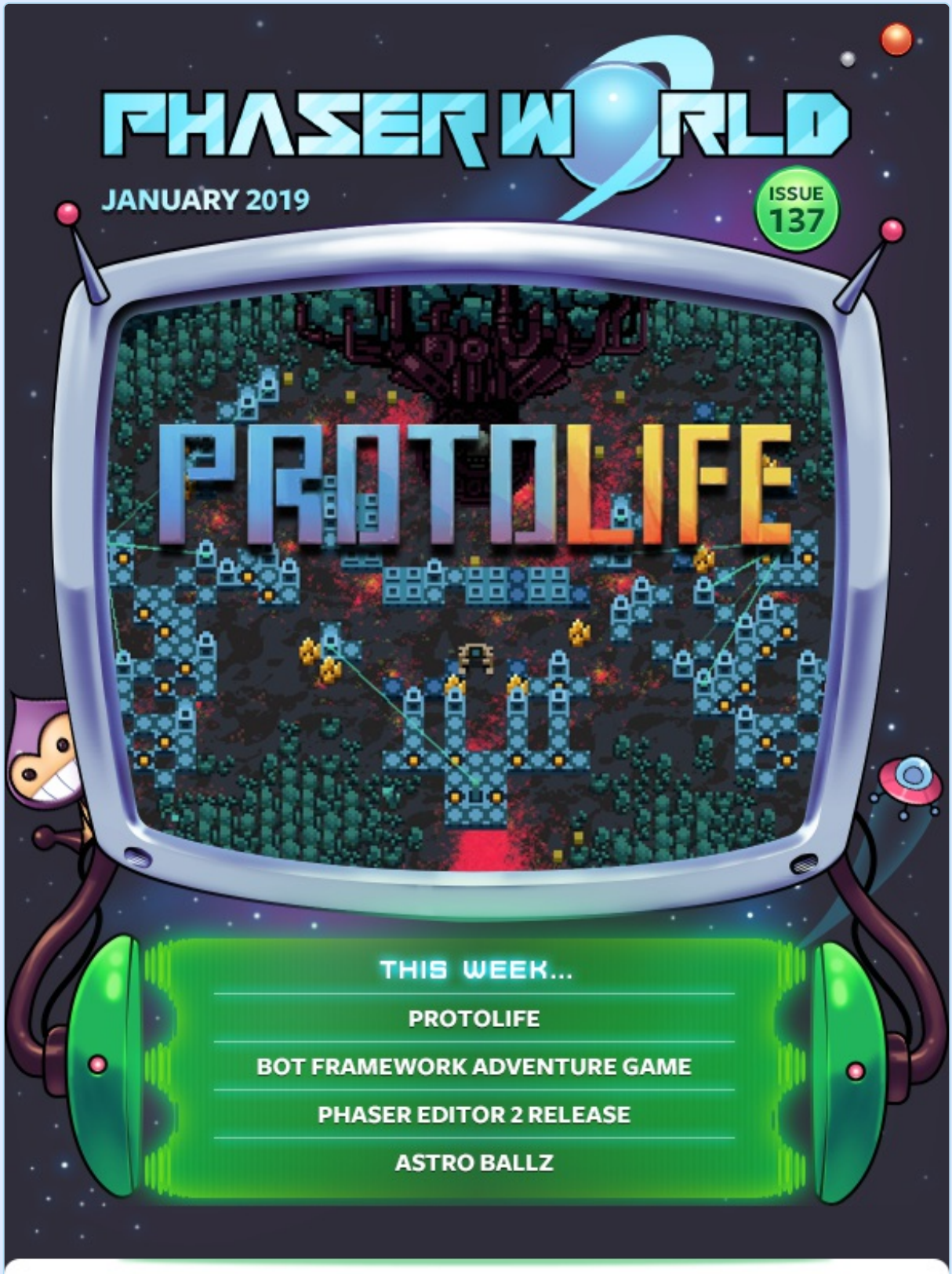
Web Version Unsubscribe



Welcome to Issue 137 of Phaser World

Because Phaser was created from web technology and built for use on the web, it's always interesting to see Phaser games pop-up in different environments. This week our cover game is Protolife. It started out as a Ludum Dare entry and ended up as a coveted commercial release on Steam, where it has a growing audience who thrive on the tiny tower defense mechanisms.

As usual, we've also a stack of new tutorials, a new book published by APress, the v2 release of Phaser Editor and lots of details about the reworked event system in the Dev Log. So, dig in! and be sure to read this newsletter on the web so you don't miss anything.

Got a game or tutorial you'd like featured? Simply reply to this email, or message me on Slack, Discord or Twitter. Until the next issue, keep on coding!

Phaser Sticker Packs + badges + freebies

The Phaser sticker packs continue to prove popular :) and each sale helps contribute towards the development of Phaser itself, so it's win-win! All orders placed this month will get a free **Phaser pin-badge** and a new **pixel-art sticker** included as well.

Click here to order a sticker pack.

The Latest Games

Game of the Week

Protolife

A unique tower-based strategy game in a struggle against the hungry proto-life organism. Protolife went from a game jam entry to commercial Steam release.

Staff Pick

Astro Ballz

It's breakout, but not as you know it. With powerful gravity and pinball physics, exploding planets, vibrant 3D graphics and 160 levels to explore.

Battle Bingo

Quick tap the bingo balls as they move across your card in this visually lovely and frantic Facebook Instant Game.

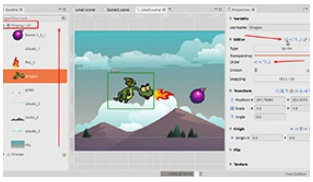
Star Raid

Run, jump and climb your way around the warehouse trying to escape in this teeny tiny 64 x 64 pixel sci-fi platformer.

Jewel Aquarium

It's match 3 under the sea! Swipe the fish to form color chains and clear the levels with as high a score as possible.

What's New?

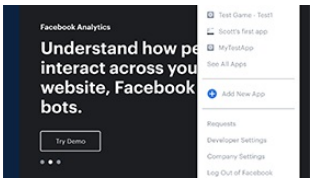


Phaser Editor 2 Preview Release

A brand new release of the premiere Phaser IDE is now out, including hundreds of new features and built-in tools.

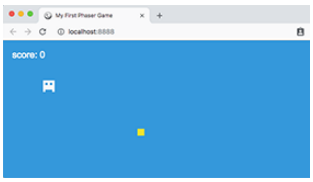
Bot Framework Adventure Games

Creating an 8-bit adventure game using Phaser that you can control via the Microsoft Bot Framework.



Facebook Instant Games Tutorial

A comprehensive guide to using the Facebook Instant Games Plugin in Phaser 3.

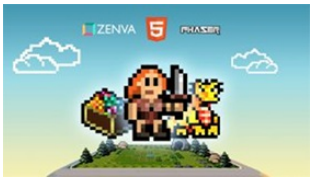


Learn to make HTML5 games with Phaser 3

A brand new tutorial covering the all important basics from a Phaser 2 veteran.

Let's Build a Multiplayer Phaser Game

With TypeScript, Socket.IO, and Phaser in this new book from Apress.



Phaser 3 Game Development Course

A complete Phaser 3 and JavaScript Game Development package. 9 courses,

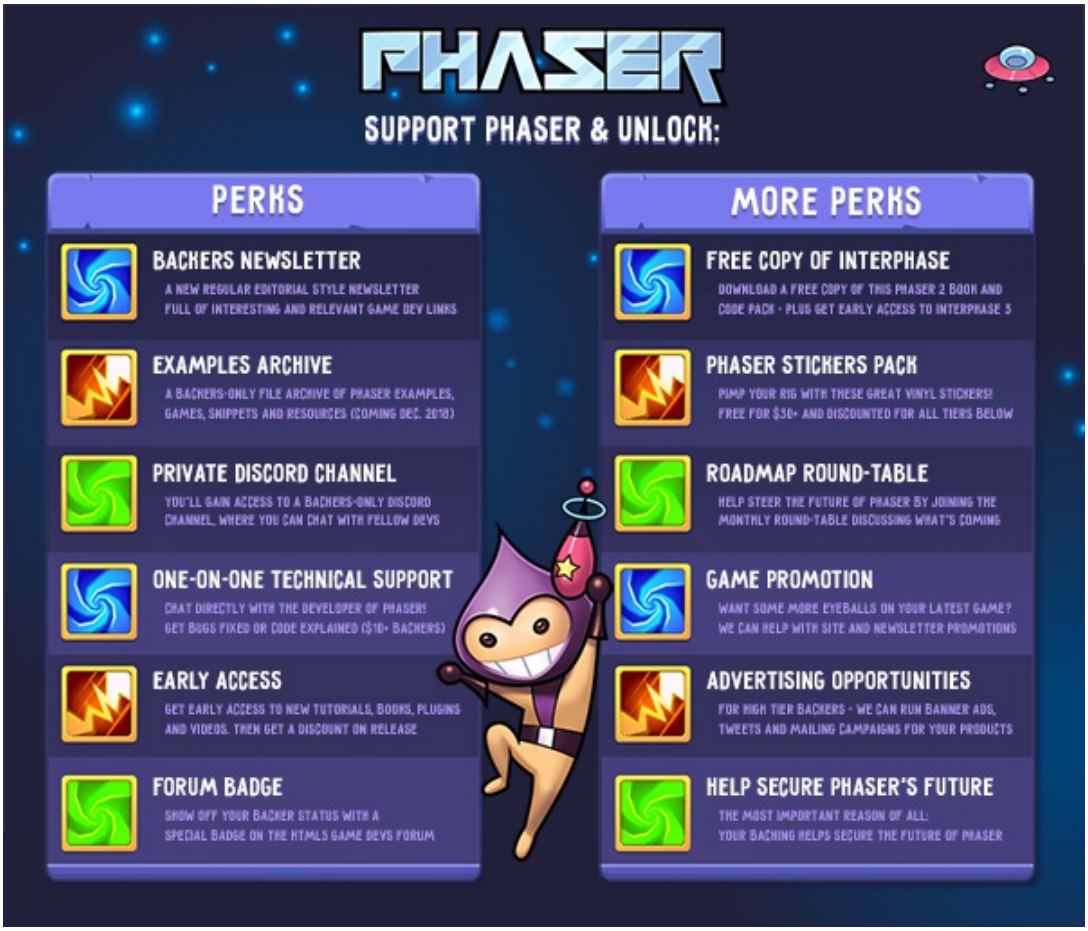
119 lessons and over 15 hours of video content. Learn to code and create a huge portfolio of cross platform games.

Help support Phaser

Because Phaser is an open source project, we cannot charge for it in the same way as traditional retail software. What's more, we don't ever want to. After all, it's built on, and was born from, open web standards. The core framework will always be free, even if you use it commercially.

You may not realize it, but because of this, we rely 100% on community backing to fund development.

Your support helps secure a constant cycle of updates, fixes, new features and planning for the future. There are other benefits to backing Phaser, too:

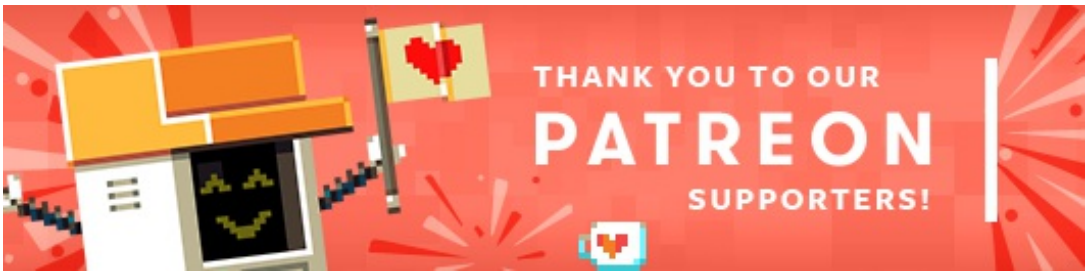


Click to see the full list of backer perks

I use Patreon to manage the backing and **you can support Phaser from $1 per month.** The amount you pledge is entirely up to you and can be changed as often

as you like.

Please help support Phaser on Patreon



Thank you to these awesome Phaser Patrons who joined us recently:

Chris K
Guzman Monne
Jacob
James Van Roose
Jason Bentley
Jon Holloway
Lorenz Glißmann
RoZs Clive
Steve Perreault

Also, thank you to **madclaws** for increasing their pledge and to **Volcanic Giraffe** for their donation!

Dev Log #137

Welcome to another Dev Log! This week has been quite an ardous one. Not in terms of personal life, that has been fine, but purely in terms of the Phaser dev work I've been doing. It started out with me fixing some issues in the new Scale Manager. Thanks to some interactions on Discord I found a couple of nice bugs and also performed a load of testing on mobile devices. One thing I kept hearing was that you wanted an example showing how to use say a 'full size'

background, but then a 'responsive' game centered within that.

Thankfully, this was always a part of the plan and will be easy to add in. It will work because each Camera Manager can maintain its own Size component, on which all scaling is based, the parent being the Scale Manager itself. This means you could have one Camera Manager using a `FIT` aspect mode, while another is using `ENVELOP` to fill up the screen.

This is easier to explain with examples and documentation, which is what I'll spend the coming week doing. My plan, as it stands today, is to get Full Screen support into the Scale Manager, fix a few more bugs and then it'll be done. I will then return to updating to the new Spine 3.7 runtimes, which I'm hoping will be a really simple task due to the way I built the Spine plugin. With that done, 3.16 can be released.

One issue I do have is that I found a nice bug in the Camera Manager that causes it to create incorrect transform matrices if you zoom or rotate the Camera *and* have a game resolution above 1. Based on the comments in the source code it looks like this is a really old issue and has likely been there since the beginning. Probably a symptom of when Felipe built it in a hurry, and we didn't test that edge case, leaving the issue to remain in there. Looking at it more closely now I can see that the root transform matrix is wrong, under certain conditions. Unfortunately, lots of systems were then built on-top of this and all assume those sometimes incorrect values, are correct.

I've a couple of options here: Spend an unknown amount of time trying to resolve it, or disable high resolution support *for now*, enabling it at a later date. The problem is that this incorrect calculation is down at the root level, and I can see that code in Camera culling, input hit testing, the render pipelines, and more, all rely on it. This means it's not a simple case of just fixing the underlying problem in the Camera Manager. All the other parts will need updating as well, which is a massive amount of testing and edge cases to consider. This takes time, a lot of it.

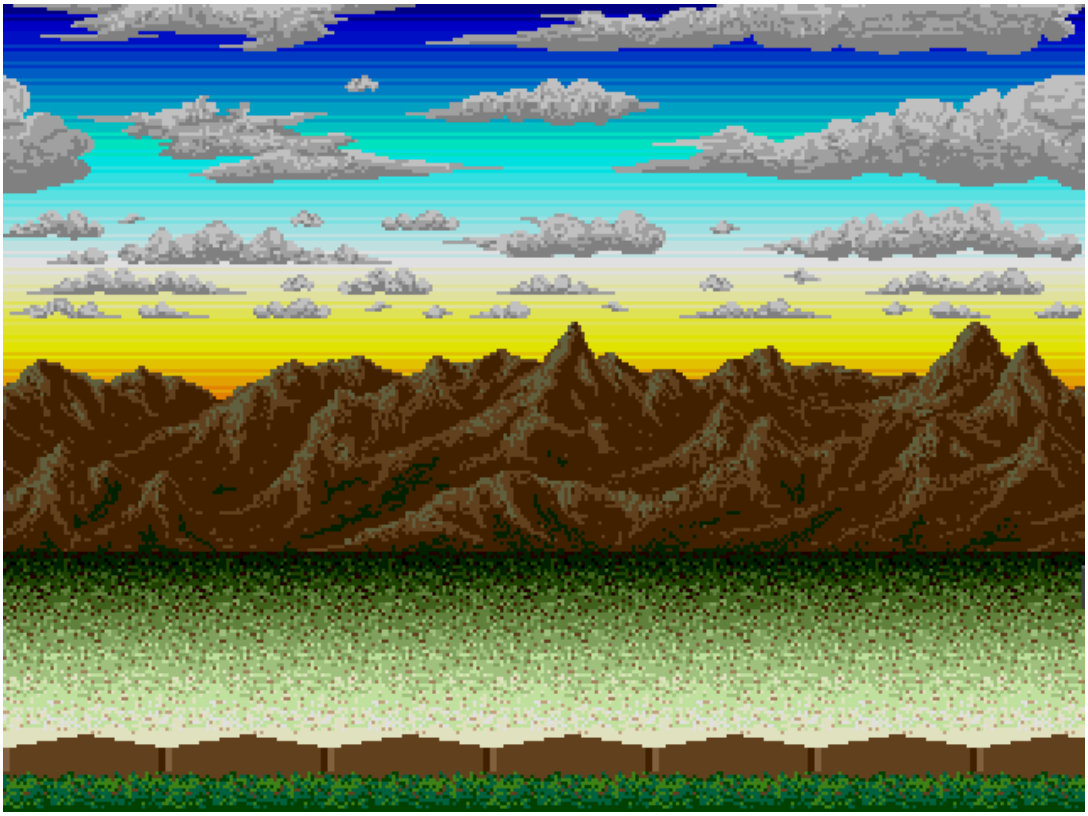
Right now, I honestly believe that the best course of action is to disable high resolution support in the Scale Manager *just for the 3.16 release*. It's not like I'm taking a feature away from you, because it doesn't work properly in any version at all. It does mean that something I really wanted in there for release won't be present. It's frustrating, because I got it working and looking nice! But the moment you want to do interact with something, or modify a camera, it falls over. I need to focus on the positives, though. There are *so many* other really important fixes and updates in 3.16, completely unrelated to scaling, that I don't want to put off the release any longer at all. The end of January was always my personal

deadline and I feel comfortable making this, as long as I don't stop to do this.

Ironically, fixing this bug has the potential to be a change that ripples through-out the entire API. I think the best thing to do would be to keep it all 'as is' for now and comment-out the high resolution config options and methods in the Scale Manager. That way, it will all still work properly and you won't accidentally trigger the issues in the Camera Manager. With 3.16 out, I can focus on resolving it for good. It's not ideal, but I'm really not sure we've a choice right now.

It's not all doom and gloom, though, as the main bulk of the week was spent doing a task that I had been putting off for a while due to its monotony, but that really needed doing. And it was all to do with Events.

Event Horizon



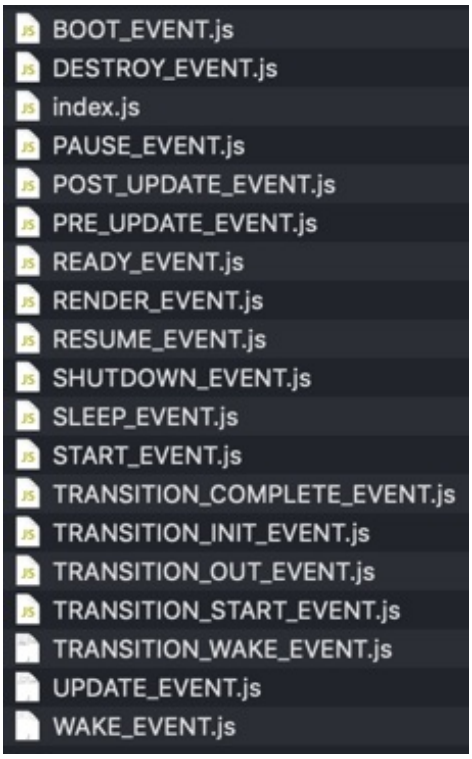
A common complaint with those new to Phaser are "where do I get a list of all the events?" - the answer, until now, has been "there isn't one". While all the important events had been documented, the naming of them was inconsistent, which made them quite hard to find in the API Documentation. What's more, it wasn't always obvious which methods fired which events. They also all used strings, as events typically do, which meant it was easy for you to make a typo in your code and wonder why your event handlers suddenly stopped working.

So, this week, I started sorting them all out. There are hundreds of events through-out the API and every single one of them needed tidying up. The way in which they are now structured is a lot cleaner. There is a proper namespace for events in all areas that use them. For example:

Phaser.Input.Events
Phaser.Sound.Events
Phaser.Animations.Events

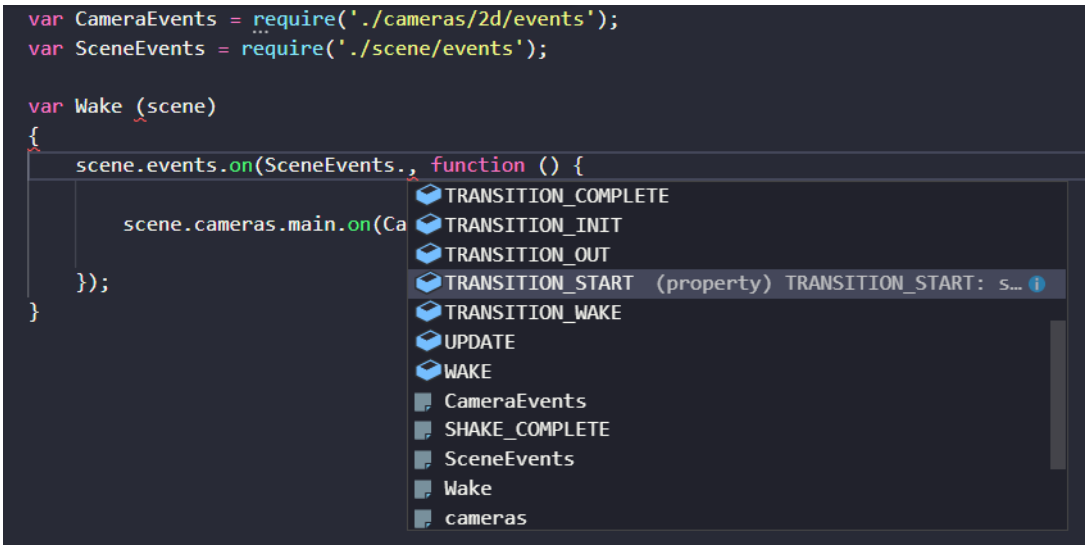
... and so on.

In terms of the source code files, events now live inside an 'events' folder and all follow the same file naming convention. For example, here are the Scene events:



As you can see, the filenames are the event name, followed by `_EVENT`. This makes them easy to distinguish if you've got one open in your IDE.

The namespacing is the most important part. Because they are all now grouped together you can import a specific bundle of events into your code, and any decent IDE will auto-complete them, like so:

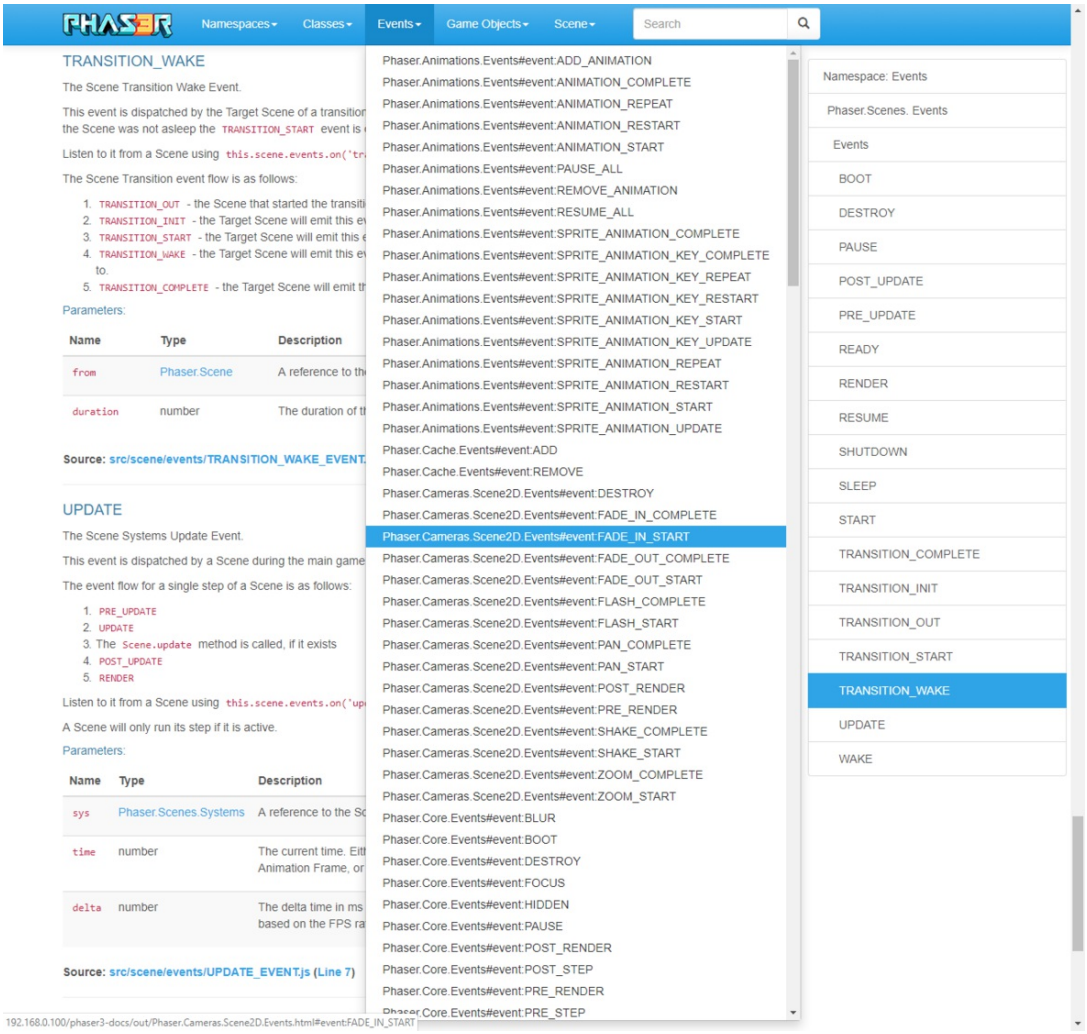


This makes working with them so much easier. If you're using an editor like VS Code you can even select an event and F12 it to jump directly to is definition. This is another important change. I have documented *every single event* in the entire API. This includes all of the parameters that are sent to your event listeners. For example, here is the POINTER_DOWN Input Event:

```
/**
 * The Pointer Down Input Event.
 *
 * This event is dispatched by the Input Plugin belonging to a Scene if a pointer is pressed down anywhere.
 *
 * Listen to this event from within a Scene using: `this.input.on('pointerdown', listener)`.
 *
 * The event hierarchy is as follows:
 *
 * 1. [GAMEOBJECT_POINTER_DOWN]{@linkcode Phaser.Input.Events#event:GAMEOBJECT_POINTER_DOWN}
 * 2. [GAMEOBJECT_DOWN]{@linkcode Phaser.Input.Events#event:GAMEOBJECT_DOWN}
 * 3. [POINTER_DOWN]{@linkcode Phaser.Input.Events#event:POINTER_DOWN} or [POINTER_DOWN_OUTSIDE]{@linkcode
Phaser.Input.Events#event:POINTER_DOWN_OUTSIDE}
 *
 * With the top event being dispatched first and then flowing down the list. Note that higher-up event handlers can
stop
 * the propagation of this event.
 *
 * @event Phaser.Input.Events#POINTER_DOWN
 *
 * @param {Phaser.Input.Pointer} pointer - The Pointer responsible for triggering this event.
 * @param {Phaser.GameObjects.GameObject[]} currentlyOver - An array containing all interactive Game Objects that
the pointer was over when the event was created.
 */
```

Where relevant, I've documented the event flow, or hierarchy, too. So you can instantly see at which stage of the flow this event comes, in case you actually needed one at a different point.

I spent a long time tweaking the formatting of the documentation so that it looked a lot better on export. Here is what the events drop-down list and page now looks like in the API Docs:



Quite an improvement, no?!

The final change, which is equally as important as those above, was that every single method that emits an event is now documented as doing so. This means if you're looking at the source code, you can see the `@fires` tags to know which events the method may dispatch. And if you're browsing the API Docs you'll see them all listed now, too (look at the bottom of the image, the Fires links):

zoomTo(zoom [, duration] [, ease] [, force] [, callback] [, context])

This effect will zoom the Camera to the given scale, over the duration and with the ease specified.

Parameters:

Name	Type	Argument	Default	Description
zoom	number			The target Camera zoom value.
duration	integer	<optional>	1000	The duration of the effect in milliseconds.
ease	string | function	<optional>	'Linear'	The ease to use for the pan. Can be any of the Phaser Easing constants or a custom function.
force	boolean	<optional>	false	Force the pan effect to start immediately, even if already running.
callback	CameraPanCallback	<optional>		This callback will be invoked every frame for the duration of the effect. It is sent four arguments: A reference to the camera, a progress amount between 0 and 1 indicating how complete the effect is, the current camera scroll x coordinate and the current camera scroll y coordinate.
context	any	<optional>		The context in which the callback is invoked. Defaults to the Scene to which the Camera belongs.

Since: 3.11.0
Source: src/cameras/2d/Camera.js (Line 652)

Fires:

- Phaser.Cameras.Scene2D.Events#event:ZOOM_START
- Phaser.Cameras.Scene2D.Events#event:ZOOM_COMPLETE

Clicking an event takes you right to the docs for it.

Internally, the Phaser code itself now uses these new files, too. This means none of the Phaser code uses actual strings anymore but uses the proper Event name instead. Again, it makes it easier to follow and also helps reduce the build size a little, as it's just a reference now, rather than a string. It also helps catch errors with typos.

All in all, there are over 200 events across the API and every single one of them now has its own documentation, parameters list, and namespace entry. Believe me, it was quite an effort to do, but I'm so pleased with the end result! It will make everyone's lives easier.

Although not something I'll do for 3.16, I will also do the same thing again, but this time for Type Defs. These are the parts of the documentation that explain what properties can go into configuration objects. It will make a lot of sense to split them into single namespaced files, just like the events are, so you can find them easily in the documentation and when browsing the source. It should also help TypeScript users a lot!

The new docs will be published with the 3.16 release, but I have already pushed the new jsdoc files live, so if you want to build them yourself from the repo you're more than welcome!

I'd just like to take a moment to thank all of the Phaser backers on Patreon. If it wasn't for your support, I wouldn't have been able to spend so much time on tasks like this. It's only because of the money Patreon brings in that I'm able to dedicate my full working week to Phaser. And when I can do that, things like this become possible. So, thank you all.

Next week I'll be entering the home-straight on 3.16, getting the final parts finished off and ready for release. I honestly can't wait! It's been a long time coming but it's easily the best version of Phaser 3 yet.

Tom duBois is an illustrator who created the elaborate paintings used on North American Konami box art and magazine ads from 1988 to 1994. This is his story.

This tiny Bluetooth chip doesn't need a battery because it harvests energy from the air.

An interactive introduction to fourier transforms.

Phaser Releases

Phaser 3.15.1 released October 16th 2018.
Phaser CE 2.11.1 released October 2nd 2018.

Please help support Phaser development

Have some news you'd like published? Email support@phaser.io or tweet us.

Missed an issue? Check out the Back Issues page.