Part 1 | Introduction

An overview of CBEffects.

For version One and One-Half

About CBEffects

CBEffects is a free particle system for Corona SDK. It's very flexible, easy to use, and uses little memory. It can create just about any effect you can possibly think of. It comes in a folder, with several Lua files, some PNG files, and a few others. The CBEffects package is just over 500 KB in size.

CBEffects uses a separate physics file - "ParticlePhysics.lua" - for moving particles, thus it isn't hung up with the limitations that come with tedious transitions.

If you find an error or problem, please post!

http://developer.coronalabs.com/code/cbeffects

A Summary of CBEffects Usage

CBEffects is used by creating CBObjects. CBObjects can either be CBVentGroups or CBFieldGroups. Each type of CBObject may have many different child objects within it - for a CBVentGroup, they're called vents, for a CBFieldGroup, they're called fields.

A vent is something that emits a given type of particle, with different settings that can be adjusted. They can be used independently from fields. They are the thing that is seen on-screen. All changes to them are usually visible.

A field is something that modifies a vent's particles. They also have adjustable settings. They can not be used independently from vents. In other words, you must have a vent to have a field. A field may do things like change colors of a vent's particles, make them move in different ways, etc.

Both of these CBObject child types are used together to make an effect. For example, you may have a CBVentGroup that has a vent with particles emitting from it, going left at a velocity of two. Then, as they go across, there may be a CBFieldGroup with a field that is making the particles from the vent turn and go the opposite direction.

So, with a mixture of both of them, you get particles that are going left, and then, halfway across the screen, turn and go the other direction. Fields can adjust anything you want, not just motion. They have their own parameter, "onCollision", that is just a function that happens when a particle collides with it.

Part 2 | Manual

How to use CBEffects. For version One and One-Half.

Getting Started

To start with, copy the entire CBEffects folder into your project - no nesting, unless you want to change all of the require() paths - and at the top of your code add the following line:

```
local CBE=require("CBEffects.Library")
```

The main CBEffects file is called "Library.lua" and when you require it, you must require it as such. Once you've required it, you can start making particle effects.

Creating CBVentGroups

CBVentGroups are created with a simple method: the VentGroup function. Using it is as simple as this:

```
local CBVentGroup=CBE.VentGroup{
  {}
  {}
}
```

I can understand people might be frustrated with me changing the name (version One and version One and One-Fourth use the function .new), but it should be simple to change. If your text editor has a find and replace ability (as most do) you can just replace .new with .VentGroup. That's it.
I'm going to go into some reference terms, now.

The VentGroup function has one big table that includes all the others, starting on the first line and ending on the last one, if you noticed. I call that the "Master Table". So if you see the name Master Table (with caps), you'll know I mean that main table.

Inside of the Master Table, there are smaller tables. Each of those contain the data to create a separate vent. I'll call those the "Data Tables". So Data Tables are those nested tables inside of the Master Table.

There can be an unlimited number of Data Tables, but only one Master Table.

Each Data Table contains the data to create a specific vent, as I said before. So each one represents a vent. Sometimes, you'll only need one CBVentGroup, with several Data Tables creating vents inside of it.

When I refer to "on creation" and "on death", I'm talking about when the particle is made (using the build function) and when it's removed (after the lifeStart and lifeSpan).

Inside of each Data Table, there's quite a lot of possible parameters. However, you will never need to use every one. That will save you quite a lot of file space and time.

All of the parameters have a different use, and as there are quite a few parameters, the list is below.

Creating CBFieldGroups

CBFieldGroups are created with almost the same method - but this time, it's called the FieldGroup function. Here it is:

```
local CBFieldGroup=CBE.FieldGroup{
  {}
  {}
}
```

Everything is termed the same, the Master Table, the Data Tables, everything. CBFieldGroups are created and used almost exactly the same as CBVentGroups.

Since version One and One-Half, the ability to change color has been added. It is recommended to make a new field for color change, as it starts a transition. And creating a new color change transition over and over (as long as the particle is within bounds) uses *a lot* of memory. So your color change field should be a singleEffect field (see the parameter list for fields, given below). The color change function is only available if you are using the default table approach to colors, not using a function. If you are using a table for colors, here's how you change color to a particle from a field.

```
--The field's onCollision parameter:
onCollision=function(particle, field)
  particle.colorChange({255, 255, 255, 255}, 5000, 0, easing.outQuad)
end
```

The first argument is a table containing the color you want to transition to - RGB and optional A. It will still work without all of the values, but it may have unexpected results. The second argument is the time for the transition to take, the third is the delay, and the last is the easing effect.

Copyright information? Here it is:

The Not-Lawyerish Copyright Information

CBEffects is free to get.
CBEffects is free to edit.
CBEffects is free to use in a game.
CBEffects is free to use in an app.
CBEffects is free to use without crediting me. *
CBEffects is free to use without crediting CBEffects. **

CBEffects is NOT free to sell for any amount of money.
CBEffects is NOT free to sell for anything.
CBEffects is NOT free to credit yourself with.

*Although crediting me would be appreciated. I'm Caleb Place, and if you'd mention me, that would be nice.

**And crediting CBEffects might make it feel good about itself. The CBEffects logo is in the CBEffects folder - you can just display it somewhere and it will be happy.

Part 3 | Documentation

Parameter List for Vents

All of the possible parameters that can be inside of a Data Table for vents. Case-sensitive.

"title"
Type: String
Description: A unique title for the vent, so that you can access it with the "start", "stop", "destroy", and "get" commands.

```
title="MyVent"
```

"x" and "y"
Types: Numbers
Descriptions: X and Y of the vent.

```
x=display.contentCenterX, y=display.contentCenterY
```

"isActive"
Type: Boolean
Description: Whether the vent starts when startMaster command is called, true means it does start, false means it doesn't.
Note: Not used very often.

```
isActive=true
```

"build"
Type: Function
Description: Function that returns a display object to use as a particle.

```
build=function()
  return display.newImageRect("SomeImage.png", 50, 50)
end
```

"color"
Types: Table, Function
Description (table): Table with nested tables inside of it, each with RGB format. CBEffects chooses a random table for the color of the particle.

```
color={{255, 255, 255}, {0, 255, 0}}
```

Description (function): Function that returns anything that can be used as a color.

```
color=function()
  return aGradientYouCreatedEarlier
  --return 255, 255, 255, 80
end
```

"emitDelay"
Type: Number
Description: How long the delay between each emission is.

```
emitDelay=500
```

"perEmit"
Type: Number
Description: How many particles are emitted on each emission.

```
perEmit=4
```

"emissionNum"
Type: Number
Description: How many times the vent emits particles when started. Set to zero to be infinite.
Note: Do not set both "emissionNum" and "emitDelay" to zero.

```
emissionNum=0
```

"lifeSpan"
Types: Number, Function
Description (number): How long a particle takes to transition to endAlpha and be removed.

```
lifeSpan=1000
```

Description (function): A function that returns how long the particle takes to transition to endAlpha and be removed.

```
lifeSpan=function()
  return aNumberYouCreatedEarlier
end
```

"alpha"
Type: Number
Description: Alpha the particle will be just before the life transition starts.

```
alpha=1
```

"startAlpha"
Type: Number
Description: Alpha the particle is when it is created.

```
startAlpha=0
```

"endAlpha"
Type: Number
Description: Alpha the particle is when it is removed.

```
endAlpha=0
```

"onCreation"
Type: Function
Description: A function called on creation of each particle. The particle and the parent vent are both function arguments.

```
onCreation=function(particle, vent)
  MyGroup:insert(particle)
  v:translate(math.random(500), math.random(500))
end
```

"onDeath"
Type: Function
Description: A function called on death of each particle. The particle and the parent vent are both function arguments.

```
onDeath=function(particle, vent)
  print(particle.alpha, vent.x, vent.y)
end
```

"propertyTable"
Type: Table
Description: A table of values that is copied to each particle. When the particle is built, each parameter in the property table is given to it.

```
propertyTable={
  rotation=45,
  xScale=0.5
}
```

"scale"
Type: Number
Description: A number that pre-defined values of the vent are multiplied by to scale the vent.

```
scale=0.5
```

"lifeStart"
Types: Number, Function
Description (number): The delay before the particle begins its life span.

```
lifeStart=0
```

Description: (function): A function that returns the delay before the particle begins its life span.

```
lifeStart=function()
  return 5
end
```

"fadeInTime"
Type: Number
Description: The time it takes a particle to go from startAlpha to alpha and start the life start/life span transition.

```
fadeInTime=100
```

"positionType"
Types: String, Function
Description (string): The way CBEffects positions particles onCreation.
Possible Values: "inRect", "inRadius", "alongLine", "atPoint"

```
positionType="inRect"
```

Description (function): A function that returns X and Y values to position the particle on creation.

```
positionType=function()
  return math.random(500), math.random(500)
end
```

"posRadius"
Type: Number
Description: The radius the particles will appear inside of for "inRadius" position type.

```
posRadius=500
```

"posInner"
Type: Number
Description: The radius the particles will appear outside of for "inRadius" position type.
Note: Must be smaller than posRadius.

```
posInner=20
```

"point1" and "point2"
Types: Tables
Description: Tables containing two values, X and Y, so that if the position type is "alongLine" particles will appear along a line drawn from point1 to point2.

```
point1={0, 0}, point2={500, 500}
```

"rectLeft", "rectTop", "rectWidth", and "rectHeight"
Types: Numbers
Descriptions: Dimensions of the rectangle particles will appear inside of is "inRect" is the position type.

```
rectLeft=0, rectTop=0, rectWidth=50, rectHeight=50
```

"physics"
Type: Table
Description: Table that contains the physics properties of the vent. Uses the built-in ParticlePhysics, not Corona's Box2D physics.

```
physics={ ... }
```

Inside the Physics Table

"linearDamping"
Type: Number
Description: How the velocity of the particle changes. Positive values decay the velocity, negative values grow it.

```
linearDamping=0
```

"density"
Type: Number
Description: How gravity and forces act on a particle.

```
density=1.0
```

"velocity"
Type: Number
Description: How fast the particle moves.

```
velocity=3
```

"angularVelocity"
Type: Number
Description: How fast the particle rotates.

```
angularVelocity=8
```

"angularDamping"
Type: Number
Description: How the angular velocity of the particle changes. Positive values decay the angular velocity, negative values grow it.

```
angularDamping=0
```

"velFunction"
Type: Function
Description: Returns X and Y velocity for a particle on creation.

```
velFunction=function()
  return math.random(5), math.random(5)
end
```

"useFunction"
Type: Boolean
Description: Whether to use the velFunction to set velocity (true) or to use the pre-defined angles to set velocity (false).

```
useFunction=true
```

"autoAngle"
Type: Boolean
Description: Whether the values in the angle table are single numbers or auto-fill tables.

```
autoAngle=true
```

"angles"
Type: Table
Description: Contains the angles for particles to travel at. 0 is pointing right, and it goes counter-clockwise around - in other words, 90 is straight up, 180 is left, etc. If autoAngle = true then it includes angle high and low bounds, low to high.

```
--autoAngle=true
angles={
  {0,15},
  {180,195}
}
```

The example above will result in these angles:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,180,181,182,183,184,185,186,187,188,189,190,195

```
(angles, cont.)
--autoAngle=false
angles={
  0, 90, 180, 270
}
```

"sizeX" and "sizeY"
Types: Numbers
Descriptions: How fast the particles grow X and Y scale. Negative values shrink particles.

```
sizeX=0.1, sizeY=0.1
```

"minX", "minY", "maxX", "maxY"
Types: Numbers
Descriptions: The minimum and maximum X and Y scale the particles can grow or shrink to.

```
minX=0.1, minY=0.1, maxX=3, maxY=3
```

"gravityX" and "gravityY"
Types: Numbers
Descriptions: The X-gravity and Y-gravity of the vent.

```
gravityX=0, gravityY=9.8
```

(end of the physics table)

"rotation"
Type: Table
Description: A table containing the rotation data of the vent.

```
rotation={ ... }
```

Inside the Rotation Table

"towardVel"
Type: Boolean
Description: Whether the particle points the direction it is going.
Note: If set to true, the towardVel parameter will overwrite angularVelocity and angularDamping.

```
towardVel=false
```
"offset"
Type: Number
Description: If towardVel is set to true, particles will rotate toward the direction they are going + offset.

```
offset=0
```

(end of the rotation table)

(end of the vent parameters)

Parameter List for Fields

All of the parameters that can be inside of a Data Table for fields.

"title"
Type: String
Description: A unique title for the field, so that you can access it with the "start", "stop", "destroy", and "get" commands.

```
title="MyField"
```

"onCollision"
Type: Function
Description: A function that is called when a particle collides with the field. The particle and the field are both arguments.

```
onCollision=function(particle, field)
  p:applyForce(field.x-particle.x, field.y-particle.y)
end
```

"x" and "y"
Types: Numbers
Descriptions: The X and Y of the field. If the field's shape is a rectangle, then it is the top left of the field.

```
x=display.contentCenterX, y=display.contentCenterY
```

"rectLeft", "rectTop", "rectWidth", and "rectHeight"
Types: Numbers
Descriptions: Dimensions of the rectangle if the collision shape is set to "rect".

```
rectLeft=0, rectTop=0, rectWidth=100, rectHeight=100
```

"radius"
Type: Number
Description: Radius of the field if the collision shape is set to "circle".

```
radius=30
```

"points"
Type: Table
Description: Table containing pairs of coordinates for the polygon if the collision shape is set to "polygon".

```
points={0,0, 1024,0, 1024,768}
```

"targetVent"
Type: Special - must be a CBVentGroup's vent.
Description: A vent that the field will be connected to. Particles from it will be modified or moved. Easiest specified with the "get" function for a CBVentGroup.

```
targetVent=CBVentGroup:get("TheTargetVent")
```

IMPORTANT: Without this parameter, errors will occur.

"singleEffect"
Type: Boolean
Description: Whether the particles that collide with the field are then blocked from the onCollision with other fields (or the same one) with singleEffect set to true. If it's set to false, then the onCollision function will continue to fire until the particle leaves the bounds, even it the particle previously collided with a singleEffect field.
Note: This should be used for vents that start timers on a particle, transitions, color changes, etc. that you want to fire only once.

```
singleEffect=false
```

(end of the field parameters)

Methods for CBVentGroups

Functions executable to a CBVentGroup once it's been created.

"start"
API: CBVentGroup:start(t)
Description: Starts a CBVentGroup vent with title [t].
Effect: Vent with title [t] begins emitting particles at the speed of that vent's emitDelay, that vent's emissionNum in number.

```
CBVentGroup:start("MyVent")
```

"stop"
API: CBVentGroup:stop(t)
Description: Stops a CBVentGroup vent with title [t].
Effect: Vent with title [t] stops emitting particles.

```
CBVentGroup:stop("MyVent")
```

"startMaster"
API: CBVentGroup:startMaster()
Description: Starts all vents in a CBVentGroup that have isActive as true.
Effect: All vents with isActive as true begin emitting particles at their own emitDelay speed, their own emissionNum in number.

```
CBVentGroup:startMaster()
```

"stopMaster"
API: CBVentGroup:stopMaster()
Description: Stops all vents in a CBVentGroup.
Effect: All vents in the CBVentGroup stop.

```
CBVentGroup:stopMaster()
```

"destroy"
API: CBVentGroup:destroy(t)
Description: Vent with title [t] is completely annihilated.
Effect: Vent with title [t] ceases to exist.

```
CBVentGroup:destroy("MyVent")
```

"destroyMaster"
API: CBVentGroup:destroyMaster()
Description: CBVentGroup is completely annihilated.
Effect: CBVentGroup ceases to exist.

```
CBVentGroup:destroyMaster()
```

"emit"
API: CBVentGroup:emit(t)
Description: Vent with title [t] emits one round of particles.
Effect: Vent with title [t] emits particles.

```
CBVentGroup:emit("MyVent")
```

"emitMaster"
API: CBVentGroup:emitMaster()
Description: Emits one round of particles from each vent with isActive set to true.
Effect: Vents that are active emit a round of particles.

```
CBVentGroup:emitMaster()
```

"get"
API: CBVentGroup:get(t)
Description: Gets a vent with title [t].
Effect: Vent with title [t] is returned; can be used for repositioning vents, editing them, etc.

```
local chosenVent=CBVentGroup:get("MyVent")
```

"clean"
API: CBVentGroup:clean(t)
Description: Purges all particles from vent with title [t].
Effect: Particles are removed.

```
CBVentGroup:clean("MyVent")
```

Methods for CBFieldGroups

Functions executable to a CBFieldGroup once it's been created.

"start"
API: CBFieldGroup:start(t)
Description: Starts the field with title [t] checking for collisions.
Effect: Field starts listening.

```
CBFieldGroup:start("MyField")
```

"stop"
API: CBFieldGroup:stop(t)
Description: Stops the field with title [t] checking for collisions.
Effect: Field stops listening.

```
CBFieldGroup:stop("MyField")
```

"startMaster"
API: CBFieldGroup:startMaster()
Description: Starts all fields checking for collisions.
Effect: All fields start listening.

```
CBFieldGroup:startMaster()
```

"stopMaster"
API: CBFieldGroup:stopMaster()
Description: Stops all fields checking for collisions.
Effect: All fields stop listening.

```
CBFieldGroup:stopMaster()
```

"destroyMaster"
API: CBFieldGroup:destroyMaster()
Description: Completely annihilates all fields and the CBFieldGroup itself.
Effect: All fields, and the CBFieldGroup, cease to exist.

```
CBFieldGroup:destroyMaster()
```

"destroy"
API: CBFieldGroup:destroy(t)
Description: Completely annihilates field with title [t].
Effect: Field with title [t] ceases to exist.

```
CBFieldGroup:destroy("MyField")
```

"get"
API: CBFieldGroup:get(t)
Description: Gets a field with title [t].
Effect: Field with title [t] is returned; can be used for repositioning vents, editing them, etc.

```
CBFieldGroup:get("MyField")
```

So that's how you use CBEffects. Have fun and enjoy!
  If you have any questions, comments, don't understand a certain part of the documentation, want something changed, or something else, please comment at the link shown above.
  I do understand that it's quite a large amount of parameters and values to learn, but, as I said before, you'll never need all of them.