Expressions

Expressions Overview

You may have seen some strange uses of expressions in the docs so far, so this doc will clear up things quite a bit. VAMP's expressions are geared towards "graceful handling" - which means it will not generate any failure unless the expression is malformed. This means your variables can have any data-type and your expression will hapily calculate it, either explicitly, or impliclitly; meaning that it will take care or things like: "devide by zero", or "negative zero", or "to the power of zero", etc, etc.

The "expression operators" `( )` can be used to "wrap" any value; like numbers (negative or positive numbers); -or even other expressions.

Expression categories

VAMP expressions can be devided into 3 categories. Each of these 3 categories have their own "expression governor" which handle expressions differently:

- `math` used for arithmetic & logical calculations
- `call` used for functions and methods with arguments
- `find` used for data lookups

Any of these categories can be nested infinitely.

MATH expressions

Before we get started here it is important to remember that "math" in this sense is NOT only "numbers".

In VAMP, a "math" expression is identified by any logical, arithmetic, reference, -or operators.
The "math expression governor" is a mean beast. It handles things is particular ways, depending on data-types, operators used and which data-type is on which side of the operator.
It also has "data modifier references", which can be changed -or extended, but it has defaults eplained below.

The best way to provide a concise reference of what it can do is to categorize things by data-types and operators used on them.

Take your time to go through the next couple of pages as it will give you a lot of insight into VAMP and what you can do with it, and how it makes your life as a developer easier. It may seem like a mountain at first, but once you get the gist of it, you won't have to remember all of it, but simply that you can do amazing things in your code, and it will most probably work accurately as you intend.

You will have to apply some savvy here, because if I have to list every possible way of how an arithmetic expression can be written, this chapter will probably span a million pages.

VAMP is concerned about what you code and what the logical results of that code should be as you intend it to be. For instance:

```
// JavaScript
// --------------------
   (0.1 + 0.2)             //    0.30000000000000004        wtf?
   ((0.1 + 0.2) == 0.3)    //    false                      kill me now!
// --------------------


:: VAMP
-----------------------
(0.1 + 0.2)                .:    0.3
((0.1 + 0.2) = 0.3)        .:    True
-----------------------
```

In the example above:

- The first block of code shows the result related to IEEE standards; more info here: https://en.wikipedia.org/wiki/IEEE_floating_point
- The second block of code shows how VAMP handles this by default.

VAMP does NOT just "throw away" the unnescessary digits; rather: it gets handled internally by looking at the expression input & output accordingly, so you don't need to bend your brain trying to figure out what the hell is wrong with your own logic -when it is perfectly fine.

In the examples below, arithmetics are handled as follows:

* (multiply)

When a numeric value is present on any side of the `*` operator, it is assumed to "duplicate" or multiply; else it is assumed to "join with" -or "merge with".

MATH :: Void & Zero

From here on, assume "zero" (or `0`) as "absolute zero".
Void & Zero is seen as "emptiness" and is numerically calculated as a number: "absolute zero" `0`.
In VAMP, "negative zero" & "positive zero" is kept "under the hood", so the final output from an expression will not show the "Spin" of zero. This is managed by the "Bios". Of coarse you can make a number like: `0.000000000001`, or `-0.000000000001` (as many zeros as your computer can handle) -but the point here is that it ends with a `1` and this makes it NOT "absolute zero".

You can of course use `-0` -or `+0` in your code, or your expression may end up calculating on these, but, it handles "zero" as "zero". This brings us to the following:

- if anything is added to Void, the Void will be filled with that thing
- if anything is subtracted from Void, the thing becomes its own negative
- if Void is added to anything, that thing remains unchanged
- if Void is subracted from anything, that thing remains unchanged
- if anything is devided by zero (or Void) then "that thing" remains unchanged
- zero to the power of zero, is zero

If you are a mathmatician and you are frowning heavily now, remember that you can always configure the way VAMP works internally, which is discussed in the "devl docs", -but in the "real world" these principles work very well and is able to do incredibly complex mathematics that produce accurate results with perfect precision.

```
()                  ::
().Type             :: Void

(() + 3)            :: 3
(() - 3)            :: -3

(() * ())           ::
(() * 0)            :: 0
(0 * ())            :: 0

(() / ())           ::
(() / 0)            :: 0
(0 / ())            :: 0
```

MATH :: Spin

```
.: TODO :.
-------------------------------------------------------------------------------
--
   Define how "Spin" would react whith different operators and data-types
--
-------------------------------------------------------------------------------
:: TODO ::
```