# Trienz

**Chapter 1: What is Trienz?: The Basics**

Trienz is a system for organizing collections of things [CoTs]. The system is designed for situations where in the types of the items in the collection are all any of the following: *stans*, *booms*, and *leaks*.

**Stans:** A stan is a normal item in a CoT. It is an item which is not incorporated by any other item. In Trienz, incorporation is when an item has another item to be a part of it, even though they are physically separate. Take for instance, a CSS file may be a different file, but it still happens to be logically part of an HTML file. So in this kind of situation, we say the HTML file incorporates the CSS file.

**Booms:** When a stan incorporates another item, the item it incorporates is called a boom. Take for instance, if a webpage [HTML file] contains a picture, the JPG file of the picture can be called a boom.

It is however important stating that while the examples given above are about computer files, Trienz is not limited to organizing only things related to computers. While the need to organize software source codes, in a better way, is what gave birth to this system, its use isn’t limited to computers only. You should rather feel free to use the system wherever you deem it fit.

Also, not only stans can have booms. Booms can also have their own booms, and lakes can also have booms. For instance if a PHP file A makes use of a function in PHP file B, and PHP file B further uses a function in PHP file C. The PHP file A can be called a stan, while the PHP file B can be called a boom. As for file C, it is also a boom.

**Lakes:** A lake is like a boom. But unlike a boom which is incorporated by a single item, a lake is an item incorporated by multiple other items. For instance, if multiple webpages [HTML files], make use of a common CSS file, the CSS file should be called a lake, not a boom.

And just like a boom, a lake can also be incorporated by a stan, a boom, or a goon.

By now, I don’t know if you’ve been able to grasp an idea of what Trienz is all about? Trienz is all about help organizing CoTs in which incorporation exists. And it does so in a way that expliticty shows the incorporations among the items in a CoT. In fact, organizing things in a way that expliticty shows the incorporations among the items in the CoT, is why Trienz was created.

**Chapter 2: Stans, Booms, and Tides**

In this first chapter, you got introduced to Trienz. But as for this chapter, we’d be taking a look at some of the elements in Trienz. Let’s start with stans.

*Before delving into the main discussion, you should note that in Trienz, we think of the items we’re organizing, as physical items, even though they might not actually be. We further think as if the place we’re organizing our items in, is a physical room. So when you start to hear things like: “divide the room into to”, “tie item X to Y, using a string,”, etc, do not get lost.*

If you recall, I described a stan as an item which is not incorporated by any other item. Well, a stan is as simple as that.

If you’d like to write down the name of a stan, all you have to do is to put down the “+” symbol, then followed by the stan’s name. For instance, if we have a stan called “index.html”, to write down it’s trien name, just write “+index.html”.

*A trien name is simply a name that can be used to identify an item within a trienstem.*

*A trienstem is a CoT that has been organized using Trien.*

**Booms**

You’ve also been introduced to booms. And just to recap, a boom is simply an item that is a part of some other item, even though it might be physically separate from its base [the item that incorporates it]. For instance, a CSS file is a boom of the HTML file it belongs to.

In Trienz, to show that an item is a boom of another item, simply tie a rope to it’s base. Afterwards, tie another rope to the boom. Then proceed with tieing the two loose ends of the ropes together, attaching the boom to its base. If the base has another boom, simply tie a new rope to the other boom. Then tie the loose end of the new rope to the point where the ropes of base and the first boom meet.

To put down the trien name of a boom, simply put down the trien name of its *base*, then followed by the “\*”, then followed by the actual name of the boom.

For instance, if the “index.html” file mentioned above has a boom called “style.css”, then its trien name would be “+index.html\*style.css”.

Recall that earlier on, I stated that booms can also have their own booms. So let’s assume we have a PHP file called “a.php”. And let’s assume that PHP file has another file called “b.php”. Then let’s further assume the “b.php” file has its own file called “c.php”. If we’re to put down the trien name of the file called “c.php”, we’re simply going to put down: “+a.php\*b.php\*c.php”.

**Special characters**

In Trienz, the name of an item can be made up of any character. However, there are some characters used by Trienz, and it is recommended to avoid using these characters in the name of your items. These special characters in question are: “[“, “]”, “+”, “\*”, “?”, “$”, “#“, and “/”. But if for some reason or the other you’d like to use any of the special characters for an item, when putting down the trien name of that item, always embed the special characters in its name within “[” and “]”. So if we have stan called “Anti-biotic [capsule] + B12 Tablet Review.doc”, the trien name of this stan would be “+Anti-biotic [[]capsule[]] [+] B12 Tablet Review.doc”.

**Tides**

There are times when you’d have a stan having many booms and you’d feel like some booms of the stan shouldn’t be allowed to mix freely with other booms of the stan. And rather than having them mix freely with other booms of the stan, you feel the booms should be brought together and put in a container, then the container should be tied to the stan.

For instance, imagine we have a stan called “index.php”, and it has the following booms:

function-1.php;

function-2.php;

function-1-from-some-othe-developer.php;

function-2-from-some-othe-developer.php.

You might feel like function-1-from-some-othe-developer.php and function-2-from-some-othe-developer.php shouldn’t be allowed to mix freely with function-1.php and function-2.php. And rather than having them mix freely, you might feel a container called “some-other-developer” should be created, then function-1-from-some-othe-developer.php and function-1-from-some-othe-developer.php should be renamed to function-1.php and function-2.php, then placed in the container.

Well, if you like to do something like what was described in paragraph above, it’s very possible in Trienz. But the only thing is that: in Trienz, we call containers *tides*.

If we’d like to put down the names of the items in the above example, the following are what we’d put down:

+index.php

+index.php\*function-1.php;

+index.php\*function-2.php;

+index.php?some-other-developer\*function-1.php;

+index.php?some-other-developer\*function-1.php;

As you must have noticed, whenever you want to put down the trien name of a container, always put down “?” before the actual name of the container.

It’s also worthwhile stating that it’s possible for a tide to be within a tide. In fact, there is no limit to how deep we can go, in putting tides within tides. You should also understand that items can have multiple tides, and tides can also have multiple tides within them.

You should further understand that a room can have tides in it. In other words, it’s possible for a room containing a stan called index.php, as well as a tide called “admin”. And the admin can contain two other stans called “sign-in.php” and “sign-out.php”. If we’d like to put down the trien name of the items in the example just mentioned, the following are what we’d put down.

+index.php

?admin+sign-in.php

?admin+sign-out.php

**Closing**

So to conclude this chapter, Trienz shows that an item is a boom of a base, by using a rope to tie them together. So whenever you see a rope tieing items together, you should automatically know that the one in front incorporates the one at the back.

**Chapter 3: Salts and Leaks**

In this first chapter, you were told that a leak happens to be an item incorporated by multiple items [the items that incorporate it may be stans, booms, or even some other leaks]. So, since a leak is related to more than one item, which item do we tie it to?

Well, we don’t tie the leak to any of its bases. What we rather do, is to divide create a section in the room, by dividing the room into two. Then we put the leak on right-hand-side of the room. So whenever you see a room divided, and you see some items on the RHS, automatically know that the items in the section are leaks.

In Trienz, the call the sections on the RHS *salts*.

Now, let’s look at an example. Imagine we have three files called “index.php”, “contact.php”, and “function.php”. And let’s also assume the first two files incorporate “function.php”. If we’re to organize these collection into a trienstem, here what we’d. Bring the first two files into the room. Then divide the room into two, creating a salt. Afterwards, bring in the third file, and put it in the salt.

If we’re to put down the trien names of the three files mentioned above, the following are what we’d put down:

+index.php

+contact.php

$1#function.php

As for the third file, the “$1” in its trien name simply indicates that it is within the first salt in the room. And yes a room can have more than salt: more on that soon. As for the “#”, this is simply a symbol that shows the name coming next is the name of a leak.

In the example given above, the room has only one salt, but it’s possible to have more than one salt. In fact, there is not limit to how many salts a room can have. And why would we ever need more than one salt?

Well, imagine we have a fourth item called “function-2.php”, the first three files are all its bases. It wouldn’t make much sense to put the fourth file in the same salt as the third file. So rather than putting the third and fourth files in the same salt, we create a second salt behind the first, then put the fourth file in the second salt. And the trien name of the fourth file would be “$2#function-2.php”.

So in short, whenever you see a leak that isn’t in the first salt, automatically understand that the leak is definitely a leak of another leak in a salt that’s directly in front of its own salt.

It is also important stating that tides can also have salts. But be careful in determining where to place a leak. It is completely forbidden for an item to incorporate a leak that’s inside a tide which is in the same room or tide. For instance, if a room contains a file called “index.php”, and also contains a tide called “admin”, it would be completely fobidden for “index.php” to incorporate: a leak within tide “admin” or any leak in a tide nested within “admin”. What is rather allowed is for items within a room/tide to incorporate leaks within the same room and leaks outside their room/tide.

So let’s assume we have a room containing contain a file called “index.php”. And the room also contains a tide called “admin” as well as a leak in the first salt called “function.php”. Then let’s also assume the “admin” contains another stan called “index.php” as well as a leak called “function.php”. In this case it is allowed for ?admin+index.php to incoporate $1#function.php. But it is unallowed for +index.php to incoporate ?admin$1#function.php.

One last thing you should note is that leaks should not be placed at levels upper than where they are being used. For instance, in the last example, let’s assume $1#function.php isn’t incoporated by +index.php, it shouldn’t be placed in the room. It should rather be placed within “admin”, since it is not used by any item at that level. The only exception is when a boom of an item [or a boom of a boom of an item, etc] within the level incorporates the leak.

**Closing**

So to conclude this chapter, whenever you see an item within a salt, automatically understand that the item is a leak. And don’t forget that the position of an leaks salt, also passes some information about the leak.

**Chapter 5: Meta**

So far, you have been exposed to almost all the types of things you might find in a *trienstai* [a collection of things organized using Trienz]. But what you haven’t be presented is a type of thing called *meta*.

In Trienz, you can think of a meta as a small placard which contains some information. Metas always have names as well information, on them. You might find metas in rooms, tides, and salts. It is also possible to find them attached to stans, booms, and leaks. Bricks [stans, booms, leaks, rooms, tides, and salts] may have zero or an infinite number of metas.

*So what’s the the usefulness of a meta?*

Well, metas are useful when you want to provide information about an brick. For instance, if you want to provide information about what type of file a file is, instead of adding the file’s type extension to the name of the file [e.g some-file.some-extension or better still image.jpg], you can simple create a meta called “file-type”, indicate what type of file the file is [on the meta], then attach the meta to the file.

There are many useful applications of metas, and the paragraph above is just an example of how this feature might prove useful.

Note metas can not have metas.

There is no restriction on what kind of symbols can be used for the names and the information of metas. But it’s strongly discourage using names that begin with “=”, for your metas. Names that begin with “=” are reserved for standard meta name. So refrain from naming your user specific metas things like “=some meta name”, “= SOME NAME!!!”, etc.

**Standard Meta Names**

1. “=**itemType”**: A name for metas that describe the type of an item.

So let’s assume you have a stan called “index.php”, and you’d like to put down its trien name, here is what you should put down “+index.php/=someType”. And yes, the “/” symbol is used to indicate that the name which follows is that of a meta.

**Chapter 7: Case sensitivity**

The name of elements [bricks and meta] as well as trien names, are case insensitive. So it’s possible to have two different items called “index.php” and “Index.php”. However, this kind of thing is strongly discouraged.

**Chapter 8: Who created Trienz**

A two-legged “homo sapiens” called [Ibrahim Oladipupo Qamardeen](https://qeetell.red), conceived Trienz. I was working on a pretty big project and felt the need to create something like Trienz. And here we are.

Since I created the system, I would also be handling its maintenance and improvement works. However, I do welcome suggestions.

If for any reason, you’d like to contact me, check out my personal website [[qeetell.red](https://qeetell.red)]. The website would have up-to-date and detailed information on how to contact me.

**Chapter 9: License**