How I became a core contributor and an expert on WordPress transients

A Developer's Story

WordCamp Edinburgh

vip

I work on the VIP team at Automattic and, no, I'm not here to talk about Gutenberg. VIP are responsible for hosting high-end business and enterprise clients on WordPress, such as **New York Post, TechCrunch and News Corp**. I'm also a WordPress plugin developer and contributor to **Core, WordPress.TV, Translations, Support and Documentation.**

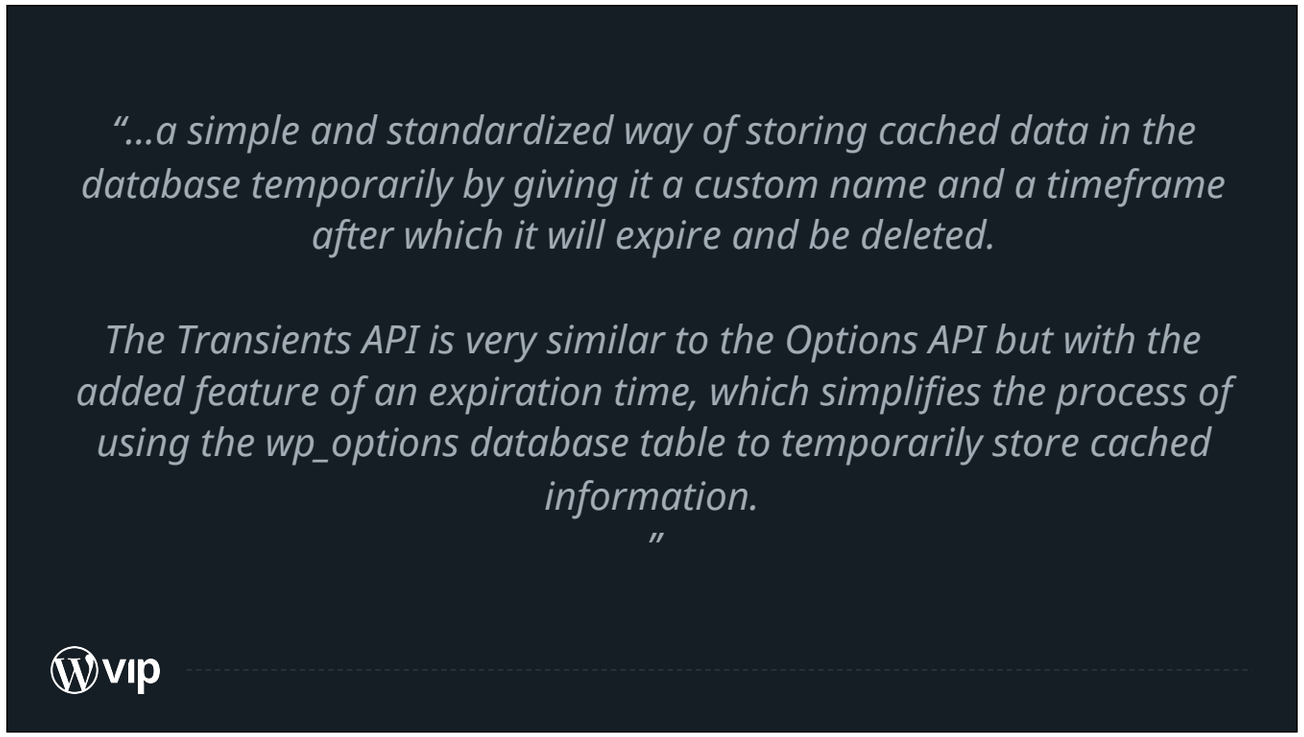
And today, I want to talk about WordPress transients. If you've wandered in here by mistake and are now panicking, I'm having the doors locked so you can't escape.

Now, how can you take a pretty dry subject, such as caching, and make it more interesting. By telling a story. So I'm going to tell you how I became a contributor to WordPress core through working with transients. And, along the way, explain a few things that you possibly didn't know about transients - in particular, **like Salamander Street, we'll explore some of the more seedy parts**. Yes, I am going to keep dropping in local, cultural references.

What are WordPress transients?

So, what are transients? I’m going to open this to the floor. Anyone?

The codex defines them as…



I need to introduce them to punctuation.

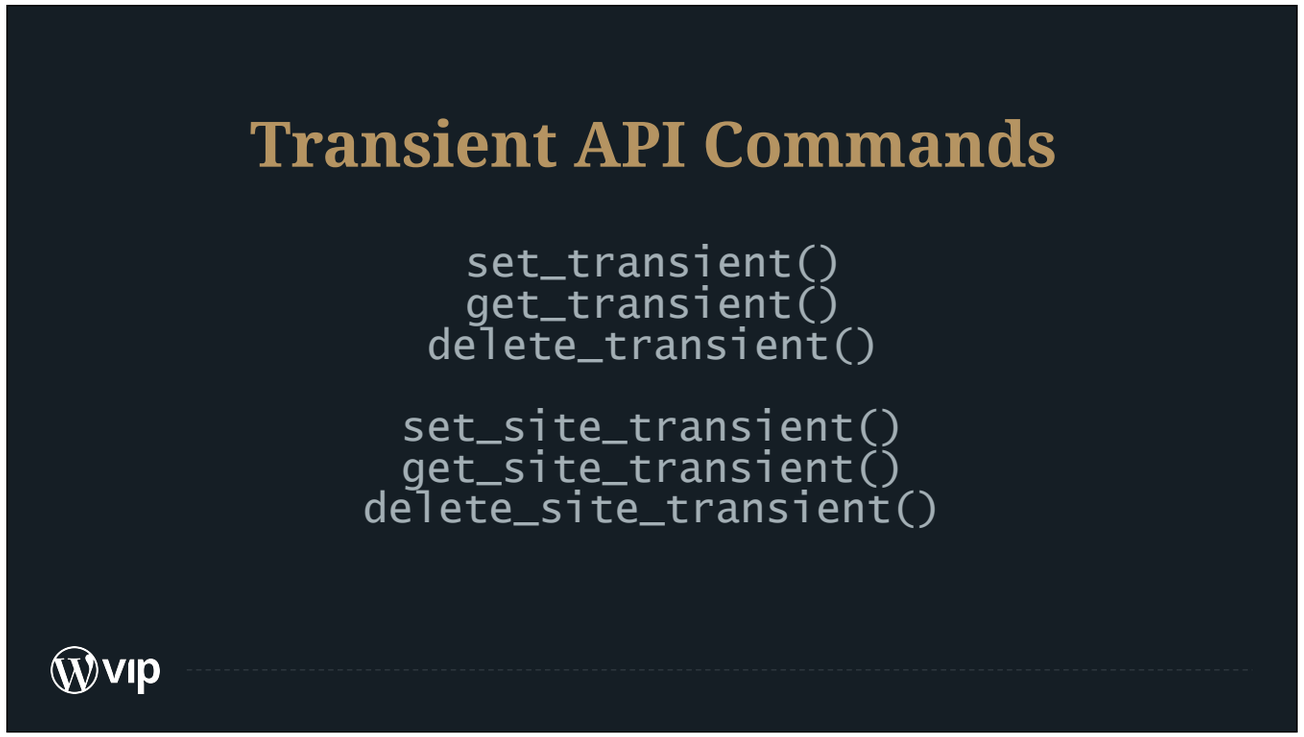
Now, before I get texts, I’m aware that the codex is now deprecated as the official documentation. Unfortunately, the Transients API has not yet been added to the Developer handbook - only the commands. For now, it remains relevant.

So, this is data that is held on the options table and is meant for temporary storage. There are 3 commands available…

Transient API Commands

```
set_transient()
get_transient()
delete_transient()
```

If you’re using a multisite, there are 3 more available to you…



And these are used for managing transients at the network level.

Let’s briefly look at the parameters..

set_transient / set_site_transient

Parameters

```
$transient -transient name (required)
$value - transient value (required)
$expiration - time until transient expires (optional)
```

Return Values

```
False is not set / True if set
```

What's particularly cool is that we **don't need to serialise our transient content before we store it.** In other words, **we aren't limited to storing simple values like strings or numbers but can store entire arrays or objects.**

And what makes these functions particularly different to those used for storing options is that last parameter - an expiry. So you save data as a transient and have it expire after a fixed period of time. This is defined by the number of seconds that need to pass before the expiry should take place.

get_transient / get_site_transient

Parameters

`$transient -transient name (required)`

Return Values

`Value of transient or False is transient does not exist`

By the way, here’s a tip. For those who haven’t yet come across these, there are a number of **time constants** in WordPress which make setting the expiry a lot easier…

So, for example, to set an expiry of a day you can simply use the ‘DAY_IN_SECONDS’ constant.

Now we get to the first ‘gotcha’ with transients. When using **get_transient** or get_site_transient it will **return false** if nothing could be found. However, unlike the other functions, that may give you incorrect information. You see, you may have stored false, or similar, as your transient content. For this reason you should always use the **identity operator** - 3 equals signs. Indeed, it’s recommended that you **don’t store plain boolean values in transients, but instead put them into an array or convert them to integers first.**

How to use transients

As the codex suggests, transients are a great method of caching data, particularly for plugins. Your plugin may generate some output and you'll save it as a transient so that it can be recalled later rather than regenerating the same output each time it runs.

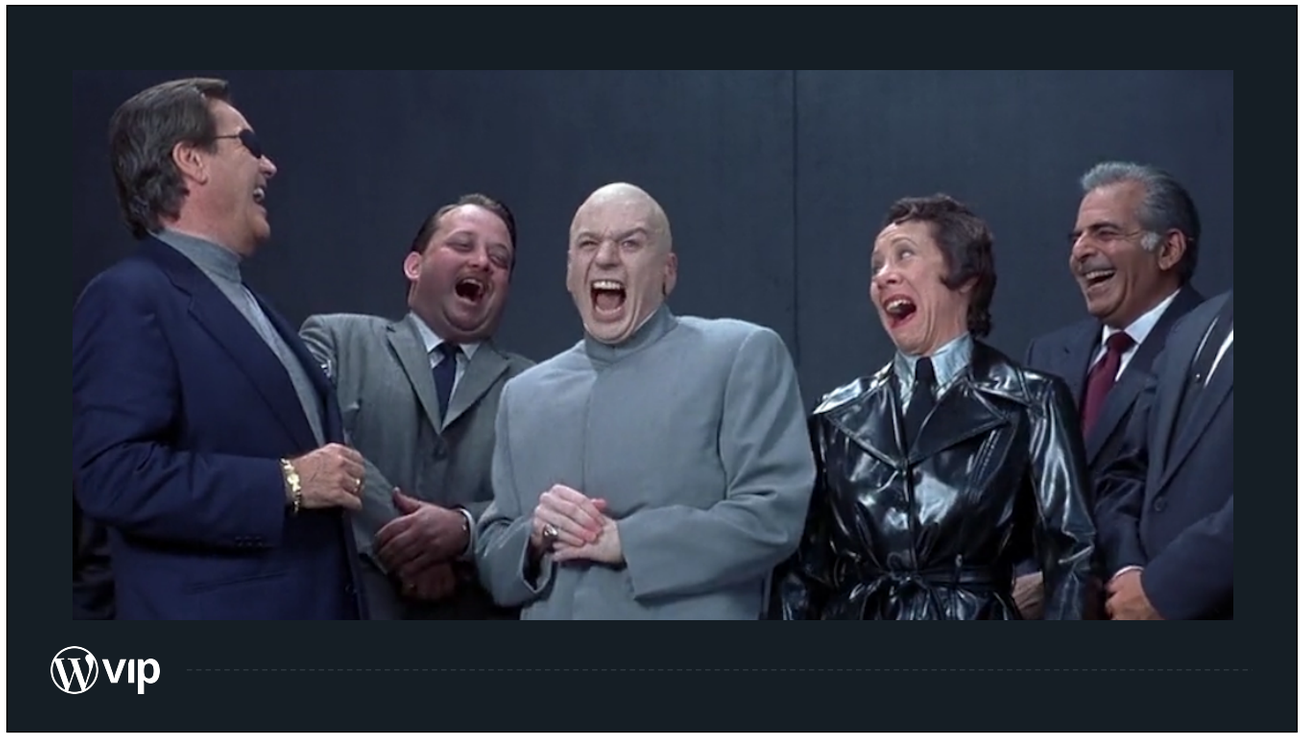
Let's take a couple of examples of how you may use it…

1. Expiring transients

In this you'll save some some data as a transient and set a specific expiry on that data. So, it may be some information you've pulled from an API and to save constantly polling it, you may save the data for, say, an hour. Maybe you wrote a Hello Dolly style plugin, because everyone likes those, that adds a quote to the dashboard. But you only want to refresh it once a day - you'd then get the transient to expire after 24 hours before generating a new fun, quote.

If you don’t specify an expiry on a transient, it will default to never expiring. “Woah, hold on”, I hear you say, wasn’t the entire MO of transients that they expired? Otherwise, they’re just options. As the earlier codex quote mentioned, transients are temporary data - if they’re deleted nothing should break, which means that these still differ from standard options.

Now, you might want to use this method if you want to control the expiry yourself. And for this one, I can give you an example from one of my own plugins. It’s called World Domination…

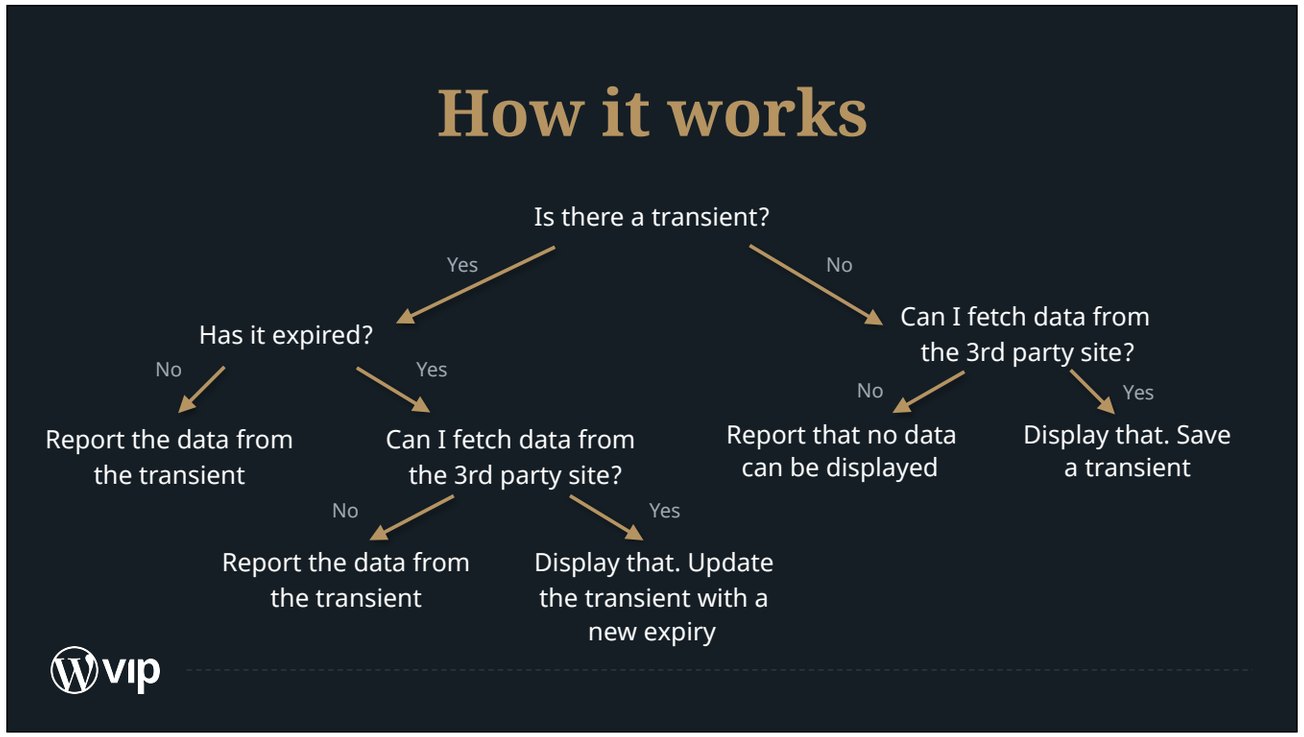


…and adds to your dashboard details of how much CMS market share WordPress currently has. You all now want it, I can tell.

The plugin works by extracting the information from a third party site and it refreshes this daily. Why don't I just get the transient to expire? Because if it expires, the data is gone. If I can't then get the latest data from the third party site then I have nothing to report. By controlling the expiry myself, if the data isn't available, I can try again later and, for now, continue to report the cached data from the previous day. I only delete the transients once I've got something new to update it with.

Let me explain how it works in more detail…

So for World Domination, the transient data is an array of the content and of the expiry timestamp. When saving the transient the expiry parameter is set to zero. Here's how the logic works…



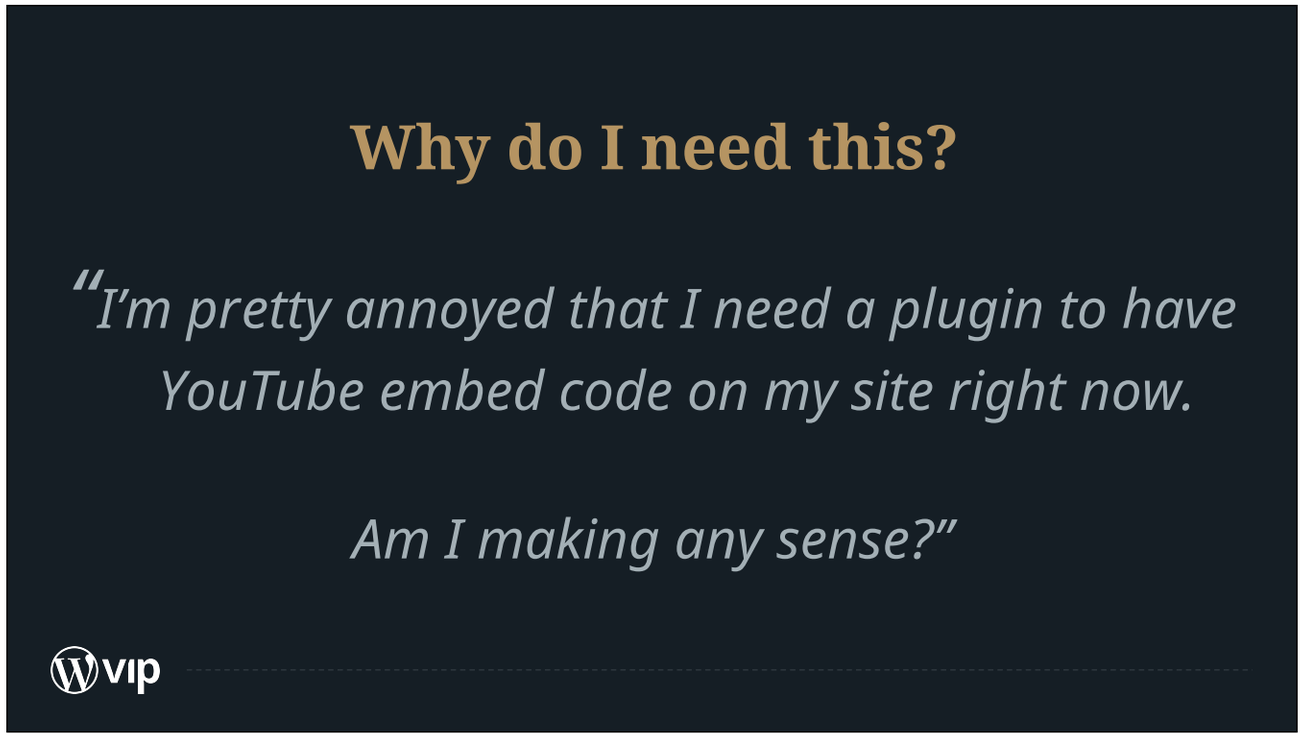
The other advantage of this is if the data has expired and we can't fetch the third party data, then it will keep trying each time, as the transient remains expired.

Another thing to consider is how you handle the naming of your transient. It can be a fixed or generated key. In the example of World Domination, there's only one transient required, and it's called, not surprisingly, 'world domination'. But, if you were caching, say, video embeds, you may want to create a transient for each and, so, use the video name to uniquely identify them. The downside to both these methods is that if anybody changes any settings associated with these, unless you go away and delete the transient pre-expiry, you're going to continue using the existing, cached code until its time is up. In the case of the video example, if you change a global setting for the plugin, which affects all the videos, how do you then identify and delete each transient?

There is a third way! My most popular plugin embeds YouTube videos. Yes, yes, I know we have oEmbed for that, but this does everything, including allowing you to generate your own playlists in the admin and having multiple profiles, so you can have videos default to particular styles on different parts of your site. This means though that generation of a video has some overhead in the time it takes to create the code. So I use transients. And that speeds up the embedding process by a factor of 5.

And how I use them is the clever bit. Well, I thought it was clever. I hash the settings for each video and use that hashed key as the name of the transient. That way, as long as the settings remain the same, the same output is required and, hence, I can use the same transient. If you then change a setting, a new key is created and so a new transient is needed.

Which is great, because the old ones will expire. Won't they? And then I got a worrying question on the plugins' support forum…



Well okay, no, this isn’t the one I meant but I did get this as a support question.

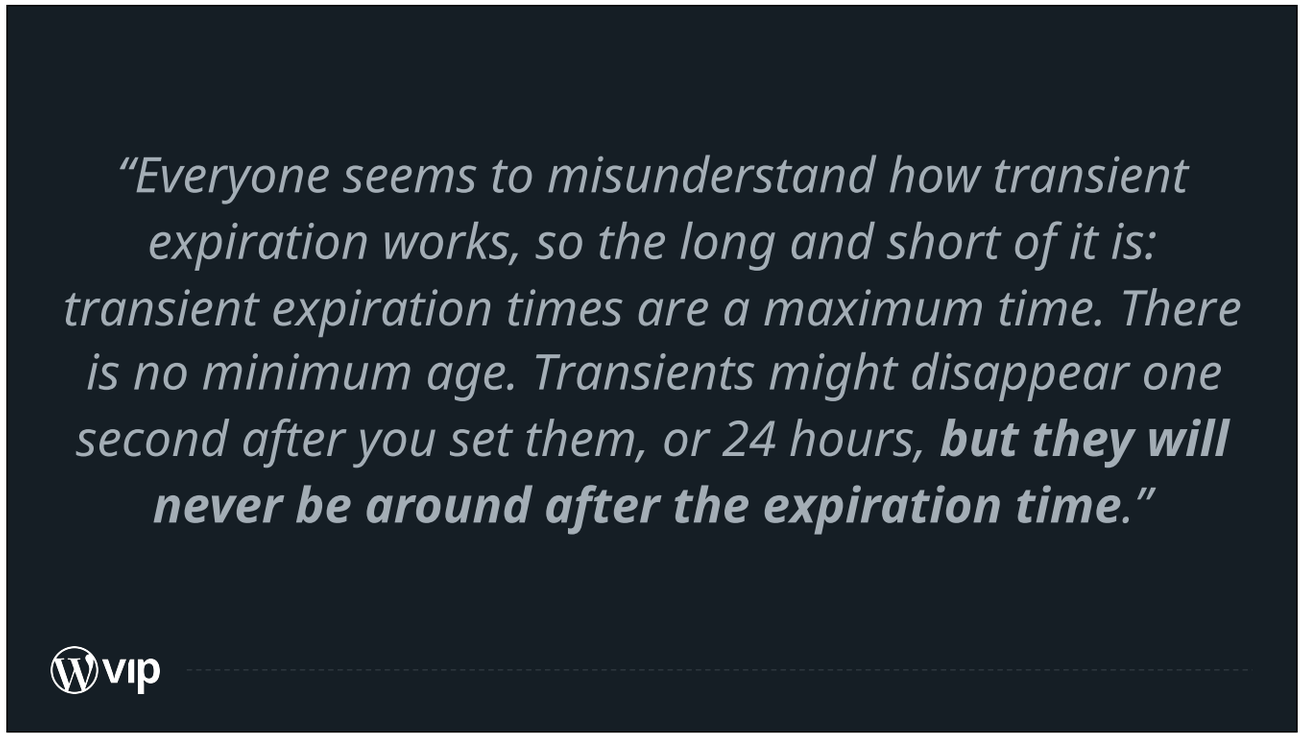
The forum admin’s response of “are you really asking why do you need a plugin?” was perfect.

No, this is the one that I mean…

“YouTube Embed [is] filling up my wp_options database with 3.8GB of logs”

What. The. Actual.

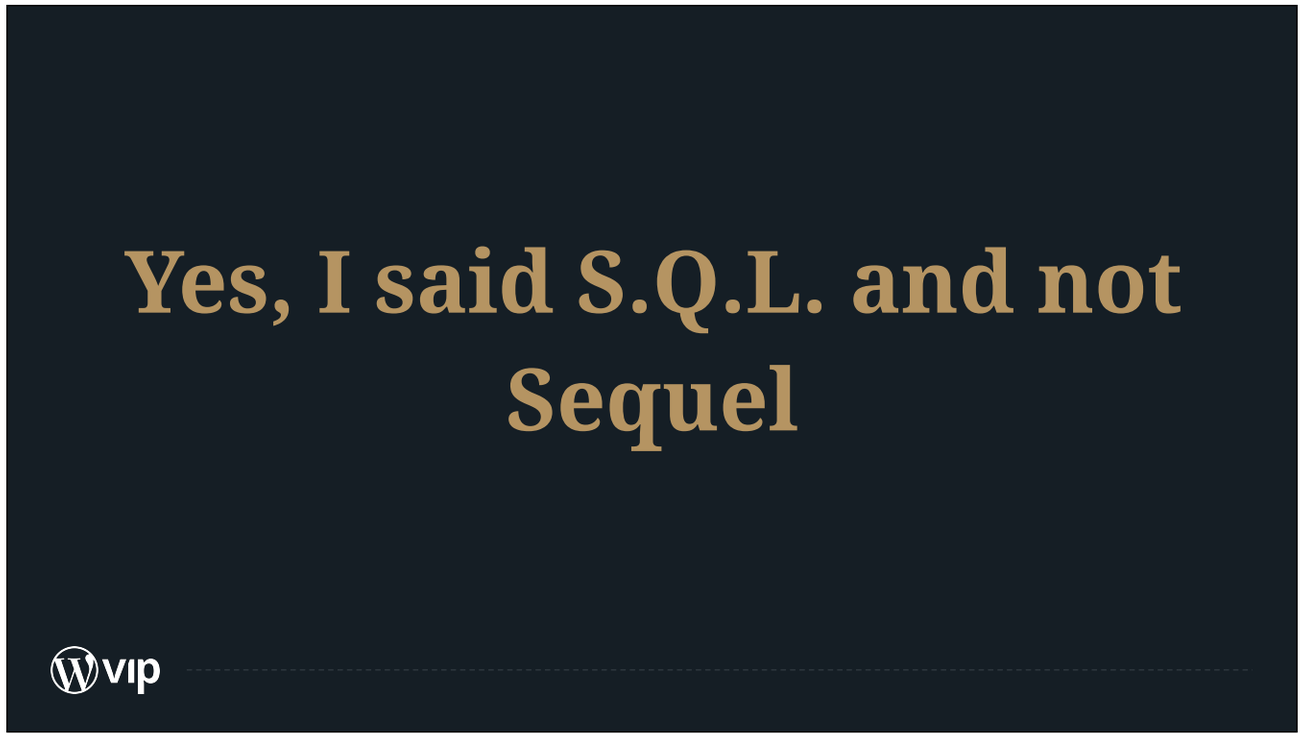
According to the codex…



My emphasis. Ironically, the person who wrote that also didn’t understand how transient expiration works. Let me explain why it’s wrong.

It turns out that WordPress does something pretty funky with transients. It housekeeps them when you try to access them. So, you ask the WordPress function to get a transient, it then looks to see if it has expired. If it has, it deletes the transient and returns nothing. In the case of my YouTube plugin, because the hash generated transient name had changed, the previous transient got left behind and never accessed again. So it never got deleted. Hence a 4GB options table.

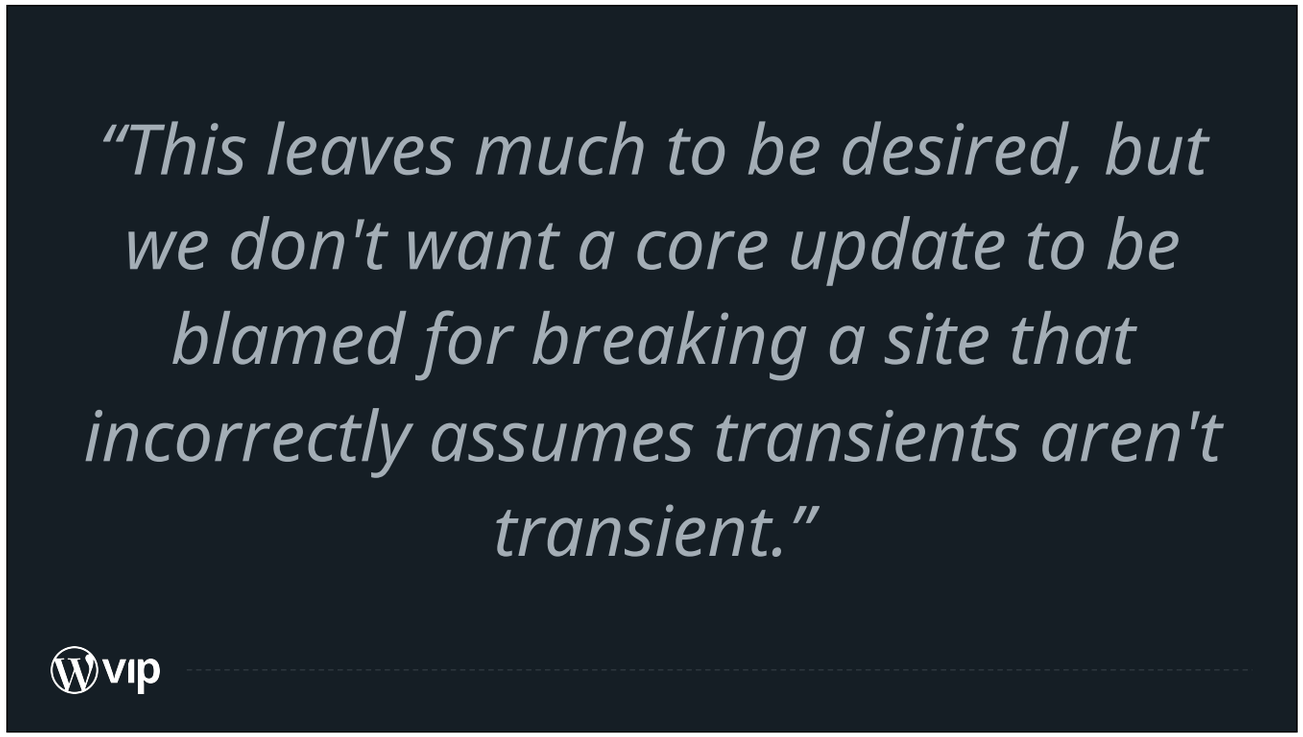
I turned to Trac and found that this, shall we say, sub-optimal way of housekeeping had already been reported. There hadn’t been much traction so, in the absence of a change in core, I wrote a plugin to plug the hole. Then movement occurred and the plan was to implement it in WordPress 3.7. Working with the SQL I’d used in my plugin…



…the proposal was to housekeep the expired transients on a daily schedule but also to nuke ALL transients every time a WordPress update required a database upgrade as well. The advantage of the latter was that it made any database upgrade quicker - the transients weren’t technically required so upgrading them just slowed down the update.

In the end, they implemented clearing only expired transients during a database upgrade and at no other time. The concern was over clearing transients daily, or fully during an upgrade, potentially breaking some sites. Essentially, plugin and theme developers could not be trusted to not be saving important data as transients.

As said on the core ticket at the time…



Another developer and I got props for the change and I was added to the list of Core Contributors for 3.7. I’m happy to report that housekeeping was finally fixed as of WordPress 4.9.

At this stage I’m also going to introduce you to this URL…

This is a part of my website that is going to include lots of documentation on transients and code examples too. I will also link to these slides and, eventually, the video of this talk. I say that in the future tense as it's not quite ready.

Now it's time to talk about **object cache** because if you're using this, transients work in a different way.

Again, we can turn to the codex to explain it a little more…

> *“Transients are inherently sped up by caching plugins, where normal Options are not. A memcached plugin, for example, would make WordPress store transient values in fast memory instead of in the database. For this reason, transients should be used to store any data that is expected to expire, or which can expire at any time. Transients should also never be assumed to be in the database, since they may not be stored there at all.”*
>
> WP VIP

Wow. That’s wordy. I’m not going to read that out loud.

So, first of all, do you have a persistent cache? Memcache is persistent, for example. If you don’t then WordPress stores Transient data both in that cache but, also, in the database, as usual. What this means is that if the cache is still available it will use that, otherwise it will fetch it from the database.

Basically, if you’re not using object cache then transients are on the database. If you are then they’ll also be in memory and may, or may not be, on the database as well. So, if we go back to our transient housekeeping, an SQL command is only going to work for those transients that are stored on the database. The good news is, though, that transients only make use of object cache if it’s persistent and should, theoretically, housekeep itself correctly. If there is no object cache or it’s not persistent, then only the database will be used.

Well, okay, no. That wasn’t my last foray into contributing to Core. Next up, I noticed an issue with transient name lengths.

To describe the issue, we need to go through how transients are stored in the options table.

The options table has 2 fields that we’re interested in here - the name and the content, and that first field had a fixed limit of 64 characters. To store the expiry time, transients actually consist of 2 options records. The first one has a name prefixed with _transient_…

and the second one with _transient_timeout_

To complicate this further, the network stored transients have prefixes of…

Transient database prefixes

```
_transient_*
_transient_timeout_*
_site_transient_*
_site_transient_timeout_*
```

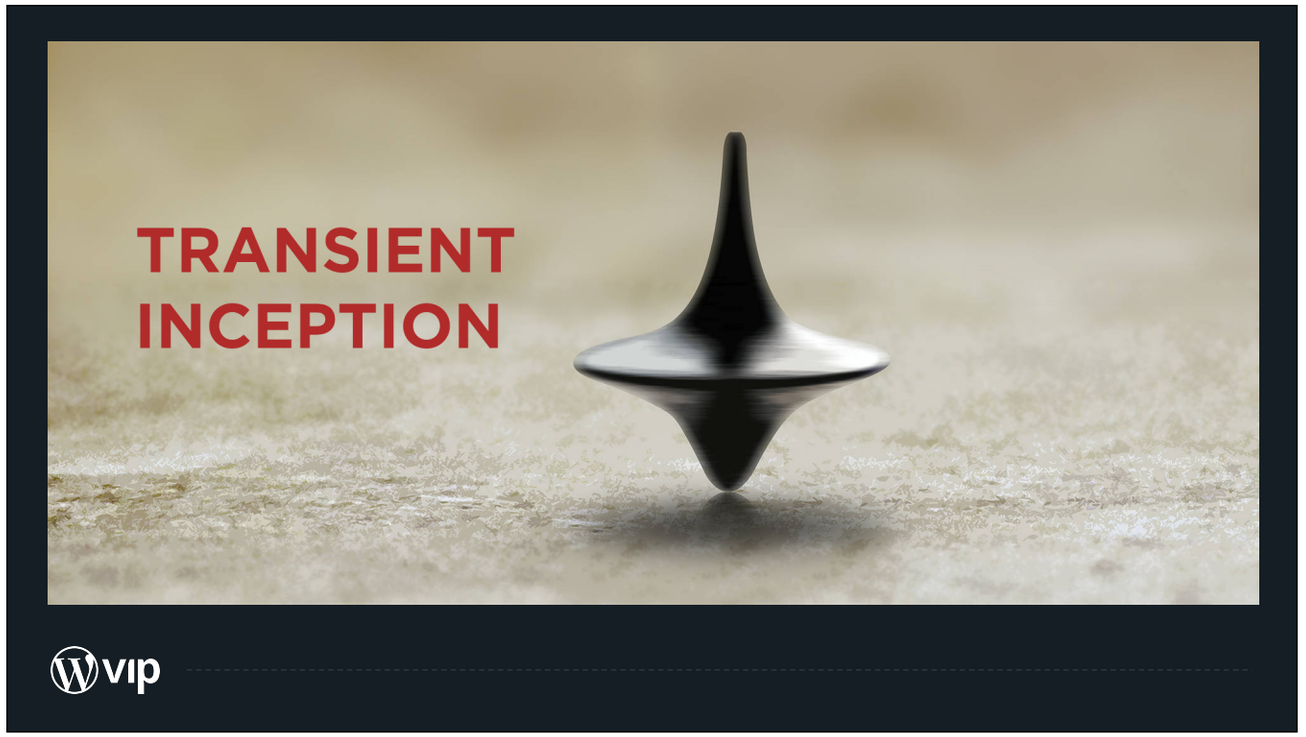
_site_transient_ and _site_transient_timeout_. This means that the maximum length of a transient name is dependant on which prefix is applied.

So, if the transient name you supplied was too long, the name would end truncated to 64 characters when it was stored in the database. Then, when you went to retrieve the transient, it would compare the full length name against what was on the options table and come up empty. This also meant that it was never housekept too as you would never access it again.

And if the only record too long was the timeout, it would only find the transient content and assume the transient was set to never expire, which could be even more dangerous.

Anyway, this lead to another core discussion and lots of solutions were thought of. In the end, though, the decision was made to expand that name field length to 256 characters, although that was also done for other reasons. However, rather than return an error if it was still too long, it was left to “silently fail”. i.e. just the same as before. Oh, and the codex was updated too <cough> (by myself) to mention the maximum name lengths.

Other solutions were discussed too, which included hashing the transient name if it was more than the maximum length. Imagine if you’d generated a transient name using a hash and it was too long so then got hashed again…

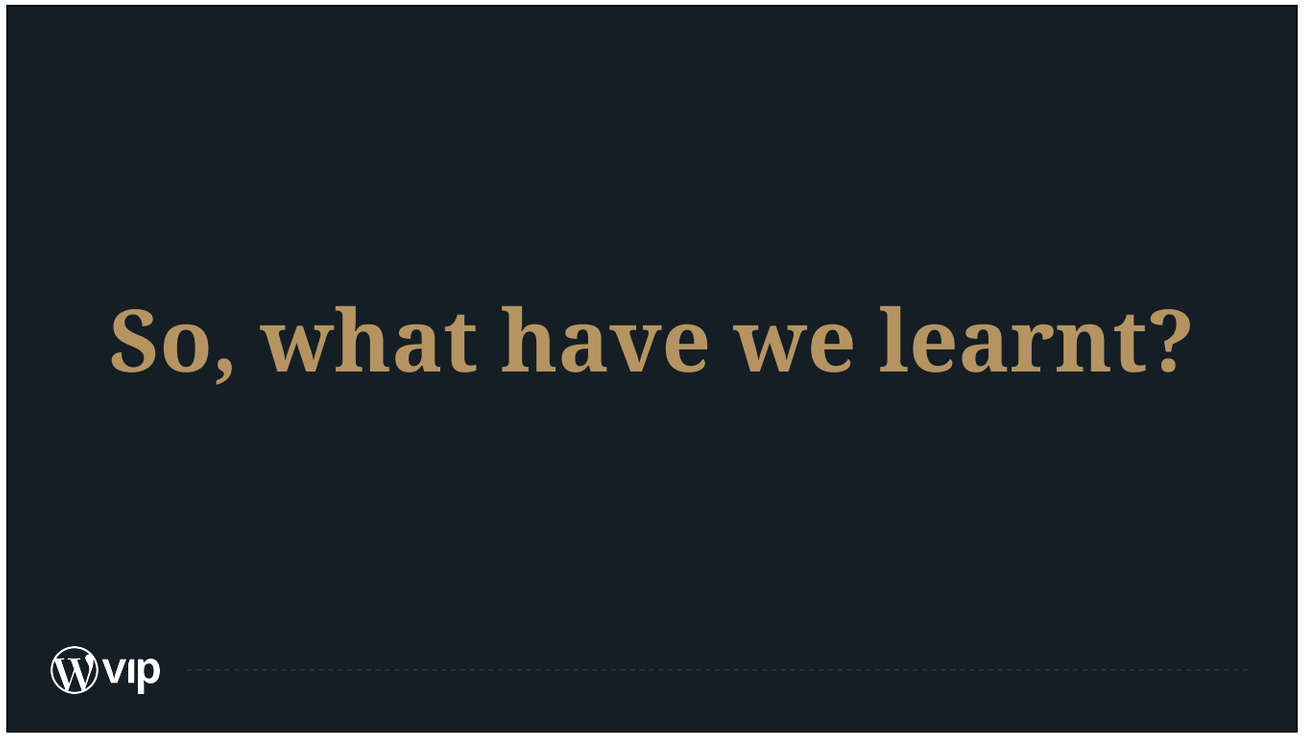


But, yes, for the second time an inherent issue in core ended up with little being done, which can feel a little like this…

But I understand it. The core team do not want to break existing sites and most of these issues are ones that's can be avoided by developers "doing things properly" in the first place.

Here's another tip… if you use a transient, if you can, please **allow the timeout to be adjustable by the user** and, if necessary, switched off. The hardest thing in the world to debug is data which may or may not be cached. Where I use transients in my plugins, I allow the users to change the frequency and that includes switching it off. That way, if they report an issue, the first thing I can get them to do is turn off the transient caching and I can be sure I'm always looking at the latest data.

He's still going.



So, what have we learnt? Apart from a ton about transients, we've also learnt that it's possible to become a core contributor by discussing fixes to issues that then don't get implemented. Mine is a rather edge case, granted. But that's not to say that I've given up on those issues as I've recently restarted the discussions on Trac about transient name lengths.

Final tip - if you have a plugin that makes use of transients, make sure it **deletes any transients as part of the uninstall** process, particularly if they're non-expiring transients.

Before I go, I want to talk about contributing, which has been a basis of this talk. And for this I'm going read straight from my notes, simply because it's important that I don't miss anything here.

Contributing

Contributing to WordPress - even core - can be easy and take little of your time. Depending on your abilities, you could help with support, documentation, accessibility, privacy, translations (here’s a tip - I’m a volunteer for translations. I translate from US English to UK English. It’s the easiest thing in the world!

Disclaimer

That’s not really true. Even converting British English to US English is more complex than most people think. It’s not all about taking the letter ‘u’ out of random words and spelling aluminium badly.
Who’d have thought? ¯_(ツ)_/¯

If you can vaguely wield a video editor, you could be the next Danny Boyle and help with WordPress.tv. Here is a link to get you started, if you’re interested…

I would recommend anyone, volunteering or not, to get yourself on the WordPress.org Slack where you can take part in, or just listen in to, community discussions. It can be REALLY eye-opening at times. When people talk about decisions being made behind closed doors, in reality it's happening here on Slack, for anybody to get involved with.

But you don't even need to be involved in one of the specific project areas to help out - just reporting issues with WordPress on Trac and then getting involved with the resulting discussion is massively helpful. WordPress is all about the community and, like a good savings account, the more you can put in the more you'll get out of it too. A savings account pre-Brexit, that is.

That's it for me. I hope you've enjoyed this and I'm happy to take any questions that you may have…

Questions?


We're Hiring.
Fully remote positions
vip
vip
Fully remote
Open vacation policy
Home office setup
Global company travel
Full family benefits
... and much more
Come work with us!
vip.wordpress.com/jobs
Thank You
vip