



**SEATTLE
UNIVERSITY**

User Manual

Energy Kiosk Creator

**ECE 21.3
Science and Engineering Project Center
College of Science and Engineering
Seattle University**

TABLE OF CONTENTS

A. List of Figures	iii
B. List of Tables	iii
I. Introduction	1
II. Installing EKC.....	1
III. Making Modifications to the Source Code	6
A. Calculating Environmental Variables	7
1. Extraterrestrial Solar Position and Intensity	7
2. Hourly Surface-Level Insolation.....	7
3. Hourly Ambient Temperature.....	8
B. Understanding EKC's User Interface	11
IV. How-To Edit EKC's Library	11
V. How-To Use EKC.....	13
VI. Recommended Updates	15
VII. Bibliography	Error! Bookmark not defined.
VIII. Appendix.....	18
A. PV Output Calculation Models.....	31
i. Insolation	31
ii. PV Cell Temperature	32
iii. PV Power	33
B. Battery State of Charge Calculation Models	33

A. List of Figures

Figure 1. Code Guide.....	11
Figure 2. Template of Inserting New Models in Component CSVs.....	12
Figure 3: Victron 12V Battery Using Relative Capacities.....	12
Figure 4: Victron 6V Battery Using Absolute Effective Capacities.....	12
Figure 5. Output Window of EKC.....	15
Figure 6. PV Array and Solar Angles.	32

B. List of Tables

Table 1. Directory Guide for EKC.....	3
Table 2: EKC-Specific Functions	18
Table 3: Main SOLPOS Functions	21
Table 4: Local solpos.c Functions	22
Table 5: Struct posdata Variables	23

I. Introduction

This manual serves as an aid in using the software application Energy Kiosk Creator (EKC). The application is straightforward in usage when it comes to inputting values and viewing the results, but this manual provides a more in-depth version of the About window found in EKC.

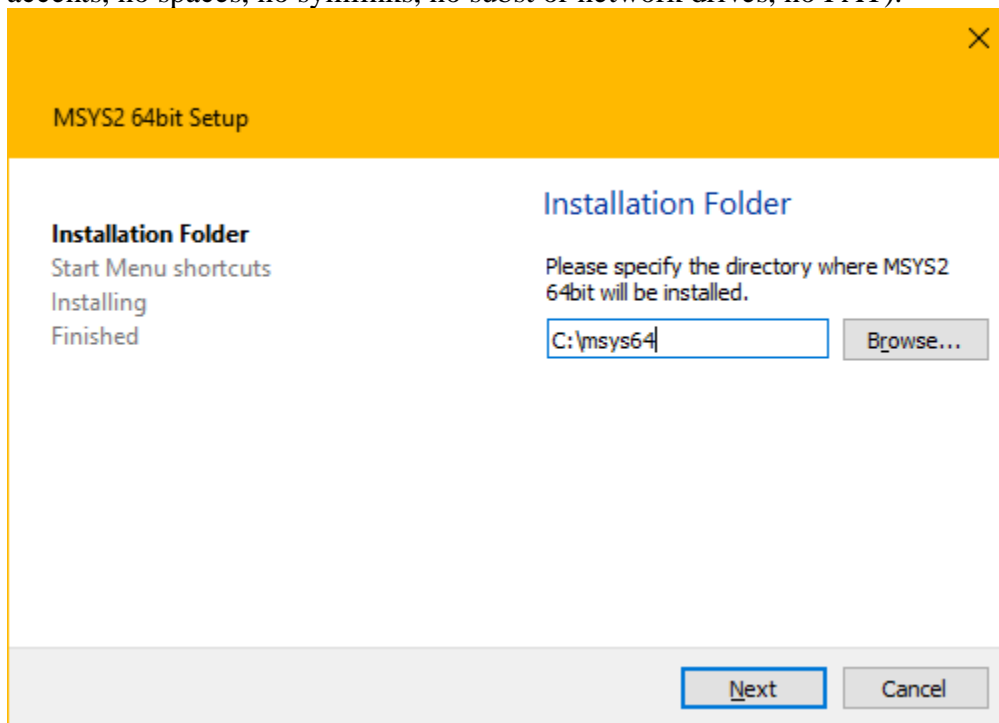
- All geometric calculations are computed in degrees.

II. Installing EKC

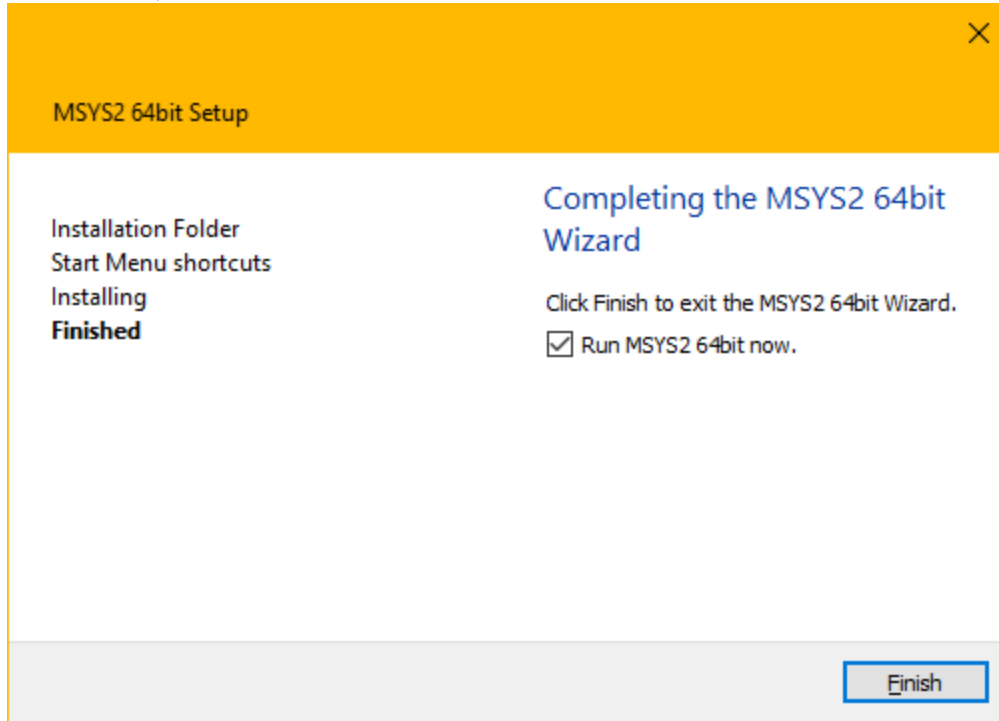
A. EKC with User Interface

To use EKC with the user interface, MSYS2 must be installed. The link that the MSYS2 install instructions come from can be found here: <https://www.msys2.org/>. We recommend that the user follows the instructions on the MSYS2 website because they may be updated and provide embedded objects that are not included here.

1. Download the MSYS2 installer from Github: https://github.com/msys2/msys2-installer/releases/download/2021-07-25/msys2-x86_64-20210725.exe.
2. Run the installer. MSYS2 requires 64 bit Windows 7 or newer.
3. Enter your desired Installation Folder (short ASCII-only path on a NTFS volume, no accents, no spaces, no symlinks, no subst or network drives, no FAT).




4. When done, tick Run MSYS2 now.



5. Update the package database and base packages. Unless your setup file is very recent, it will take two steps. First run "pacman -Syu".
6. Run "MSYS2 MSYS" from Start menu. Update the rest of the base packages with "pacman -Su".
7. Now MSYS2 is ready for you. Install some tools and the mingw-w64 GCC to start compiling: "pacman -S --needed base-devel mingw-w64-x86_64-toolchain".
8. To start building using the mingw-w64 GCC, close this window and run "MSYS MinGW 64-bit" from Start menu. Now you can call make or gcc to build software for Windows. This step is unnecessary at the moment, but MSYS2 MinGW 64-bit will need to be run later to run EKC.

Once MSYS2 has been installed, download EKC from Github at <https://github.com/osheas21/Energy-Kisok-Creator>.

1. Click on the green "Code" button  to open the drop down menu. Then click "Download ZIP" to download a .zip folder of all of EKC's components.
2. Extract all files to your downloads folder, then navigate to the folder "Energy-Kiosk-Creator-main".
3. Copy all the folders and files within the "Energy-Kiosk-Creator-main" directory, then navigate to the location that you installed MSYS2 to and go to the "home" directory within msys64 (i.e. "..\msys64\home\"). Paste the folders and files that you copied from "Energy-Kiosk-Creator-main" within the "..\msys64\home" folder.

The application is now ready to be run. To run EKC with the user interface, search "MSYS2 MinGW 64-bit" from the Start Menu and run it. Once a dialog box opens, type "./ekc" and EKC will launch.

You may want to add a shortcut to the desktop if you do not prefer to search for items in the start menu. To add a shortcut to “MSYS2 MinGW 64-bit” search for it in the Start Menu, right click on “MSYS2 MinGW 64-bit”, and click on “Open File Location”. This will take you to where the MinGW 64-bit shortcut has been installed (directory should look something like: “C:\Users_user_x_\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\MSYS2 64bit”). Click and drag the shortcut icon onto your desktop. You can now launch MSYS2 MinGW 64-bit by double clicking the icon on your desktop.

The installment process of EKC requires Windows 10. When the application is finished installing, please ensure the following directories and files are located where they are needed:

Table 1. Directory Guide for EKC.

Directory	Files Contained in Directory
<i>EKC Main Directory</i>	<ol style="list-style-type: none"> 1. Library 2. Glade 3. res 4. src 5. Makefile 6. Compiled File
<i>Library</i>	<ol style="list-style-type: none"> 1. Backup Library (extra directory containing the same files as the Library) 2. PV Modules.csv 3. Batteries.csv 4. Charge Controller.csv 5. Inverter.csv 6. Load Profile.csv
<i>glade</i>	<ol style="list-style-type: none"> 1. window_main.glade 2. window_main.glade~
<i>res</i>	Photos found in the applciation
<i>src</i>	<ol style="list-style-type: none"> 1. main.c 2. solpos.c 3. solpos.h

Each file plays a massive role in how the application runs. If a file is missing or if the name of a file changes, the application will not run as intended and will not print the results as ECE 21.3 expected. The *res* directory is only photos that are embedded in the application, this directory is not important, and does not play a role in the modeling of solar energy kiosks.

B. EKC Command Line Edition (Not Recommended)

This version effectively trades the user interface (UI) in favor of the header file “Entities.h”. This version is not recommended because it currently lacks the ability to connect to the internet and download data. This version also does not provide any sort of cost estimation. This edition has been included as a way to make it easier for any future modifications to the application to be made and can allow for faster comparison of different energy kiosks. The lack of an interface also eliminates many glitches and complexities that come with a user interface.

To install, follow the instructions above for EKC with User Interface, then simply copy (or cut) and paste the contents of the EKC Command Line Edition wherever is most convenient to you.

We recommend that Microsoft Visual Studio be used, but an equivalent code editor will work for EKC Command Line Edition. To use, open the EKC Command Line Edition with your preferred code editor. Then adjust the macros to align with the proposed energy kiosk design. Once data has been downloaded to “Weather Data.txt” (see “Acquiring Data” instructions below), the user must compile the program with “gcc EKC.c solpos.c” and run the program with “./a.exe”. The results will be displayed in the terminal.

A number of different metrics can be viewed by modifying the blocks of code that has been commented out near the end of the program. These blocks of code have been provided by ECE 21.3, but this part of the code is highly customizable and allows the user to choose exactly which metrics they would like displayed.

i. Acquiring Data

Before a simulation is run, data must be downloaded from NASA POWER. **The simple way of doing this is to run a simulation with the EKC user interface edition**, then copy the file “Weather Data.txt” from the “src” folder.

If the UI is not available, the alternative to this is to download the data directly from NASA POWER. To do this, the user must visit the [NASA POWER Data Access Viewer](#) website and **enter the latitude and longitude** of the future location in the entry box to the left. The user must then **change the time frame to begin at 01/01/1984 and end at 12/31/2005**. Any other time frame will cause errors or miscalculations of the performance of the energy kiosk.



4. Select Time Extent

Start Date 01/01/1984 (MM/DD/YYYY)

End Date 12/31/2005 (MM/DD/YYYY)

Figure 1: NASA POWER Date Range

Next, the user will need to select “**Insolation Clearness Index**”, “**Maximum Temperature at 2 Meters**”, “**Minimum Temperature at 2 Meters**”, and “**Temperature at 2 Meters**” as is shown in the figure below. If these specific entries are not selected, the application will not work properly.

6. Select Parameters (Limit 20 parameters)

The Climatology temporal period has the most parameters.
Double-click folders to expand and show available parameters.

Search Parameters

Meteorology (Moisture and Other)

Meteorology (Temperature)

Temperature Range at 2 Meters

Earth Skin Temperature

Dew/Frost Point at 2 Meters

Wet Bulb Temperature at 2 Meters

Maximum Temperature at 2 Meters

Minimum Temperature at 2 Meters

Temperature at 2 Meters

Meteorology (Wind)

Sizing and Pointing of Solar Panels and for Solar Thermal Applications

Insolation Clearness Index

Clear Sky Insolation Incident on a Horizontal Surface

All Sky Insolation Incident on a Horizontal Surface

Solar Cooking

Thermal Infrared Parameters

Tilted Solar Panels

Figure 2: NASA POWER Data Selection

All other inputs should be left the same. Press “Submit” at the bottom of the input window and wait until some graphs show up. Then scroll to the bottom of that window and click on the light blue “here” to access the URL that was used to construct the data.

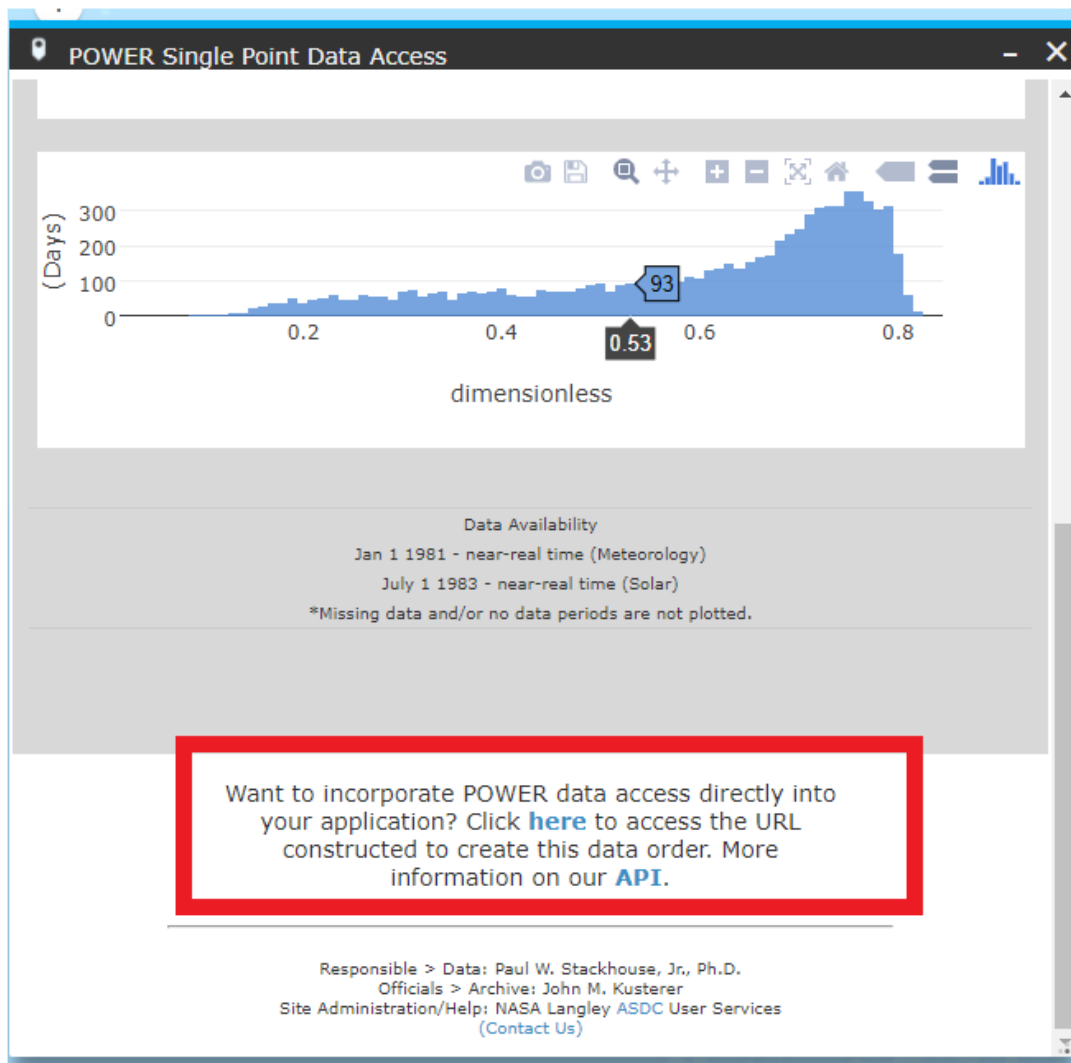


Figure 3: Access the URL That Contains the Data

Select everything in the tab that opens, copy the data, and paste it into the file “Weather Data.txt” within the EKC Command Line Edition folder. EKC Command Line Edition will then be ready to run simulations based on the data that has been saved into the “Weather Data.txt” file.

ii. Specifying Parameters for an Energy Kiosk

To enter the required inputs, the user must access the “Entries.h” file. Within this file there are various definitions for macros such as longitude, latitude, and PV and battery array size. The numbers should be changed to reflect the future energy kiosk, but the macro names must not be modified.

III. Making Modifications to the Source Code

If a user would like to modify EKC, they will need to go through the steps for setting up [GTK](#), a free and open-source cross-platform widget toolkit, that allows EKC to run in a Windows

environment. The application also uses the library cURL (libcurl) that allows EKC to download data from NASA's POWER API. Without libcurl, the application will not be able to compile without causing errors of failure to find library. In order to [install libcurl](#) to use with GTK, the user must use the MSYS2 MSYS compiler (that comes from the GTK download), NOT MSYS2 MinGW 64-bit (as it is used for compiling and running programs, rather than editing the compiler settings), to unpack the libcurl package found in the directories of the msys64.

A. Calculating Environmental Variables, Power Produced, and State of Charge

iii. Extraterrestrial Solar Position and Intensity

The Solar Position and Intensity calculator (SOLPOS) is a program provided by the National Renewable Energy Laboratory (NREL) that can be used to calculate the extraterrestrial irradiance, angle of the sun, sunset, and contains many other functions that are not currently used in EKC. SOLPOS serves as the foundation upon which the application is built. The solar constant *solcon* used is 1367W/m². *S_init()* should be used to initialize all of the values in the *posdata* struct, while *S_solpos()* runs the solar position and intensity calculations. *S_decode()* can be used after *S_solpos()* to return potential invalid inputs.

Calculation of declination angle *d*:

$$d = 360 \times \frac{(\text{daynum} - 1)}{365}$$

Calculation of solar azimuth angle *zenetr*:

$$\text{zenetr} = \sin(d) \times \sin(\text{latitude}) + \cos(d) \times \cos(\text{latitude}) \times \cos(\text{hour-angle})$$

Calculation of extraterrestrial irradiance on a surface normal angle of the incidence *etrn* is calculated using the earth-radius vector *erv* which is detailed in lines 428-453 of *solpos.c*.

$$\text{etrn} = \text{solcon} \times \text{erv}$$

The extraterrestrial irradiance on a horizontal surface *etr* is then calculated using *etrn* and the solar zenith angle after refraction has been accounted for *zenref*. The calculation for refracted solar zenith angle can be found in lines 765-804 of *solpos.c* in the "refrac" function. Adjusting for refraction has a negligible effect on the PV output, but it is provided by NREL, so it is used.

$$\text{etr} = \text{etrn} \times \cos(\text{zenref})$$

The extraterrestrial irradiance on a tilted surface *etrtilt* is then calculated:

$$\begin{aligned} \text{etrtilt} = & \cos(\text{zenref}) \times \cos(\text{tilt}) \\ & + \sin(\text{zenref}) \times \sin(\text{tilt}) \\ & \times [\cos(\text{azim}) \times \cos(\text{aspect}) + \sin(\text{azim}) \times \sin(\text{aspect})] \end{aligned}$$

where *tilt* is the angle from horizontal that the solar panels are tilted, *azim* is the solar azimuth angle, and *aspect* is the solar panel azimuth angle.

iv. Hourly Surface-Level Insolation

This function calculates the average irradiance on a tilted surface over the course of an hour. The extraterrestrial irradiance at minute 0, 15, 30, and 45 of each hour are used to approximate the

average irradiance throughout the hour. More frequent increments could be used, but the program would run more slowly. This function calculates the hourly extraterrestrial insolation on a tilted surface with units of W/m²/hr.

Once the extraterrestrial irradiance on a tilted surface has been calculated, clearness index K_T data from NASA POWER is applied to the extraterrestrial irradiance to calculate the irradiance on a tilted surface at the surface of the Earth. The insolation at the surface of the Earth is stored in the *insolation* array within the *DataValues* struct.

$$insolation = calcAvgIrradianceTilt(struct posdata *pdat) \times K_T$$

Note: clearness index is constant throughout every day and is not adjusted by the hour. This means that intraday changes in weather will not be accounted for and may lead to an underestimation in the number of charge cycles per year.

v. Hourly Ambient Temperature

Hourly ambient temperature *temp* is calculated from a cosine function:

$$temp = \frac{range}{2} \times \cos \left[\frac{2 \times \pi}{24} \times (hour - (sunsethour - 3)) \right] + \frac{maxTemp + minTemp}{2}$$

where *range* is the difference between the maximum and minimum temperature *maxTemp* and *minTemp*, respectively. Dividing the range by 2 creates a cosine function that has an amplitude of half of the range, allowing for a function that reaches its minimum and maximum value equal to the minimum and maximum temperature, respectively.

$\frac{2 \times \pi}{24}$ is the frequency of the function, calibrated to make the period last 24 hours.

The $hour - (sunsethour - 3)$ portion of the function is designed to make *temp* peak 3 hours before sunset, where *hour* is the hour of the day and *sunsethour* is the hour of the day that the sun sets, calculated from *S_solpos()*. The function's temperature peak at 3 hours before sunset is based on observations of historical data that show that temperature typically peaks 3 hours before sunset [1].

The curve is offset by $\frac{maxTemp + minTemp}{2}$. This creates a function that will have an accuracy within 1 degree Celsius in the middle of the day when the temperature is highest. This function does not account for odd weather conditions that may cause the actual hourly temperature to peak at any time other than 3 hours before sunset.

This function can be found as *calcAmbientTemp()* within the *solpos.c* file. Daily average, minimum, and maximum temperature data is download from NASA POWER.

vi. PV Cell Temperature

PV cell temperature is calculated in the *calcCellTemp()* function. Cell temperature *cellTemp* is calculated from the nominal operating cell temperature (NOCT) *noct*, ambient temperature *ambientTemp*, and irradiance *irradiance*.

$$cellTemp = ambientTemp + (noct - 20) \times \frac{irradiance}{800}$$

vii. PV Power Output

The power produced during each hour of the day is calculated in the *calcPVPower()* function. The function uses inputs of the panel's rated maximum power *stcPower*, insolation *insolation*, power temperature coefficient *alphaP*, and the temperature of the module *cellTemp*. These inputs are used to calculate the power that would be produced throughout the hour *power* which converts into Wh through a simple 1:1 conversion.

$$power = stcPower \times \frac{irradiance}{1000} \times \left[1 + \frac{alphaP}{100} \times (cellTemp - 25) \right]$$

The power output by the PV array is then adjusted for the charge controller efficiency by multiplying the power produced by the efficiency of the charge controller.

viii. Hourly Load

The load profile of the energy kiosk that is stored in "Load Profile.csv" is read into EKC through the *importLoadProfile()* function. *importLoadProfile()* stores the load profile in the *loadProfile* variable within the *DataValues* struct. The current flowing into and out of the battery is calculated and stored as *loadCurrent*, where positive values indicate a discharging of the battery and negative values indicate a recharging of the battery. The current flowing into or out of the battery is calculated from the equation:

$$loadCurrent = \frac{loadProfile - hourlyCControllerOutput}{arrayVoltage}$$

where *loadProfile* is the power that is being consumed by the energy kiosk, *hourlyCControllerOutput* is the power being produced by the PV array after accounting for charge controller losses, and *arrayVoltage* is the voltage of the battery array.

ix. State of Charge

At the beginning of the first hour of simulation, the state of charge of the battery is set at the midpoint of the minimum state of charge (specified in "Batteries.csv") and maximum state of charge (100%).

EKC first calculates the additional current that will be needed to account for inverter losses. Observations have shown that nearly all of the inverter losses are due to the zero-load power consumption of the inverter when the current is less than 10% of the nominal inverter capacity [2]. Under these relatively small loads, the additional current out of the battery due to the inverter is calculated as the zero-load power of the inverter *zeroLoadPower* divided by the voltage of the battery array *arrayVoltage*.

$$\text{additional current} = \frac{zeroLoadPower}{arrayVoltage}$$

Note: Lower inverter efficiencies due to high temperatures are not accounted for in EKC.

a. Discharging

EKC first calculates the additional current that will be needed to account for inverter losses. Observations have shown that nearly all of the inverter losses are due to the zero-load power consumption of the inverter when the current is less than 10% of the nominal inverter capacity. Under these relatively small loads, the additional current out of the battery due to the inverter is calculated as the zero-load power of the inverter *zeroLoadPower* divided by the voltage of the battery array *arrayVoltage*.

$$\text{additional current} = \frac{\text{zeroLoadPower}}{\text{arrayVoltage}}$$

When the current flowing out of the battery *loadCurrent* is positive, EKC uses Peukert's equation to calculate how much the state of charge will decrease over the course of the hour.

$$\text{chargeCapacity} = \text{arrayEffectiveCapacity} \times \left(\frac{\text{arrayDischargeCurrent}}{\text{currentOut}} \right)^{\text{peukert}-1}$$

In this equation, *chargeCapacity* is the absolute change in capacity that the battery array will undergo, measured in amp-hours. *arrayEffectiveCapacity* is the known capacity of the battery array at the known discharge current *arrayDischargeCurrent* that comes from "Batteries.csv" and the battery's datasheet. The Peukert exponent *peukert* is calculated from two known discharge capacities *arrayEffectiveCapacity₁*, *arrayEffectiveCapacity₀* and two known discharge currents that correspond to the discharge capacities *arrayEffectiveCurrent₁*, *arrayEffectiveCurrent₀*.

$$\text{peukert} = \frac{\log \left(\frac{\text{arrayEffectiveCapacity}_1}{\text{arrayEffectiveCapacity}_0} \right)}{\log \left(\frac{\text{arrayEffectiveCurrent}_0}{\text{arrayEffectiveCurrent}_1} \right)} + 1$$

EKC uses this equation to calculate the Peukert exponent through a logarithmic interpolation of known discharge values. After the Peukert exponent is calculated, the Peukert exponent is used to calculate the change in capacity of the battery *chargeCapacity* from the first equation mentioned in this section using a known capacity and current for the nearest to the calculated *currentOut*. *chargeCapacity* is then used to calculate *cRate*, the change in the state of charge of the battery over the course of the hour. *cRate* is calculated by dividing the current out of the battery *currentOut* by the absolute change in capacity *chargeCapacity*.

$$\text{cRate} = \frac{\text{currentOut}}{\text{chargeCapacity}}$$

The final state of charge of the battery is then calculated by subtracting *cRate* from the state of charge at the beginning of the hour.

$$\text{endSoC} = \text{startSoC} - \text{cRate}$$

endSoC is the state of charge of the battery at the end of the hour.

b. Charging

If the current out of the battery *currentOut* is less than or equal to 0, the battery is charging. EKC first accounts for the power consumed from the inverter by adding the current needed to power the inverter to the negative current flowing out of the battery.

B. Understanding EKC's User Interface

EKC was designed in a user interface (UI) designer called Glade. Glade is Rapid Application Development (RAD) tool that enabled a quick and easy development of the UI for EKC. The UI's designed in Glade are saved as XML extensions and can be found in the *Glade* directory of the application. By using GtkBuilder with GTK object, the UI's built can be loaded by applications dynamically as needed. To learn the usage of GTK and Glade together, ECE 21.3 referenced the website [ProgNotes](#) which allowed the team to go through multiple tutorials to understand how Glade and GTK work together.

When creating signal-handling functions, the functions created within the code will need *G_MODULE_EXPORT* to be placed in front of the functions, see Figure for reference. This will allow EKC to compile properly in the MSYS2 MinGW 64-bit. Other normal functions that do not require signal-handling, does not require *G_MODULE_EXPORT*.

```
void curlToFile (char *url, FILE *current_file) ...  
  
void createTZLink (char *lat, char *lon) ...  
  
// called when window is closed  
G_MODULE_EXPORT void on_window_main_destroy(GtkWidget *w) ...  
  
// called when window is closed  
G_MODULE_EXPORT void on_inputWin_destroy(GtkWidget *w) ...  
  
// called when window is closed  
G_MODULE_EXPORT void on_aboutWin_destroy(GtkWidget *w) ...  
  
// pv csv choice  
G_MODULE_EXPORT void on_button1_clicked(GtkButton *b, gpointer *data) { ...
```

Figure 4. Code Guide.

IV. How-To Edit EKC's Library

At the bottom of every CSV file, a template can be found that describes what the library requires to be filled for each component, see Figure 1. Note that **when a CSV section allows the user to fill in more than 1 value, at least 2 entries must be filled**. This is present for the *Discharge Time* and *Effective Capacity* entries in the *Batteries.csv* file. It is a requirement to have at least two values and at most five values. Any values, beyond five will not be read into EKC. If there is only one value found in the array, then the application will not run as expected and likely encounter a segmentation fault.

Model	MODEL NAME								
Voltage (V)	NOMINAL VOLTAGE								
Capacity (Ah)	NOMINAL CAPACITY								
Discharge rate (hr)	SLOWEST DISCHARGE RATE			FASTEST DISCHARGE RATE					
Effective capacity (Ah) or (%/100)	LARGEST EFFECTIVE CAPACITY CORRESPONDING TO ABOVE RATE			SMALLEST EFFECTIVE CAPACITY CORRESPONDING TO ABOVE RATE					
Minimum state of charge (%)	MINIMUM STATE OF CHARGE BEFORE BATTERY CANNOT PROVIDE POWER, DECIMAL FORMAT (i.e. 0.35 NOT 35%)								
Price (\$)	PRICE PER BATTERY								

Figure 5. Template of Inserting New Models in Component CSVs.

When entering values into the *Effective Capacity* row in *Batteries.csv*, entries of absolute effective capacity or capacity relative to the nominal capacity will be accepted. For example, Figure 2 shows the entry of the Victron 12 Volt AGM battery using effective capacities as a percentage of the nominal capacity corresponding to that discharge time. As shown, the battery will have a capacity of 187Ah, or 85% of the nominal capacity, when it is discharged at the 5-hour rate. The user could have entered “.85” or “187”. This would be interpreted by EKC as the same thing.

Model	Victron 12 Volt AGM (using % effective capacities)				
Voltage (V)	12				
Capacity (Ah)	220				
Discharge time (hr)	20	10	5	3	1
Effective capacity (Ah) or (%/100)	1	0.92	0.85	0.78	0.65
Minimum state of charge (%/100)	0.4				
Price (\$)	800				
Reference	KWH Equipment Specifications Google Drive				

Figure 6: Victron 12V Battery Using Relative Capacities

Figure 4 shows the Victron 6 Volt AGM battery using absolute effective capacities. The user could have entered “.92” instead of “206.8” for the 10-hour rate of this battery. Both methods are equally acceptable.

Model	Hypothetical Victron 6 Volt AGM (using Ah effective capacities)				
Voltage (V)	6				
Capacity (Ah)	240				
Discharge time (hr)	20	10	5	3	1
Effective capacity (Ah) or (%/100)	220	206.8	193.6	180.4	167.2
Minimum state of charge (%/100)	0.35				
Price (\$)	550.68				
Reference	KWH Equipment Specifications Google Drive				

Figure 7: Victron 6V Battery Using Absolute Effective Capacities

However, it should be noted that if, for some reason, you were using an extremely small battery (like a AAA battery) and the absolute capacity was less than or equal to 1Ah, you would need to use relative effective capacities (%/100).

V. How-To Use EKC

The application opens two windows when it is first launched; an about and input window. The about window will contain information on how to use the application as well as the top 10 components that can be found in the library. In order to change the component models that are presented in the about window, the user should rearrange the library to have the model be shown in the top 10 slots. Though the application presents 10 models of each component, the application will still be able to use any model that may be found beyond the 10 models that are printed in the about window.

The input window requires that the user enter their inputs in the order that they are presented, the main thing that should be noted is that if the location should be changed, both entry boxes should be reentered. After inputting values into the entry box, it should be noted to press the *Enter* button next to the entry box to have the values stored in EKC's memory. The location input requires a location that is not located anywhere on the in the middle of the ocean. After data has been downloaded, the user can push the *Solar Radiation* button to be view a calculation of the average daily insolation present in that location. A red, yellow, or green circle will appear around the average daily insolation that is qualitative indication of a low, medium, or high-quality solar resource in that area, respectively. Greater than 5kWh/m²/day is considered a high-quality level of insolation and that location is likely suitable for an energy kiosk, while a value less than 4kWh/m²/day is a low-quality level of insolation, and 4-5kWh/m²/day is a medium-quality level of insolation.

The PV entry boxes prompt the user to enter the model they would like to use from the list displayed in the "About" window, as well as the number of PV strings and modules per string that will be used. The user will be prompted for the tilt of the PV array which should be expressed as degrees from horizontal. The azimuth angle should be expressed as degrees from north. For example, "0" would point the array due north, "180" would point the array due south, and "230" would point the array southeast.

EKC Input Window

Location Inputs | **PV Inputs** | Battery Inputs | Inverter Inputs | Charge Controller Inputs | Reliability

PV Model #	Enter	PV Choice: Positive Whole Number Number of Strings: Positive Whole Number Modules per String: Positive Whole Number PV Tilt: 0 < VALUES < 90 Azimuth Angle: 0 < VALUES < 360, North = 0, East = 90, South = 180, West = 270
# of Strings	Enter	
Modules per String	Enter	
PV Tilt	Enter	
Azimuth Angle	Enter	

Figure 8. Input Window for EKC.

The Battery inputs tab requests for the battery model that the user wishes to use, as well as the number of batteries they wish to use for their system. Other inputs would be the inverter and charge controller requesting for which model the user would like to use as well as the amount they wish to use for their system. The user will then input how reliable they wish their system to be.

When the values have been stored, the output window of EKC will appear and present the user with the inputs used to drive the calculation as well as results of their inputs that can be found in the respected output tabs as seen in Figure 9.

Energy Kiosk Creator

File Help

Values Used

		PV Output	Charge States of Battery		Components Breakdown		Capital Cost & Reliability								
		Months	January	February	March	April	May	June	July	August	September	October	November	December	Total Yearly Production
Longitude	28	Average Daily Energy													
Latitude	-15														
PV Array	1		1675.39	1719.47	1733.49	1786.04	1731.43	1642.35	1721.00	1919.17	2027.09	2060.06	1855.91	1681.94	1796.35
Battery	1	Worst Daily Average													
Inverter	1														
Charge Controller	1														
PV Tilt	1														
Azimuth Angle	1														
Reliability	.95														

Figure 9. Output Window of EKC.

The PV output presents the average and worst daily radiation that can be produced by the PV arrays. The Battery States of Charge tab has the maximum and minimum state of charge for the battery, given the user's inputs, as well as a grid that presents the range of states of charge that the battery may be in given the percent of time it spends in that range as well as the number of hours per year. The Components Breakdown tab contains the name of the components chosen by the user as well as the number of models that will be used to design the solar energy kiosk. The Capital Cost and Reliability output tab states the percent of time that the energy kiosk will provide power, given the users reliability, and present a grid that describes what drives the total cost. Two entry box sections allow the user to add tax, in the form of decimal, or balance of system (BoS) cost.

VI. Recommended Updates

As EKC is one of the first student software applications for KWH, ECE 21.3 proposes some recommendations for enhancing the application. These upgrades will range from the functionality of the application to the results that are being printed in the output window.

One recommendation ECE 21.3 can offer is more security for EKC when the application is downloading data from NASA's POWER API. Some of the following features may be added to increase user security when using the application:

1. A two-way authentication, which would require the user to login into a profile to use the application. With this feature, anyone using the computer that EKC is housed on would require user authentication to use or retrieve any information within the application.
2. Include a "time-out series" that will consist of EKC closing out if the application takes too long to download data from NASA POWER API.

As the connection to NASA's POWER API is a single-port access, having the extra-security to allow no other users to interact with the URL will be beneficial. Other recommendations for user

experience, would include allowing EKC to run on a Macintosh Operating System (MacOS). Since, EKC was designed for 64-bit Windows machines, it is suggested to make it more accessible to users by giving the option of using EKC on a MacOS.

For the output window, the *Batteries State of Charge* tab is underpopulated and ECE 21.3 intended to have a histogram that presented the number of hours the batteries were within a specific state of charge. KWH engineers have requested for this in one of ECE 21.3's feedback sessions. The graph would give a clear understanding of what was occurring with the battery, being the depth of discharge and the length of being charged.

Engineers at KWH may benefit from the ability to model different battery types when designing energy kiosk systems using EKC. EKC's models currently only include lead-acid batteries, but these models could be improved if formulas for lithium-ion, lead-carbon, and other battery types that are commonly used in energy kiosks could be included.

ECE 21.3 also recommends having the application size solar energy kiosk for the user specified location, while still allowing the user to input the other entries. This would allow the user to know if their designed system was over, or under, sized, and give KWH an alternative system to the one they intended to build.

One improvement that may have a mild effect on the quality of the PV output calculations would be to take into account albedo and the level of diffuse and direct irradiance. EKC currently only uses global horizontal irradiance to calculate insolation, but accuracy of the calculations may benefit from compensations in PV array output due to varying levels of direct and diffuse irradiance.

VII. References

- [1] "Historical Weather," Wunderground, [Online]. Available: <https://www.wunderground.com/history>. [Accessed 7 July 2021].
- [2] M. Rymes, Solar Position and Intensity (SOLPOS), National Renewable Energy Laboratory, 1998.
- [3] "Numerical Weather Prediction," National Oceanic and Atmospheric Administration (NOAA), [Online]. Available: <https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/numerical-weather-prediction#:~:text=GEFS%20is%20a%20global%2Dcoverage,going%20out%20to%2016%20days..> [Accessed 13 03 2021].
- [4] "Wunderground," TWC Product and Technology, [Online]. Available: <https://www.wunderground.com/history/daily/cd/alpha/FZKA/date/2010-4-26>. [Accessed 2021 02 26].

- [5] "Worldwide Elevation Map Finder Sunset Sunrise Times Lookup," MAPLOGS.COM, 2021. [Online]. Available: https://sunrise.maplogs.com/ituri_democratic_republic_of_the_congo.100346.html. [Accessed 2021 02 26].
- [6] glade-devel-list, "Glade - A User Interface Designer," The Glade Project, 2020. [Online]. Available: <https://glade.gnome.org/>. [Accessed 2020-21].
- [7] G. Team, "GTK," GNOME Foundation, 1997-2021. [Online]. Available: <https://www.gtk.org/>. [Accessed 2020-21].
- [8] N. R. E. L. (NREL), "Solar Position and Intensity," Alliance for Sustainable Energy LLC, 25 March 1998. [Online]. Available: <https://www.nrel.gov/grid/solar-resource/solpos.html>. [Accessed 2020-21].
- [9] J. P. Paul Stackhouse, "POWER | Prediction of Worldwide Energy Resources," NASA Privacy Statement, Disclaimer, and Accessibility Certification, 1 March 2019. [Online]. Available: <https://power.larc.nasa.gov/docs/>. [Accessed 2020].
- [10] H. Tan, "Sizing Your Solar Power System," blog.GoGreenSolar.com, 10 10 2019. [Online]. Available: <http://blog.gogreensolar.com/2019/10/sizing-your-solar-power-system.html>. [Accessed 19 03 2021].

VIII. Appendix

A. All Functions and Variables

x. EKC-Specific functions

Table 2: EKC-Specific Functions

FILE *openReadFile (char file_name[])	Opens a file to be read and returns a pointer to the file. Prints to console the success of the attempt to open the file
void initializeBattery (struct BatteryData *battery);	Sets variables associated with battery calculations to 0.
void setBatteryArray (struct BatteryData *battery);	Calculates the arrayEffectiveCapacity, moduleDischargeCurrent, arrayDischargeCurrent, arrayNominalCapacity, and arrayVoltage values using the parameters for the size of the array and parameters of individual battery modules
void loadValues (char line[], float *array);	Generic function for use in import functions that takes in a line from a csv in the line field, parses out data, and places data in the array or variable. *array can be an array or a single variable.
void loadModelName (char line[], char *modelName);	Generic function for use in import functions that takes in a line from a csv that contains the model name of an item and stores it into the modelName string.
void importTimeZone (FILE *timezone_file, struct posdata *pdata);	Reads the time zone from the Bing Maps API and sets the time zone in the posdata struct accordingly.
void importLoadProfile (FILE *load_profile, int *load);	Imports the load profile from “Load Profile.csv” and stores the data into

	the loadProfile array in the DataValues struct.
void importChargeControllerLibrary (FILE *chargeController_file, struct ChargeControllerData *chargeController, int selection);	Imports the charge controller data from “Charge Controllers.csv” and stores it into the ChargeControllerData struct.
void importInverterLibrary (FILE *inverter_file, struct InverterData *inverter, int selection);	Imports the inverter data from “Inverters.csv” and stores it into the InverterData struct.
void importBatteryLibrary (FILE *battery_file, struct BatteryData *battery, int selection);	Imports the battery data from “Batteries.csv” and stores it into the BatteryData struct.
void importPVLibrary (FILE *pv_file, struct PVData *panel, int selection);	Imports the PV data from “PV Modules.csv” and stores it into the PV Data struct.
void readAllData (struct posdata *pdat, FILE *allData, float dailyKTData[NUM_YEARS][NUM_DAYS], float avgTempData[NUM_YEARS][NUM_DAYS], float maxTempData[NUM_YEARS][NUM_DAYS], float minTempData[NUM_YEARS][NUM_DAYS]);	Reads all the data from NASA POWER and stores it in the dailyKTData, avgTempData, maxTempData, and minTempData arrays.
float readPOWERData(char line[]);	Extracts the data from the line currently being read in the NASA POWER dataset.
int readPOWERYear(char line[]);	Extracts the year from the line currently being read in the NASA POWER dataset.
float calcAvgIrradianceTilt (struct posdata *pdat);	Calculates the average extraterrestrial irradiance on a tilted surface over the course of an hour.
float calcAmbientTemp (int hour, float sunset, float minTemp, float maxTemp, float avgTemp);	Calculates the ambient temperature at the current hour of the day.
float calcCellTemp (float ambientTemp, float noct, float irradiance);	Calculates the temperature of the PV modules at the current hour of the day.

float calcPVPower (float stcPower, float irradiance, float alphaP, float cellTemp);	Calculates the average power that can be produced by the PV array over the course of the hour.
float calcCCControllerOutput (struct DataValues *data, struct ChargeControllerData *cController, float pvOutput);	Calculates the average power that is output by the charge controller. If power produced by PV array is greater than charge controller rating, power is capped.
float calcPeukert (struct BatteryData *battery, float currentOut);	Calculates the Peukert exponent to be used when calculating the state of charge of the battery.
float calcBatteryCharge (struct BatteryData *battery, struct InverterData *inverter, float startSoC, float currentOut);	Calculates the change in state of charge of the battery throughout the hour and the state of charge at the end of the hour.
float calcReliability (struct BatteryData *battery, float SoC[NUM_YEARS][NUM_DAYS][NUM_HOURS]);	Calculates the proportion of time in decimal format that the batteries are above their minimum state of charge and therefore able to provide power to the energy kiosk.
void calcChargeCycles (struct posdata *pdat, struct DataValues *data, float startSoC, float endSoC);	Calculates the number of charge cycles the battery will undergo each year.
void calcMaxSoC (struct posdata *pdat, struct DataValues *data);	Calculates the maximum state of charge for each month, year, and each month of each year and stores it in the DataValues struct.
void calcMinSoC (struct posdata *pdat, struct DataValues *data);	Calculates the minimum state of charge for each month, year, and each month of each year and stores it in the DataValues struct.
void calcSoCIntervals (struct posdata *pdat, struct DataValues *data, float minSoC);	Creates 6 equally-sized state of charge ranges from the battery's minimum state of charge to 100% charged. Ranges are stored in the SoCIntervalRanges array within the DataValues struct.

void calcAvgPVOutput (struct posdata *pdat, struct DataValues *data, float ***pvPower);	Calculates the average PV output for each month, year, and all years. Stores the values in the DataValues struct.
void lowestPVOutput (struct posdata *pdat, struct DataValues *data, float ***pvPower);	For each month, calculates the year that had the lowest PV output. Also calculates the year that had the lowest total PV output. Stores that year in month/yearMinPVOutput in the DataValues struct.
void readModelNames (FILE *current_file, char *modelNames, int string_size, int num_strings);	Used to display the model names in the “About” window of the EKC UI. string_size indicates the maximum length for the name of the model, while num_strings indicates the number of models to be displayed. *modelNames is a pointer to a 2D struct of dimensions num_strings by string_size.
void storeData (struct posdata *pdat, struct DataValues *data, struct BatteryData *battery, float ***pvPower);	Calls calcMaxSoC(), calcMinSoC(), calcSoCIntervals(), calcAvgPVOutput(), and lowestPVOutput().
float insolationQuickLook (struct posdata *pdat, float dailyKTDData[NUM_YEARS][NUM_DAYS]);	Calculates the average daily surface-level insolation over all years using clearness index data download from NASA POWER.
void saveData (struct posdata *pdat, struct DataValues *data, float ***pvPower, float ***hourlyTemp, FILE *savefile);	Saves calculated data and inputs into “Recent Save.csv”.

xi. Main SOLPOS Functions

Table 3: Main SOLPOS Functions

long S_solpos (struct posdata *pdat);	Calculates the apparent extraterrestrial solar position and intensity based on date, time, and location on Earth.
---------------------------------------	---

<code>void S_init (struct posdata *pdat);</code>	Initialized all of the input functions to <code>S_solpos()</code> .
<code>void S_decode (long code, struct posdata *pdat);</code>	Decodes the error codes from the value that is returned by <code>S_solpos()</code> , if there are any.

Table 4: Local solpos.c Functions

<code>static long int validate (struct posdata *pdat);</code>	Calculates any invalid inputs to <code>S_solpos()</code> .
<code>static void dom2doy(struct posdata *pdat);</code>	Converts day-of-month to day-of-year within posdata struct.
<code>static void doy2dom(struct posdata *pdat);</code>	Computes the month/day from the day number within the posdata struct.
<code>static void geometry (struct posdata *pdat);</code>	Does the underlying solar position geometry for a given time and location
<code>static void zen_no_ref (struct posdata *pdat, struct trigdata *tdat);</code>	Calculates the solar zenith angle.
<code>static void ssh(struct posdata *pdat, struct trigdata *tdat);</code>	Calculates the sunset hour angle.
<code>static void sbcf(struct posdata *pdat, struct trigdata *tdat);</code>	Calculates the shadowband factor.
<code>static void tst(struct posdata *pdat);</code>	Calculates true solar time in minutes from midnight.
<code>static void srss(struct posdata *pdat);</code>	Calculates sunrise and sunset times.
<code>static void sazm(struct posdata *pdat, struct trigdata *tdat);</code>	Calculates solar azimuth angle.
<code>static void refrac(struct posdata *pdat);</code>	Calculates the solar position adjusted for atmospheric refraction.
<code>static void amass(struct posdata *pdat);</code>	Calculates relative optical and pressure-corrected airmass.
<code>static void prime(struct posdata *pdat);</code>	Converts between K_T and K_T' . Not used in EKC.

<code>static void etr(struct posdata *pdat);</code>	Calculates extraterrestrial solar irradiance on a horizontal surface.
<code>static void tilt(struct posdata *pdat);</code>	Calculates the extraterrestrial solar irradiance on a tilted surface.
<code>static void localtrig(struct posdata *pdat, struct trigdata *tdat);</code>	Calculates various trigonometric variables used by other functions.

xii. SOLPOS (struct posdata) Variables

Table 5: Struct posdata Variables

int day;	Day of the month. When <i>daynum</i> is used, <i>doy2dom()</i> must be used to calculate <i>day</i> .
int daynum;	Day of the year. When <i>day</i> is used, <i>dom2doy()</i> must be used to calculate <i>daynum</i> .
int function;	Used to determine which functions will be used in calculations. (Recommended: = "L_ALL" OR = "0xFFFF")
int hour;	Hour of day (0-23).
int interval;	Interval of a measurement period in seconds. Forces solpos to use the time and date from the interval midpoint. Date and time are the end of the interval, <i>interval</i> serves as the beginning of the interval, which is the number of seconds preceding the date and time. Measurements are taken as half the number of seconds in <i>interval</i> before the date and time.
int minute;	Minute of hour (0-59).
int month;	Month number (January = 1, February = 2, etc.). Calculated from <i>daynum</i> in <i>doy2dom()</i> .
int second;	Second of minute (0-59).
int year;	4-digit year. EKC uses 1984 through 2005.
float amass;	Relative optical airmass.
float ampress;	Pressure-corrected airmass.

float aspect;	Azimuth of panel surface (N=0, E=90, S=180, W=270).
float azim;	Solar azimuth angle (N=0, E=90, S=180, W=270).
float cosinc;	Cosine of solar incidence angle on panel.
float coszen;	Cosine of refraction-corrected solar zenith angle.
float dayang;	Day angle (daynum * 360° / 365 (days in year)).
float declin;	Solar declination angle (degrees North).
float eclong;	Ecliptic longitude (degrees).
float ecobli;	Obliquity of ecliptic.
float ectime;	Time of ecliptic calculations.
float elevetr;	Solar elevation, no atmospheric correction.
float elevref;	Solar elevation, degrees from horizontal, adjusted for refraction.
float eqntim;	Equation of time (not used in EKC).
float erv;	Earth radius vector
float etr;	Extraterrestrial global horizontal irradiance (W/m ²).
float etrn;	Extraterrestrial direct normal irradiance (W/m ²).
float etrtilt;	Extraterrestrial global irradiance on a tilted surface (W/m ²).
float gmst	Greenwich mean sidereal time, hours.
float hrang;	Hour angle. Hour of sun from solar noon (degrees West).

float julday;	Julian Day of January 1 st , 2000 minus 2,400,000 days (in order to regain single precision).
float latitude;	Latitude (degrees North. South is negative.).
float longitude;	Longitude (degrees East. West is negative.).
float lmst;	Local mean sidereal time (degrees).
float mnanom;	Mean anomaly (degrees).
float mnlong;	Mean longitude (degrees).
float rascen;	Right ascension (degrees).
float press;	Surface pressure (millibars), used for refraction correction and <i>ampress</i> .
float prime;	Factor that normalized K_T , K_N , etc. (not used in EKC).
float sbcf;	Shadow-band correction factor.
float sbwid;	Shadow-band width (cm).
float sbrad;	Shadow-band radius (cm).
float sbsky;	Shadow-band sky factor.
float solcon;	Solar constant (1367 W/m ²).
float ssha;	Sunset hour angle (degrees).
float sretr;	Sunrise time, minutes from midnight, without refraction.
float ssetr;	Sunset time, minutes from midnight, without refraction.
float temp;	Ambient dry-bulb temperature (°C), used for refraction correction.
float tilt;	PV panel tilt from horizontal (0°: on flat ground)
float timezone;	Time zone (East = positive, West = negative).

float tst;	True solar time, minutes from midnight.
float tstfix;	True solar time, local standard time.
float unprime;	Factor that denormalizes K_T , K_N , etc. (not used by EKC).
float utime;	Universal (Greenwich) standard time.
float zenetr;	Solar zenith angle, no atmospheric correction.
float zenref;	Solar zenith angle, degrees from vertical, corrected for refraction.

xiii. PV Panel Variables

*Note: Variables denoted with a * after the semicolon (;) are variables that have been imported from “PV Modules.csv”.*

Table 6: Struct PVData Variables

char model[100];*	Model name of selected PV module
float moduleMaxPower;*	Maximum rated power at standard test conditions for individual module (W).
float arrayMaxPower;	Maximum power at standard test conditions for entire array (W).
float moduleVoc;*	Open-circuit voltage for an individual module (V).
float arrayVoc;	Open-circuit voltage for entire array (V).
float noct;*	Nominal operating cell temperature (°C).
float alphaP;*	Maximum power temperature coefficient (%/K)
int stringSize;	Number of PV modules per string.
int numStrings;	Number of strings in the array.
float price;*	Price of individual solar panel.

xiv. Charge Controller Variables

All variables for *ChargeControllerData* are imported from “Charge Controllers.csv”.

Table 7: Struct ChargeControllerData Variables

char model[100];	Model name of selected charge controller.
float capacity;	Nominal capacity (W).
float maxVoltage;	Maximum voltage (V).
float efficiency;	Rated efficiency of model (decimal).
float price;	Price per module (\$).

xv. Inverter Variables

All variables for *InverterData* are downloaded from “Inverters.csv”.

Table 8: Struct InverterData Variables

char model[100];	Model name of selected inverter.
float capacity;	Nominal capacity (W).
float maxVoltage;	Maximum rated input voltage (V).
float maxEfficiency;	Maximum rated efficiency (decimal).
float zeroLoadPower;	Power consumption under no load (W).
float price;	Price per inverter (\$).

xvi. Battery Variables

*Note: Variables denoted with a * after the semicolon (;) are variables that have been imported from “Batteries.csv”.*

Table 9: Struct BatteryData Variables

char model[100];*	Name of selected battery model.
--------------------------	---------------------------------

float dischargeTime[5];*	Hour-rate of the corresponding charge capacity (hours). Listed in order from highest to lowest (e.g. {20, 10, 5}).
float moduleEffectiveCapacity[5];*	Discharge rated capacity (Ah) for a single battery corresponding to the discharge times. Listed in order from highest to lowest (e.g. {55, 51.5, 48.2}) and imported from “Batteries.csv”.
float moduleDischargeCurrent[5];*	Discharge current (A) for a single battery corresponding to the discharge times. Listed in order from lowest to highest discharge current (e.g. 2.75, 5.15, 9.64).
float arrayEffectiveCapacity[5];	Discharge rated capacity for the entire battery array corresponding to the discharge times. Calculated from the size of the array and the values stored in <i>moduleEffectiveCapacity</i> .
float arrayDischargeCurrent[5];	Discharge current (A) for the entire battery array corresponding to the discharge times. Calculated from the number of strings in the array (<i>numStrings</i>) and <i>moduleEffectiveCurrent</i> .
float moduleVoltage;*	Voltage of a single battery module (V). Imported from “Batteries.csv”.
float arrayVoltage;	Voltage of the entire array (V). Calculated from the number of batteries per string (<i>stringSize</i>) and <i>moduleVoltage</i> .
float moduleNominalCapacity;*	Capacity of an individual battery module (Ah). Imported from “Batteries.csv”.
float arrayNominalCapacity;	Capacity of entire battery array (Ah). Calculated from size of the battery array and <i>moduleNominalCapacity</i> .
int stringSize;	Number of battery modules per string.
int numStrings;	Number of strings in the array.
float minSoC;*	Minimum state of charge that the battery can be discharged to. Imported from “Batteries.csv”.

float price;*	Price of an individual battery module. Imported from “Batteries.csv”.
----------------------	--

xvii. Data Storage Variables

Table 10: Struct DataValues Variables

int loadProfile[NUM_HOURS];	Stores the load profile from “Load Profile.csv”. Each index represents 1 hour of the day.
float insolation[NUM_YEARS][NUM_DAYS][NUM_HOURS];	Calculated hourly all-sky insolation on the tilted solar panel.
float reliability;	Percent of the year that the battery’s state of charge is greater than the minimum state of charge. If state of charge is greater than minimum, battery can presumably provide power to the energy kiosk.
float hourlySoC[NUM_YEARS][NUM_DAYS][NUM_HOURS];	State of charge of the battery during every hour of every year (decimal).
float allMaxSoC;	All-time maximum state of charge.
float allMinSoC;	All-time minimum state of charge.
float yearlyMaxSoC[NUM_YEARS];	Maximum state of charge that occurred every year.
float yearlyMinSoC[NUM_YEARS];	Minimum state of charge that occurred every year.
float monthlyMaxSoC[NUM_YEARS][NUM_MONTHS];	Maximum state of charge that occurred every month of every year.
float monthlyMinSoC[NUM_YEARS][NUM_MONTHS];	Minimum state of charge that occurred every month of every year.
int numChargeCycles;	Number of charge cycles that occur each year.

float SoCIntervalRanges[6];	The lower end of each range for the state of charge histogram. For example, if the minimum state of charge is 40% (or .40), the ranges will be .40-.50, .50-.60, ... , .90-1.0. <i>SoCIntervalRanges</i> will store the lower end of each range with the upper end of the last range assumed to be 1.0. If the minimum state of charge was 40%, the <i>SoCIntervalRanges</i> array would look like: {0.4, 0.5, 0.6, 0.7, 0.8, 0.9}.
float SoCIntervalProportion[6];	The percent of time that the state of charge of the battery spent in each range corresponding to <i>SoCIntervalRanges</i> .
int SoCIntervalTime[6];	The number of hours per year that the state of charge was within each range corresponding to <i>SoCIntervalRanges</i> .
float allAvgPVOutput;	All-time average PV array hourly power (W).
float eachYearAvgPVOutput[NUM_YEARS];	Average PV array hourly power for each year (W).
float eachMonthAvgPVOutput[NUM_YEARS] [NUM_MONTHS];	Average PV array hourly power for each month (W).
float avgMonthPVOutput[NUM_MONTHS];	Average PV array hourly power produced for each month over all years (W).
int yearMinPVOutput;	Year number with worst PV output.
int monthMinPVOutput[NUM_MONTHS];	Stores the year that had the lowest average PV output for that month. (i.e. <i>monthMinPVOutput[4]</i> might return 1993 indicating that the worst April recorded between 1984 and 2005 was in 1993).

xviii. Constants

Table 11: EKC-Specific Constants

PI	3.14159265358979323
-----------	----------------------------

NUM_YEARS	22 Number of years that EKC uses to calculate the feasibility of the energy kiosk.
NUM_MONTHS	13 Used to create arrays that can be indexed from 1-12 for each month of the year. Allows for 0 th index to be ignored.
NUM_DAYS	367 Used to create arrays that can be indexed from 1-366 for each day of the year. Allows for 0 th index to be ignored.
NUM_HOURS	24 Used to create arrays that can be indexed from 0-23 for each hour of the day.
TOTAL_NUM_DAYS	8036 Total number of days used during simulation.
START_YEAR	1984 First year of simulation.
START_DAY	1 First day of simulation (January 1 st).

B. PV Output Calculation Models

c. Insolation

The Solar Position and Intensity (SOLPOS) calculator provided by the NREL is used to calculate extraterrestrial (top of atmosphere) irradiance for every minute of the year $G_{min-etr}$ at the user-specified location [2]. EKC then uses the irradiance at the beginning of each minute to calculate an average for the hour G_{h-etr} .

$$G_{h-etr} = \frac{\sum_{t=0}^{59} G_{min-etr}(t)}{60} \quad (\text{A. 1})$$

Clearness index data, retrieved from NASA's POWER API, is a measure of how much solar radiation passes through the Earth's atmosphere. Clearness index, K_T , is the ratio of irradiance at the Earth's surface G to the irradiance at the top of the Earth's atmosphere G_{etr} .

$$K_T = \frac{G}{G_{etr}} \quad (\text{A. 2})$$

K_T is multiplied by the extraterrestrial irradiance to return the irradiance at the surface of the Earth G_h .

$$G_h = G_{h-etr} \times K_T \quad (\text{A. 3})$$

The hourly extraterrestrial insolation I_h is then calculated using the average irradiance throughout the hour as shown in equation A. 1.

$$I_h = G_h \times 1\text{hr} \quad (\text{A. 4})$$

The extraterrestrial insolation on a tilted surface $I_{h_{tilt}}$ is then adjusted for a solar panel tilted at a user-specified angle from horizontal θ and at a user-specified azimuth angle, as shown in Figure 10. **Error! Reference source not found.:**

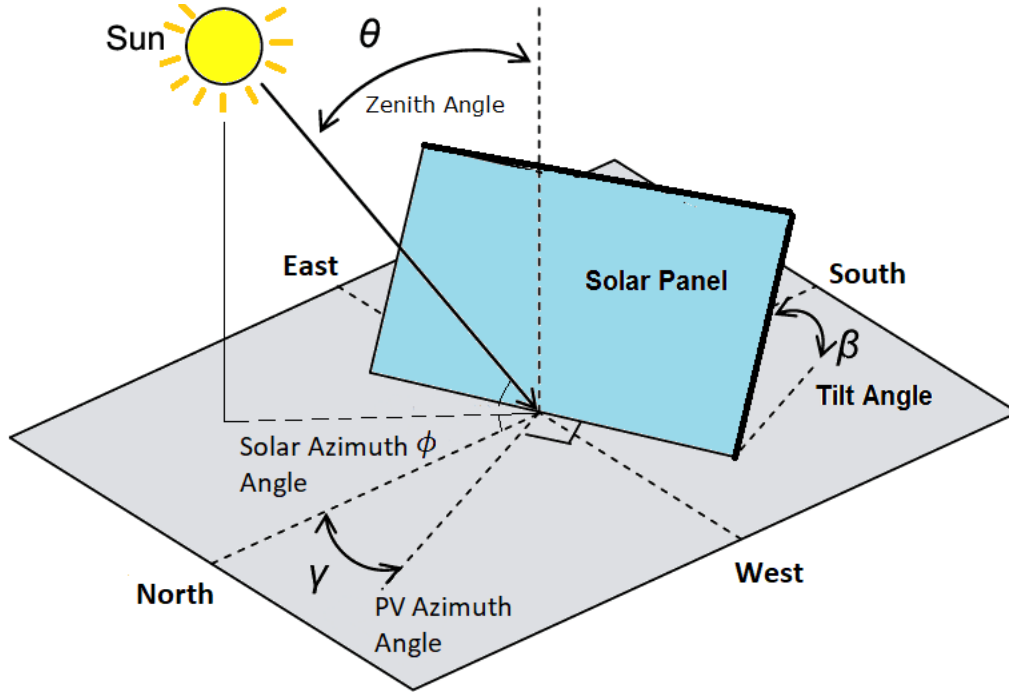


Figure 10. PV Array and Solar Angles.

$$I_{h_{tilt}} = I_h (\cos(\theta) \cos(\phi) + \sin(\theta) \sin(\phi) (\cos(\alpha) \cos(\beta) + \sin(\alpha) \sin(\beta))) \quad (A.5)$$

where:

θ = solar zenith angle (0° is directly overhead) [$^\circ$]

ϕ = tilt of PV array (0° is parallel to ground) [$^\circ$]

α = azimuth angle of sun (0° is North) [$^\circ$]

β = azimuth angle of PV array (0° is North) [$^\circ$]

d. PV Cell Temperature

The power produced by a PV cell is temperature dependent. To calculate the temperature of the PV cells, the ambient temperature must first be calculated. Currently, a simple cosine curve in conjunction with a daily min, max, and mean downloaded from the POWER API is used to approximate the hourly temperature. Accurate models for hourly temperature calculation over the course of a year are not currently available to the public, so this is a simple estimation of temperature [3]. This calculation of ambient temperature is very accurate during solar noon ($\pm 3\%$) but becomes very inaccurate during off peak hours ($\pm 20\%$). Because the PV panels are not producing as much power during off peak hours, the effect on total PV production is minimal. However, if it is determined that EKC requires a more accurate calculation of ambient

temperature, equation A. may be modified. Sunset time is calculated from a SOLPOS function and the maximum daily temperature is set to 3 hours before sunset. The ambient temperature T_a is calculated as shown in equation A..

$$T_a(t) = A \cos[\omega(t - t_{ss} - 3)] + \phi \quad (\text{A. 6})$$

Where ω is the frequency of the function, adjusted to make the period 24 hours long:

$$\omega = \frac{2\pi}{24} \quad (\text{A. 7})$$

A is the amplitude of the function, adjusted as half of the daily temperature range:

$$A = \frac{T_{max} - T_{min}}{2} \quad (\text{A. 8})$$

and t_{ss} is the daily sunset time provided by NASA. The ambient temperature is offset by ϕ , the average of the minimum and maximum temperature.

$$\phi = \frac{T_{max} + T_{min}}{2} \quad (\text{A. 9})$$

The peak temperature is offset by a value of 3 hours because it can be observed that temperature typically peaks about 3 hours before sunset in sub-Saharan Africa [4] [5]. Once the ambient temperature is calculated, the PV cell temperature T_c can be calculated using the irradiance G and the ambient temperature T_a :

$$T_c = T_a + (\text{NOCT} - 20) \frac{G}{800} \quad (\text{A. 10})$$

where NOCT is the nominal operating cell temperature that is specified by the PV cell manufacturer.

e. PV Power

The calculation of the power produced by the PV array is shown in equation A. 1. The calculations for cell temperature T_c and irradiance G are used to calculate the power produced by the PV array:

$$P_{PV}(t) = Y_{PV} \frac{f_{PV}}{100} \left(\frac{G(t)}{G_{STC}} \right) \left[1 + \frac{\alpha_P}{100} (T_c(t) - T_{STC}) \right] \quad (\text{A. 11})$$

where Y_{PV} is the rated power of the PV array at standard test conditions (STC), f_{PV} is the user-specified panel-specific derating factor, α_P is the maximum power temperature coefficient expressed as a percentage, and G_{STC} and T_{STC} are the irradiance (1000W/m²) and temperature (25°C) at STC, respectively.

The average irradiance and temperature values for every hour of the day are used in the PV power equation to calculate an approximation for the power produced in that hour.

C. Battery State of Charge Calculation Models

The state of charge of a battery can be calculated using the discharge current of the battery. The rate of discharge current can be easily calculated using the power output calculated from the

previous PV output calculations. The sum of the PV power P_{PV} minus the load L divided by the voltage of the battery array V_{bat} can be used to calculate x , the current being discharged from the battery.

$$x = \frac{P_{PV} - L}{V_{bat}} \quad (A.12)$$

The charge rate (C-rate) is used to relate the discharge current to the charge capacity c_x at that current. The C-rate at current x is represented as τ and is used to calculate the change in state of charge.

$$\tau = \frac{x}{c_x} \quad (A.13)$$

However, because c_x varies by battery and current, the charge capacity must be calculated for every load that is placed on the battery. Peukert's equation can be used to calculate and approximate charge capacity at a given current. The charge capacity c_x at a discharge current x can be calculated:

$$c_x = c_{x_r} \left(\frac{x_r}{x} \right)^{k-1} \quad (A.14)$$

where k is the Peukert exponent and c_{x_r} is the known charge capacity at a known current x_r . This equation is used to calculate the charge capacity at variable loads placed on the battery by the load and solar production. Charge capacities and currents will be specified in the manufacturer's datasheet, but the Peukert exponent will not be. To calculate the Peukert exponent, we use the rearranged Peukert equation:

$$k = 1 + \frac{\log\left(\frac{c_1}{c_0}\right)}{\log\left(\frac{x_0}{x_1}\right)} \quad (A.15)$$

where c_1 and c_0 are charge capacities at the respective discharge rates x_1 and x_0 .

The application will be calculating the state of charge in intervals of 1 hour, so the C-rate can be applied to the change in the state of charge with a simple equation, where SoC_1 is the state of charge of the battery at the end of the hour and SoC_0 is the state of charge at the beginning of the hour, expressed as a percent of maximum capacity.

$$SoC_1 = SoC_0 - \tau \quad (A.16)$$