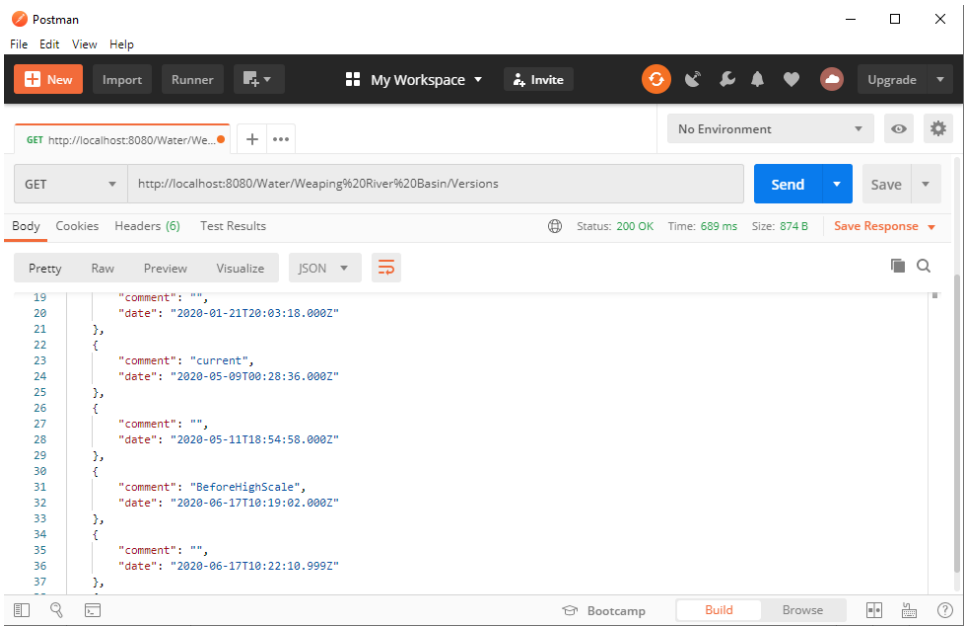


Figure 20. Calling the URL "/Water/Weaping%20River%20Basin/Versions" in the Postman.

5.3. Node

The C-WEAP's APIs related to the Node category are listed in Table 5. As mentioned before, one of the values from Table 2 (Node Type) must be replaced with the **NodeType** in the URLs in Table 5.

Table 5. List of APIs for the Node module.

#	Method	URL	Return Value/s	Description
N1	GET	/Water/:projectName/**NodeType**	Node[]	Get the list of all nodeType components in a project
N2	GET	/Water/:projectName/**NodeType**/:nodeName	Node	Get a nodeType component in a project
N3	GET	/Water/:projectName/**NodeType**/:nodeName/Inputs	Variable[]	Get a list of all input variables of a nodeType component in a project
N4	GET	/Water/:projectName/**NodeType**/:nodeName/Inputs/:variableName	Variable	Get an input variable of a nodeType component in a project
N5	GET	/Water/:projectName/**NodeType**/:nodeName/Inputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an input variable of a nodeType component in a project
N6	GET	/Water/:projectName/**NodeType**/:nodeName/Inputs/:variableName/:scenarioName/Expression	String	Get the expression of an input variable of a nodeType component in a project
N7	PUT	/Water/:projectName/**NodeType**/:nodeName/Inputs/:variableName/:scenarioName	Boolean	Update the values of an input variable of a nodeType component in a project, by setting new values for the in the body of the request
N8	PUT	/Water/:projectName/**NodeType**/:nodeName/Inputs/:variableName/:scenarioName/Expression	Boolean	Update the expression of an input variable of a nodeType component in a project

N9	GET	/Water/:projectName/**NodeType**/:nodeName/Outputs	Variable[]	Get a list of all output variables of a nodeType component in a project
N10	GET	/Water/:projectName/**NodeType**/:nodeName/Outputs/:variableName	Variable	Get an output variable of a nodeType component in a project
N11	GET	/Water/:projectName/**NodeType**/:nodeName/Outputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an output variable of a nodeType component in a project

Note: The returned values for APIs N5 and N11 can be filtered using query parameters. It means, adding the "?&startYear=N&endYear=N&startTimeStep=N&endTimeStep=N" at the end of the URL.

Example: As an example, the API N3 from Table 5 for the DemandSite is presented here. Figure 21 shows the input variables of the "*South City*" demand site in the "*Weaping River Basin*" project in the WEAP system. Calling the URL "http://localhost:8080/Water/Weaping%20River%20Basin/DemandSites/South%20City/Inputs" in Postman (or web browser) returns the list of variables (see Figure 22).

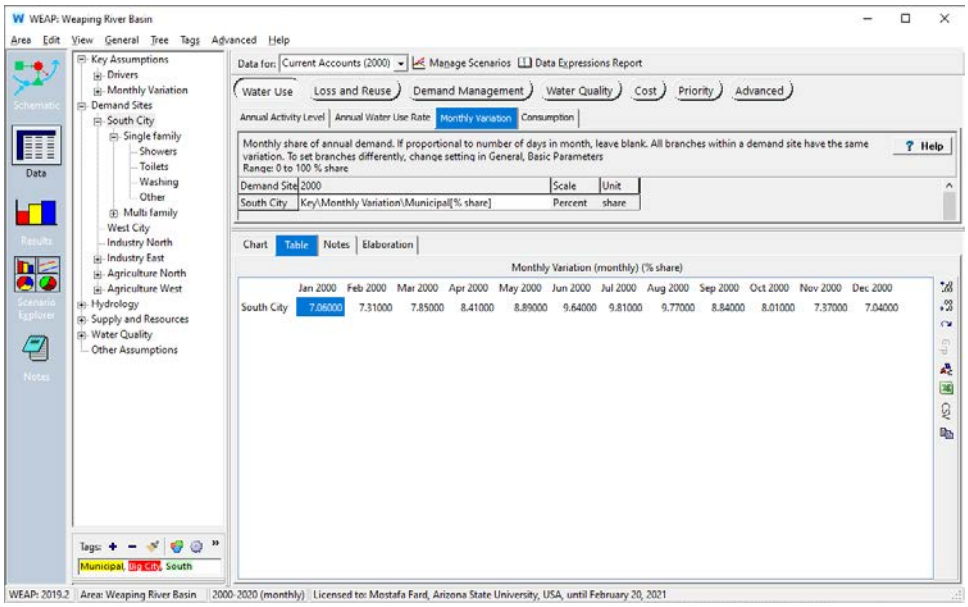


Figure 21. The input variables of the "*South City*" demand site in the "*Weaping River Basin*" project in the WEAP system.

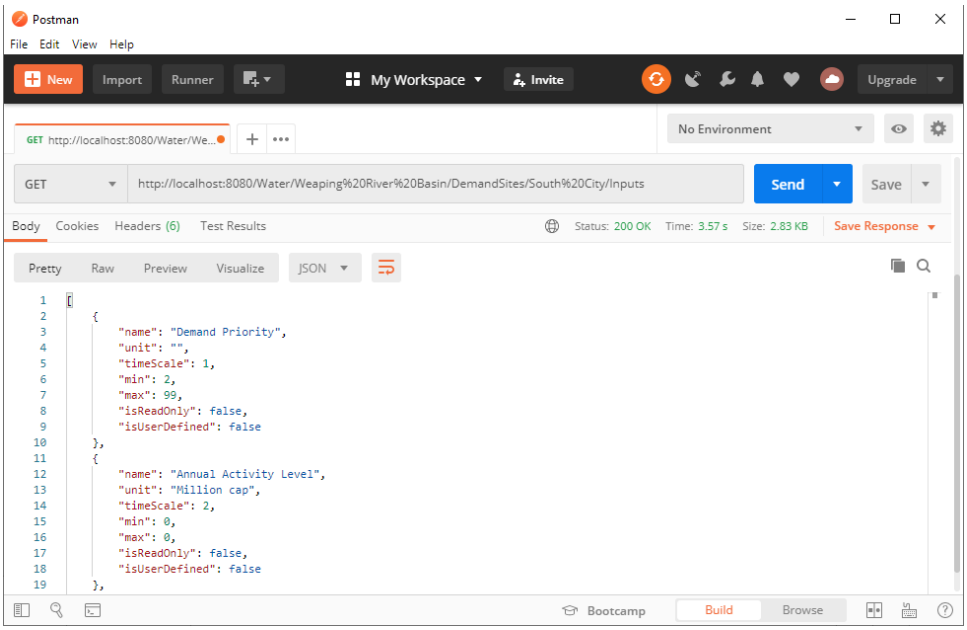


Figure 22. Calling the URL "/Water/Weaping%20River%20Basin/DemandSites/South%20City/Inputs/Monthly%20Variation/Current%20Accounts?&startYear=2000&endYear=2000" in the Postman.

5.4. Link

The C-WEAP's APIs related to the Link category are listed in Table 6. As mentioned before, one of the values from Table 2 (Link Type) must be replaced with the **LinkType** in the URLs in Table 6.

Table 6. List of APIs for the Node module.

#	Method	URL	Return Value/s	Description
L1	GET	/Water/:projectName/**LinkType**	Link[]	Get the list of all linkType components in a project
L2	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName	Link	Get a linkType component in a project
L3	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs	Variable[]	Get a list of all input variables of a linkType component in a project
L4	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs/:variableName	Variable	Get an input variable of a linkType component in a project
L5	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an input variable of a linkType component in a project
L6	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs/:variableName/:scenarioName/Expression	String	Get the expression of an input variable of a linkType component in a project
L7	PUT	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs/:variableName/:scenarioName	Boolean	Update the values of an input variable of a linkType component in a project, by setting new values for the in the body of the request

L8	PUT	/Water/:projectName/**LinkType**/:sourceName/:targetName/Inputs/:variableName/:scenarioName/Expression	Boolean	Update the expression of an input variable of a linkType component in a project
L9	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Outputs	Variable[]	Get a list of all output variables of a linkType component in a project
L10	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Outputs/:variableName	Variable	Get an output variable of a linkType component in a project
L11	GET	/Water/:projectName/**LinkType**/:sourceName/:targetName/Outputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an output variable of a linkType component in a project

Note: Like Node APIs, filtering can be applied on the link APIs L5 and L11 in Table 6.

Example: As an example, the API L11 from Table 6 for the DemandSite is presented here. Figure 23 shows the "*Water Demand*" output variable for the "*Reference*" scenario of the "*Weaping River Basin*" project in the WEAP system. Calling the URL "http://localhost:8080/Water/Weaping%20River%20Basin/DemandSites/South%20City/Outputs/Water%20Demand/Reference?&startYear=2002&endYear=2002" in Postman (or web browser) returns the list of intervals filtered for the year 2002 (see Figure 24).

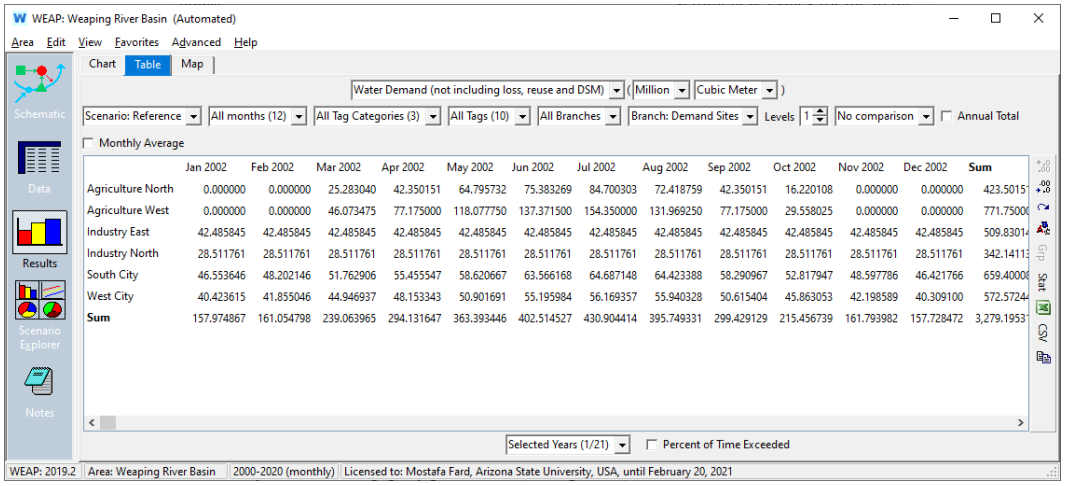


Figure 23. The "*Water Demand*" output variable of the demand sites for the "*Reference*" scenario of the "*Weaping River Basin*" project in the WEAP system.

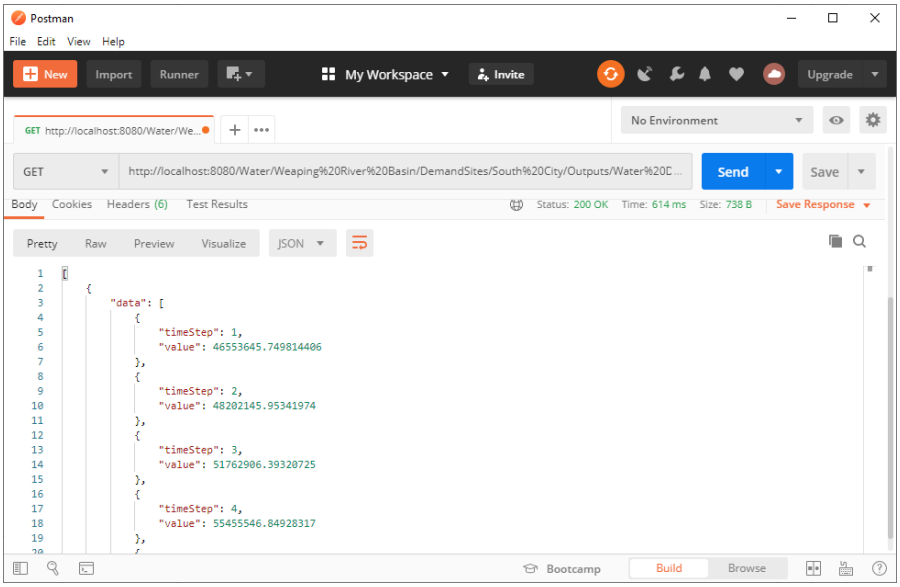


Figure 24. Calling the URL "/Water/Weaping%20River%20Basin/DemandSites/South%20City/Outputs/Water%20Demand/Reference?&startYear=2002&endYear=2002" in the Postman.

5.5. Flow

The C-WEAP's APIs related to the Flow category are listed in Table 7. As mentioned before, one of the values from Table 2 (Flow Type) must be replaced with the **FlowType** in the URLs in Table 7.

Table 7. List of APIs for the Node module.

#	Method	URL	Return Value/s	Description
F1	GET	/Water/:projectName/**FlowType**	Node[]	Get the list of all flowType components in a project
F2	GET	/Water/:projectName/**FlowType**/:flowName	Node	Get a flowType component in a project
F3	GET	/Water/:projectName/**FlowType**/:flowName/Inputs	Variable[]	Get a list of all input variables of a flowType component in a project
F4	GET	/Water/:projectName/**FlowType**/:flowName/Inputs/:variableName	Variable	Get an input variable of a flowType component in a project
F5	GET	/Water/:projectName/**FlowType**/:flowName/Inputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an input variable of a flowType component in a project
F6	GET	/Water/:projectName/**FlowType**/:flowName/Inputs/:variableName/:scenarioName/Expression	String	Get the expression of an input variable of a flowType component in a project
F7	PUT	/Water/:projectName/**FlowType**/:flowName/Inputs/:variableName/:scenarioName	Boolean	Update the values of an input variable of a flowType component in a project, by setting new values for the in the body of the request
F8	PUT	/Water/:projectName/**FlowType**/:flowName/Inputs/:variableName/:scenarioName/Expression	Boolean	Update the expression of an input variable of a flowType component in a project

F9	GET	/Water/:projectName/**FlowType**/:flowName/Outputs	Variable[]	Get a list of all output variables of a flowType component in a project
F10	GET	/Water/:projectName/**FlowType**/:flowName/Outputs/:variableName	Variable	Get an output variable of a flowType component in a project
F11	GET	/Water/:projectName/**FlowType**/:flowName/Outputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an output variable of a flowType component in a project
F12	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**	Node[]	Get the list of all subNodeType components in a flowType components in a project
F13	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName	Node	Get a subNodeType component of a flowType component in a project
F14	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs	Variable[]	Get a list of all input variables of a subNodeType component of a flowType component in a project
F15	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs/:variableName	Variable	Get an input variable of a subNodeType component of a flowType component in a project
F16	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an input variable of a subNodeType component of a flowType component in a project
F17	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs/:variableName/:scenarioName/Expression	String	Get the expression of an input variable of a subNodeType component of a flowType component in a project
F18	PUT	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs/:variableName/:scenarioName	Boolean	Update the values of an input variable of a subNodeType component of a flowType component in a project, by setting new values for the in the body of the request
F19	PUT	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Inputs/:variableName/:scenarioName/Expression	Boolean	Update the expression of an input variable of a subNodeType component of a flowType component in a project
F20	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Outputs	Variable[]	Get a list of all output variables of a subNodeType component of a flowType component in a project
F21	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Outputs/:variableName	Variable	Get an output variable of a subNodeType component of a flowType component in a project
F22	GET	/Water/:projectName/**FlowType**/:flowName/**sunNodeType**/:subNodeName/Outputs/:variableName/:scenarioName	Interval[]	Get a list of all values of an output variable of a subNodeType component of a flowType component in a project

Note: Like Node and Link APIs, filtering can be applied on the Flow APIs F5, F11, F16, and F22 in Table 7.

Example: As an example, the API F12 from Table 7 for the *Rivers* FlowType and *Reservoirs* for the subNodeType is presented here. Figure 13 shows the Schematic view for the "*Weaping River Basin*" project in the WEAP system. As can be seen, there are two reservoirs on the "*Weaping River*" river. Calling the URL "http://localhost:8080/Water/Weaping%20River%20Basin/Rivers/Weaping%20River/Reservoirs" in Postman (or web browser) returns the list of nodes (shown in Figure 25).

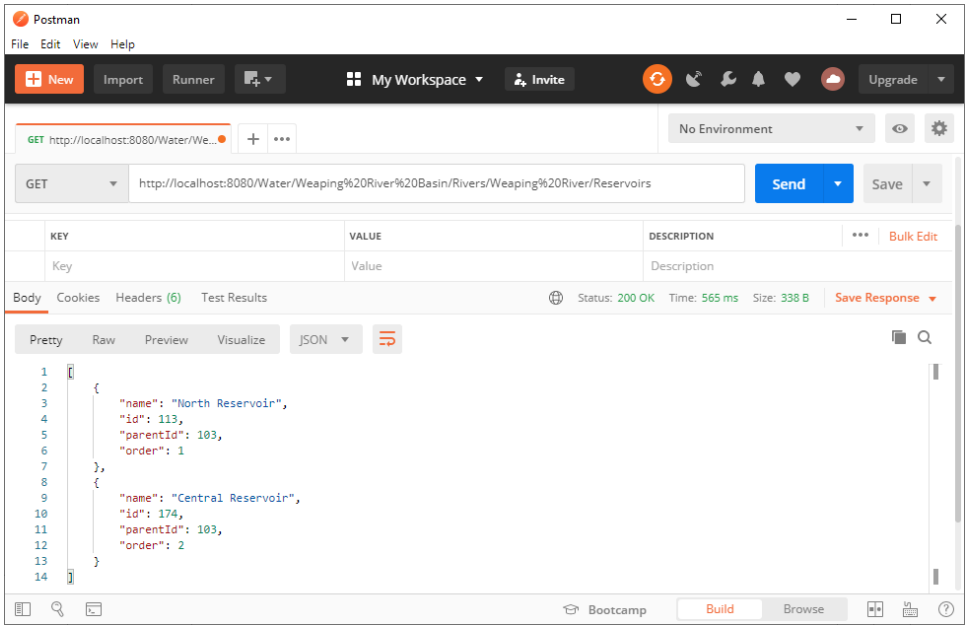


Figure 25. Calling the URL "/Water/Weaping%20River%20Basin/Rivers/Weaping%20River/Reservoirs" in the Postman.

6. The WEAP's APIs used in developing the Componentized-WEAP framework

#	API	Return Object	Category
1	WEAP.ActiveArea	Area	WEAP
2	WEAP.ActiveArea.Name	String	
3	WEAP.WaterYearStart	Integer	
4	WEAP.ActiveScenario	Scenario	
5	WEAP.BaseYear	Integer	
6	WEAP.EndYear	Integer	
7	WEAP.TimeStepName(Id)	String	
8	WEAP.NumTimeSteps	Integer	
9	WEAP.View	String	
10	WEAP.Calculate(LastYear, LastTimestep, AlwaysCalculate)	-	
11	WEAP.ResultValue(BranchName:VariableName, Year, TimeStep, ScenarioName)	Double	
12	WEAP.Areas(Id)	Area[]	Area
13	WEAP.Areas.Count	Integer	
14	WEAP.Versions.Count	Integer	Version
15	WEAP.Versions(Name/Id)	Version	
16	WEAP.Versions.Exist(VersionName)	Boolean	
17	WEAP.SaveVersion(VersionName)	-	
18	WEAP.Versions(VersionName).Revert()	-	
19	WEAP.Scenarios(Id)	Scenario[]	Scenario
20	WEAP.Scenarios.Exists(ScenarioName)	Boolean	
21	WEAP.Scenarios.Add(ScenarioName)	-	
22	WEAP.Scenarios(ScenarioName).Delete()	-	
23	WEAP.Branch(BranchName)	Branch	Branch
24	WEAP.BranchExists(BranchName)	Boolean	
25	WEAP.Branch(BranchName).Children	Branch[]	
26	WEAP.Branch(BranchName).Variables	Variable[]	
27	WEAP.Branch(BranchName).Variables.Exists(VariableName)	Boolean	

7. References

[1] ACIMS, "Componentized-WEAP RESTful Framework," 20 June 2020. [Online]. Available: https://acims.asu.edu/software/c-weap. [Accessed 20 June 2020].

[2] M. D. Fard and H. S. Sarjoughian, "A Web-service Framework for the Water Evaluation and Planning System," *Spring Simulation Conference (SpringSim)*, pp. 1-12, 2019.

[3] M. D. Fard and H. S. Sarjoughian, "Coupling WEAP and LEAP Models using Interaction Modeling," in *SpringSim Conference*, Fairfax, VA, USA, 2020.