

Table of contents

Table of contents	1
User documentation	1
Introduction	1
Accessing the website	1
Introducing calculation parameters	2
Physics Explanation	4
Programming reference	6
Saving the input	6
Calculation	6
Printing the results	6
Converting two-line element sets	7
Fetching upcoming launches from the Launch Library API	7
Further development and improvements	7

User documentation

Introduction

Launch Window Calculator is a static web application for calculating the optimal launch time for spacecraft going to non-equatorial orbits. The calculations are performed by a JavaScript program executed on the user's machine and the user-interface is created using HTML and the mini.css framework. This allows the application to be served over the internet and to be used on any device with a modern web browser.

Accessing the website


The website can be accessed by navigating to the <https://launchwindow.barnazagoni.com>. It was tested with Google Chrome, Mozilla Firefox and Microsoft Edge on Windows 10 and

Google Chrome on Android 6.0, but it should run without any problem on any modern web browser that supports JavaScript.

Introducing calculation parameters

The url opens the website's homepage, which contains a single form for introducing the data for the calculation. The form consists of three sections. One for choosing the launch site, one for specifying the details of the target orbit and one for changing calculation settings.

The launch site location can be specified either by clicking on one of the launch site names, which autofills the information or by manually introducing the coordinates of the launch site. As the number of operational launch sites barely reaches a dozen, in most cases the use of the autofill buttons is the easiest option.



The screenshot shows a form titled "Launch Site". It contains two input fields: "Latitude:" with the value "28.607958" and "Longitude:" with the value "-80.604345". Below these fields are several buttons representing launch sites: "KSC/CCAFS", "Wallops", "Vandenberg", "Boca Chica", "Korou", "Baikonur", and "Jiquan".

Users can introduce target orbits by specifying their orbital parameters, such as the semi-major axis, eccentricity or the inclination of the orbit. All fields have to be filled in order to perform a successful calculation. There is an unlimited number of potential spacecraft orbits, so providing autofill options for most of them wasn't possible, but two autofill buttons were added for the most popular target orbits: the International Space Station and the Chinese Tiangong 2 space station.

Target Orbit

Semi-major axis (km):

Eccentricity:

Angular frequency (rad/s):

Inclination:

Raan (rad):

Epoch (unix timestamp):

A third autofill button is also present for autofilling from TLEs or Two-line element sets. A Two line element set is a data format that contains the parameters that define a spacecraft's path and its orbit. TLEs aren't only easy to process with code, but are also very easy to obtain. There are numerous websites dedicated to sharing TLEs for spacecraft in orbit, in fact the US government publishes a TLE every day for each non-military object orbiting Earth. These data sets are available on websites like Celestrak (<https://celestrak.com/>). In order to fill the orbital parameters from a TLE the user has to click on the *Add by TLE* button, which opens a new input box under the autofill buttons. The TLE can be simply copied and pasted from any source. After clicking the *Fill from TLE* button under the input box, the script will fill the form with the target orbit parameters after performing some calculations.

ISS (ZARYA)

```
1 25544U 98067A 18021.93083348 .00001740 00000-0
33449-4 0 9999
2 25544 51.6427 27.8262 0003583 34.6985 55.6917
15.54193959 95770
```

The last section of the form is for adjusting the details of the calculation, such as how many days to calculate launch windows for and from what time. This part of the form is already filled when the webpage is loaded. The default value for the start time is the unix timestamp of when the page was loaded (aka the current time), while 10 days is the default value for the number of days to calculate for. There are two checkboxes using which the user can define the types of launch windows to calculate.

Calculation

Start Time (unix timestamp):

Number of days to calculate:

Ascending Node Descending Node

After pressing the *Calculate* button, the results will appear under the form pretty much instantly (during testing 100 days' worth of launch windows were calculated under 5 milliseconds). The results are presented in a table, with the launch times specified in Universal Coordinated Time on the left and the corresponding node on the right.

Launch Time	Node
Tue, 23 Jan 2018 00:25:43 GMT	Ascending Node
Tue, 23 Jan 2018 08:51:23 GMT	Descending Node

Physics Explanation

In order to make this topic as easy to understand as possible only launch windows to the ISS are explained below, but the physics isn't much different for other non-equatorial launches.

Most people think that the best time to do a launch is when the ISS is roughly above the launch site. That is not really true. The position of the space station doesn't matter. It can be over the Atlantic Ocean, Europe or even at the opposite end of the planet (like it was at the recent launch of a Russian Progress spacecraft). What matters is the launch site's position relative to the ISS's orbit. Theoretically the ideal time to launch is when the launch site gets exactly under the orbit, because that is when there is no need for fuel-expensive on orbit maneuvers to match the ISS's orbit. That happens twice every sidereal day (23 hours 56 minutes and 4.09 seconds).

Of course with Earth things aren't so simple. The Earth isn't a perfect sphere, it's a geoid, which means that its gravity isn't uniform either. It's weaker at the poles and stronger at the equator, which causes spacecraft orbits to act in a weird way. They aren't stationary in a fixed reference frame. They slowly start to rotate around the Earth's polar axis. This is called nodal precession (as the ascending node - the point at which the spacecraft crosses the equator going south to north - is precessing relative to the reference frame).

This also has an effect on launch windows. In the ISS' case nodal precession causes the launch window to move ahead an extra 19 minutes (+ the shift caused by sidereal days).

While there are launch opportunities to the ISS every day, there are certain times of the year when launches are not practical. As the orbit precesses around the Earth and the Earth moves around the Sun, the angle between the orbital plane and the light coming from the Sun constantly changes. This is called the Beta angle. With a low Beta angle the ISS spends around half of its time behind the Earth's shadow and half of its time exposed to the Sun, but as the Beta angle increases so does the time spent in sunlight. Sunlight has a lot of energy and makes things heat up pretty quickly. Now in the vacuum of space you cannot just dissipate heat away as you do on Earth. You can only get rid of as much heat as the radiators are capable of radiating away.

During high beta angles the space station goes into a low-energy/low-heat mode. This means having to shut down many systems in an effort to produce less heat. During these periods there isn't enough wiggle room to conduct docking and berthing operations without overheating the station.

This calculator is only capable of calculating the launch times with an accuracy of 3 to 5 minutes. This is because rockets' acceleration is not infinite, so they are launched a couple of minutes before the theoretical launch windows.

Programming reference

Saving the input

When the user presses the *Calculate* button the *saveInput()* function is triggered. This function saves all the parameters introduced in the form to javascript variables. This not only makes the code cleaner and easier to understand, but also reduces the times the script will access the DOM during the calculation. This is essential for reducing calculation times. After getting the parameters from the form, the function triggers the *calculate()* function.

Calculation

The *calculate()* function contains a while loop, that initiates the calculating functions *asc()* and *desc()*. The loop runs until both *t1* and *t2* (the results of the two calculation functions) will be larger than the time until which launch times are to be calculated.

The *asc()* and *desc()* functions are almost entirely identical. Their purpose is to calculate the ascending and descending node launch windows respectively and then call the *printTable()* function. The only differences are in the equations used for the calculations and the strings passed to the *printTable()* function.

Printing the results

The *printTable()* function is responsible for updating the div containing the results of the calculation. The function takes a string as a parameter, which is then appended to the *thediv* string, which contains the HTML markup of the results section.

The function counts the number of lines appended since the last time the DOM was updated and only updates it again when it reaches 100. This way poor performance caused by the repeated access of the DOM can be avoided and there isn't any noticeable delay for users either.

Converting two-line element sets

Two-line element sets were specifically designed to be interpreted by computer programs. Each orbital parameter can be found at the same location in every TLE, which makes processing them very easy. In the case of most parameters simple unit conversions are sufficient to make them usable with the calculation functions.

Fetching upcoming launches from the Launch Library API

Launch Library is a public database and API of every upcoming orbital launch, maintained by members of the online space community. It contains many types of information about upcoming launches, ranging from operating agency to rocket type.

Launch Library also has launch pad coordinates, which can be used to create autofill buttons for upcoming launches.

When the website is loaded the *loadUpcoming()* function populates the “Upcoming Launches” field on the right side of the form. The function fetches the launches from the JSON data delivered by the API and also saves the data in a local object.

When the user clicks on one of the autofill buttons the *completeUpcoming()* function is called, with the index number of the launch passed as the *j* variable. Using this number the function gets the launch pad coordinates from the saved object and fills in the respective fields of the form.

Further development and improvements

There is certainly room for improvement with this project. While the code does its job and performance is also reasonable, there are still many places where the code can be simplified and made more efficient. There are also many bad, by most programmers understandably discouraged practices, in use, such as the excessive use of global variables. With over forty global variables I am certainly guilty of not considering where a variable will be used before declaring it.

This project has a lot of potential for further development in terms of its feature set as well. By calculating Beta-angles the website could provide further insight into the factors influencing launch windows for rocket enthusiasts.