# 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 separate file, but it still happens to be logically part of the HTML file it styles. 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 the files of a software’s source code, 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 leaks 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 is called a stan, while the PHP file B is called a boom. As for the PHP file C, it is also a boom.

**Leaks:** A leak is like a boom. But unlike a boom, which is incorporated by a single item, a leak 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 leak 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.

*Element: A general name for stans, booms, leaks, rooms, tides, salts, and metas.*

Note: in Trienz, we think of the items we’re organizing, as if they’re physical items, even though they might not actually be. We further think as if the place we’re organizing our items in, as if it were a physical room. So when you start to seeing things like: “*divide the room into two*”, “*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 trien name of a stan, all you have to do is to put down the “+” symbol, then followed by the stan’s actual name. For instance, if we have a stan called “index.html”, to write down it’s trien name, just write “+index.html”.

*Trien name: A name that can be used to uniquely identify an item within a Trienztine.*

*Trienztine: A collection of things organized using Trienz.*

**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*. For instance, a CSS file is a boom of the HTML file it belongs to.

*Base: The item that incorporates a boom.*

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, followed by the “\*” symbol, 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 the boom’s trien name would be “+index.html\*style.css”.

**Chained booms trien name**

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 a boom called “b.php”. Then let’s further assume the “b.php” file has its own boom called “c.php”. If we’re to put down the trien name of the file called “c.php”, here’s what we’re 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 elements. These special characters in question are:

“[“,

“]”,

“<”,

“>”,

“+”,

“\*”,

“?”,

“$”,

“#“,

“~”,

“!”,

“@”

“%”

“^”

“&”

and

“/”.

However, if for some reason or the other you’d like to use any of the special characters for an element’s name, when putting down their trien name, always embed the special characters in their name within “[” and “]”. So let’s assume we have stan called “Anti-biotic [capsule] + B12 Tablet Review.doc”, then 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 attached 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’d like to do something like that, it’s very possible in Trienz. However, the only thing is that: in Trienz, we call these containers *tides*.

So let’s assume we’ve put the two files in the tide called “some-other-developer”, and we’d like to put down the names of all 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 tide, always put down the “?” symbol, before putting down the actual name of the tide.

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

Another thing you should understand is that all bricks can all have tides. They can have multiple tides.

*Brick: A general name for stans, booms, leaks, rooms, tides, and salts.*

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

+index.php

?admin+sign-in.php

?admin+sign-out.php

**Closing**

To conclude this chapter, Trienz help 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 ones at the back.

**Chapter 3: Salts and Leaks**

In this first chapter, you were told that a leak is an item incorporated by multiple other 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 a leak to any of its bases. What we rather do, is to create a section in the room/tide, by dividing the room into two. Then we put the leak on right-hand side [RHS] of the room. So, whenever you see a room divided, and you see some items on the RHS, automatically understand that the items in the section are leaks.

Note: In Trienz, we call the sections on the right-hand sides *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 you’re to trien [organize using Trienz] these collection into a Trienztine, here what we would do:

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 one 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.

**We can have multiple salts**

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

Well, imagine we have a fourth item called “class.php”, and 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’d create a second salt behind the first salt, then put the fourth file in the second salt.

As for the trien name of the fourth file, it would be “$2#class.php”.

So, 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.

**Placing your leaks correctly**

I don’t know if you’re aware of the hint I’ve given, as regards it being possible for tides to also have salts. Well, it is actually possible. However, you have to be very careful when 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 the item is. For instance, if a room contains a file called “index.php”, and also contains a tide called “admin”, it would be completely forbidden for the “index.php” file 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 the 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” tide 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 incorporate “$1#function.php”. But it is forbidden for “+index.php” to incorporate “?admin$1#function.php”. I hope you get the gist. For those of you who understand Object-oriented Programming, you can think of this rule as a rule than enforces encapsulation. Things within a capsule can use things outside the capsule. But things outside the capsule can’t use things within the capsule.

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 incorporated by +index.php, it shouldn’t be placed in the room. It should rather be placed within the “admin” tide, 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 a leak’s salt, also tells some things about the leak.

**Chapter 4: Meta**

So far, you have been exposed to almost all types of elements you might find in a Trienztine. But one thing you haven’t been exposed to is a type of element called *meta*.

In Trienz, you can think of a meta as a small placard containing some information about a brick.

Metas always have names as well information, on them. You might find metas on the wall of rooms, tides, and salts. It is also possible to find metas attached to stans, booms, and leaks. Bricks 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 some information about a 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 just 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: There is no limit to how many metas a brick can have. But it is unacceptable to have multiple metas using the same name.

There is further no restriction on what kind of symbols can be used for the names and the information on metas. But it’s strongly discourage to use names that begin with the “=” symbol, for regular metas. Names that begin with the “=” symbols are meant for *reserved meta names*.

**Reserved Meta Names [RMN]**

There are going to be times when you’d feel like having control over what types of metas can be given a particular name. If you ever find yourself in such a situation, Trienz has got you covered. In Trienz, we have somethings called *reserved meta names [RMNs]*. An RMN is simply a meta name which an entity [individual, group, organization, government, etc] has control over, as regards what type of metas can be given the name. If you want an RMN, all you have to do is to use a name based on the following syntax:

**=[Heed type ID]:[Your heed]=[Some meta name]**

A real world example of an RMN is: “=ridn:red.qeevell.trienz=fileType”. In this example, the first “=” symbol simply indicates that the name is a reserved name. The “ridn” that follows is simply the ID of the type of heed you’re using.

For those who don’t understand what an heed is, an heed is simply a symbol or a set of symbols that uniquely identifies a thing in the omniverse. Take for instance, internet domain names can be considered heeds. A domain name ordinarily uniquely identifies a specific computing device. Under normal circumstances, the domain name “google.com” would uniquely identify a single computing device at Google. So whenever a symbol or a set of symbols uniquely identify something, we can call that thing an heed.

If we’re to put in other words, a heed can be seen as a name that is so unique that no other thing in the omniverse is called that name.

So when you need an RMN, one of the elements you’d need to include in the RMN, is your heed. We have a couple of heedtines [a system for creating heeds] in this world. And Trienz supports one of the, which is the “Reverse Internet Domain Name Heedtine [RIDNH]”.

The RIDNH is simply a system that creates heeds by reversing the order of an internet domain name. For instance, if we’re to create an heed from “google.com”, using RIDNH, the following is what we’d get “com.google.”

To use an RIDNH heed in Trienz, use “ridn” to indicate you’re using an heed created with the system.

So if you want an RMN, and you own a domain called “janedoe.com”, and you’d ordinarily like to name your meta “git”, then here is what your RMN would look like: “=ridn:com.janedoe=git”. I hope you get the gist.

As of this version of Trienz, only one heed type is supported by Trienz, and that heed type is the Reverse Internet Domain Name [RIDN].

It is completely forbidden to use someone else’s heed. For example if you don’t own the domain called “hello-world.com”, it is completely forbidden to use “com.hello-world” as your heed, unless given the permission.

You should also note that “heed type IDs” as well as “heeds” are case sensitives. So if you want to indicate that you’re using an RIDNH heed, use “ridn”, not “RIDN”, etc. Also, it is recommended to always use lowercase character for your heeds. For example, use things like “com.orangemarket” not “com.orangeMarket”.

**Some Trienz RMNs**

Trien has some RMNs. And if you’d name a meta of these RMNs, you must comply with the information provided, as regards what types of metas they can be used for,

**Trienz RMN 1: “=ridn:red.qeevell.trienz=trienz”:**

This RMN should be used only for metas of rooms and tides, which help show that the room or the tide was organized using Trienz. In fact, it is strongly recommended that every room and tide have a meta carrying this name. And the information on a meta carrying this name, should be what version of Trienz was used to organize the room or the tide. For example, the information on a meta having this name could be something like: “1.28” -- only the major and minor version numbers are need to be indicated.

**Trienz RMN 2:** **“=ridn:red.qeevell.trienz=fileType”:**

This RMN should be used only for metas which are owned by items which are electronic files. And a meta this name is given to, must also be a meta which provides information about what type of file a file is. For instance, if you’re creating a meta which indicates that an item is a JPG file, that meta can be given this name.

Also, the information carried by a meta given this RMN must be a “mime type”. For instance, a meta given this name can carry this information “image/jpg”, but not something like “This is file is a JPG image”.

**Meta trien-names**

Let’s assume you have a stan called “index.php”, and you’d like to put down the trien name of its meta “=ridn:red.qeevell.trienz=fileType”, here is what you should put down “+index.php/=ridn:red.qeevell.trienz=fileType”.

So yes, the “/” symbol is used to indicate that the name which follows is that of a meta.

If the meta whose trien name you’re trying to put down, belongs to the room of your Trienztine, you only have to put down the “/” symbol, followed by the meta’s name. Example: “/=ridn:red.qeevell.trienz=fileType”.

**Chapter 5: Case sensitivity**

The names of elements [bricks and metas] 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 6: Who created Trienz?**

A four-eyed “homo sapiens” called [Ibrahim Oladipupo Qamardeen](https://qeetell.red), conceived Trienz. I was working on a pretty big project and felt the need for something like Trienz. So I created it, 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 7: License**