This documentation can also be found online at: https://spam.infinitevoid.games/

Intro

This is the official documentation of the SPAM-framework which can be purchased from Unity Asset Store.

You can also use the free-text search in the top right corner to find a specific component or setting like General ability settings or Effect settings in the online documentation

Video tutorials are also available on the Infinite Void YouTube-channel

The changelog can be found here

Quickstart

- Open the main window by going to either *Abilities->Abilities and Effects* or *Window->Abilities and effects*. This will open two new windows: **Abilities** and **Effects**.
- Click Create new ability, and choose one of the available Ability Types
- Create and add an Effect to your ability (see also: Create new effect)
- Add the ability to an invoker.
- Add an Ability target-component to the targets that should be affected by abilities.
- Cast your ability by either using the pre-supplied component `CastAtMousePosition` or see Casting and invoking abilities

Example project

There's an included example project, *Top Down Royale*, which can be imported if you go to the Unity Package Manager and select "Assets in Project" and then SPAM Framework.

Create new Ability

Ability Name
The name of your ability.

Path
The project path where the assets should be created. A good structure is to have a separate folder for each ability to keep the files organized.

Create subfolder
If you want to organize each ability in it's own folder (recommended), check this to automatically create a subfolder to the current path. The folder will have the same name as the ability.

Ability type
See Ability Types

Create new effect

This is where you create new effects to be used with your abilities.
All current effects in the project are listed to the left, and specific settings of the selected effect are shows to the right after you've selected an ability.

Create an effect by setting the path and selecting the type of effect you wish to add.

Basic settings

Path
The path where the effect asset should be created.

Effect type
Here you show which type of effect you will create. All effects that derive from `AbilityEffectSO` are show here, including your own effects.

Add ability to invoker

It's preferable to have all abilities under a separate child GameObject, or every ability under it's own child GameObject. This wont bog down the root GameObject with lots of components and make for a clearer structure.

Before you add an ability to an invoker, make sure that the root GameObject has an Ability invoker-component attached.

First, select the ability which you want to add to an invoker.

Next, in the scene hierarchy, select the game-object that should be the caster of the current ability, and click one of the buttons under *Add to selected gameobject*.

When you add an ability it will try to automatically resolve its dependencies (f.eg it's invoker) from its inter-GameObject hierarchy.

See also: Casting and invoking abilities, Directional ability component, Projectile ability component, Raycast ability component, Create new ability

Casting/invoking abilities

There's an example-component provided in the Components-folder that you can either use directly or look at for reference.

To cast abilities you first have to create an ability and add it to an invoker.

To make abilities effect targets you either have to add the pre-supplied Ability target-component to a GameObject or implement the `IAbilityTarget`-interface in one of your own `MonoBehaviours`.

The `Cast` method accepts a `Vector3` that is a point in the direction that the ability should be casted.
If the ability is targeted there's also an overload that accepts an `IAbilityTarget`.

`Cast` handles everything related to cooldowns, effects, timings etc., and there's multiple early exits that safeguards against performance degradation if you continually call it every frame. In other words: it's no problem spam `Cast` to your heart's content!

When an ability is cast, it goes through all of the Ability stages in order.

Casting targeted abilities

To cast a targeted ability you first have to resolve the actual `IAbilityTarget` or position that you wish to cast it on. Your game will decide how targets are selected, but a simplified example is:

```
// class CastTargetedAbility : MonoBehaviour

[SerializeField] private TargetedAbility _ability;

void Update()
{
        var leftMousePressed = Input.GetMouseButtonDown(0);
        if(!leftMousePressed) return;
        CastTargetedAbility();
}

void CastTargetedAbility()
{
        var mousePos = Input.mousePosition;
        var ray = Camera.main.ScreenPointToRay(mousePos);
        if (!Physics.Raycast(ray, out var hit, Mathf.Infinity)) return;

        if (_ability.RequiresTarget)
        {
            var target = hit.transform.GetComponent<IAbilityTarget>();
            if (target != null)
                _ability.Cast(target);
            return;
        }

        _ability.Cast(hit.point);
}
```

Casting other abilities

All other abilities except targeted only requires a point along the cast-direction. A simple starting point could be:

```
// class CastAbility : MonoBehaviour

[SerializeField] private AbilityBase _ability;

void Update()
{
        var leftMousePressed = Input.GetMouseButtonDown(0);
        if(!leftMousePressed) return;

        CastAbility();
}

void CastAbility()
{
        var direction = transform.forward;
        _ability.Cast(direction);
}
```

Effects

See also: Effect settings

Effects are what you abilities do when they are executed. To apply effects you either have to add the pre-supplied Ability target-component to a GameObject or implement the `IAbilityTarget`-interface in one of your own `MonoBehaviours`.
You can see all effects in the project by going to either *Abilities->Abilities and Effects* or *Window->Abilities and effects* and selecting the "Effects"-tab.

All effects derives from the base class `AbilityEffectSO`. Every class in the project that derives from this base can be applied as an effect to an ability, and will subsequently show up in the effects tab.

When an ability is cast (or when a projectile hits) the framework will resolve the target(s) of the ability and apply effects. The built in effects are applied by checking for an specific component-interface (f.eg. `IDamageable`) on the target and telling that component-implementation do execute some behaviour.

All interfaces that pre-supplied effects utilize can be found in *AbilitySystem/ExternalSystems*.

You are encouraged to create your own effects that suit your specific game, and free to implement them as you wish.

The pre-supplied "Deal Damage" effect looks like this (editor-only parts stripped out for brevity):

```
public class DamageEffectSO : AbilityEffectSO
{
    // Effect specific data. Here you put all the values that can vary
between different instances of the same effect
        [SerializeField] private int _damageValue;

        // This shows up in the editor window when creating an effect
        protected override string _metaHelpDescription =>
```

```
            $"Deals damage to the target. Requires the target to have an
IDamageable-component.";

        // This is the implementation for actually applying the effect to
a target
        public override void ApplyTo(IAbilityTarget target,
              Vector3 abilityPos,
              AbilityBaseSO ability,
              AbilityInvoker invoker)
        {
        var damageable =
target.Transform.GetComponent(typeof(IDamageable));

        if (damageable == null) return;

        damageable.Damage(_damageValue);
   }
}
```

General ability settings

These are the topmost and general settings of an ability.

See also: Ability Cooldown, Ability effects, Animations and Timings, SFX Settings, VFX Settings

Name
The name of the ability.

Description
The description of the ability.

Icon
The icon of the ability.

Ability cooldown
The cooldown of the ability.

Cooldown type
Can be set to either automatic or manual.

Automatic
The ability will enable itself and use it's own `Update()` loop to handle the cooldown.

Manual
You have to explicitly call `Tick Cooldown(float timeToTick)` to tick the cooldown. This can be useful in a turn-based game where each round ticks the cooldown of abilities.

Cast range
How far the ability can be casted from the caster. If set to 0 it can be cast at indefinite range. This value is used when checking if the given point or target is within casting range of the caster (Ability invoker).

Telegraph
Determines when the ability should be Telegraphed.
Works out of the box if you use an event-driven Telegraph, else you'll have to code up the logic for hiding/showing the telegraph yourself.

Custom settings
Here you can assign a custom settings object that can hold ability data that's applicable to your game.

Ability Cooldown

An ability's cooldown is controlled in Ability settings -> Ability Cooldown. It starts when the ability enters the Cast stage.

Ability effects

This is where you control what your ability actually does to its target(s). Effects are what you abilities actually do when cast on a target.

Effects list
This is a list of all effects that will be applied to the targets of the ability. The controls for each row are (from left to right):

- Remove effect from ability.
- Effect to apply
- Effect settings. Links to Effect settings.

- Time: The time it takes to apply the effect. This will add a delay between the given effect and the next one in order.
- On caster: If the effect should be applied to the caster instead of the target.

Add effect
Adds an effect to the ability.

Create new effect
Link to Create new effect.

AOE
Check this if the effects should be applied in an area. This will display AOE settings

Pre-conditions
See Ability pre-conditions.

Conditional effects
Conditional effects are a separate set of effect that's only applied when a target meets a certain set of condition constraints. These effects can be applied before or after the ability's main effects. See also Conditional effects.

Animations & Timings

This is where you set up the timings for each stage of your ability. The object created to hold these values can be reused with many abilities to keep the same animation timings for multiple abilities.

Timings
You can set these values manually, or add you clip to the corresponding Animation Clip slot and then press 'Set timings from animations' to automatically map the length of the animation to the timing of the stage.

Animator parameters
Here you set the parameters to be triggerd at each stage of the ability. Not that when you use a boolean for 'Cast' then you have to manuall call `Ability.StopCasting()` to make the bool switch to an off-state. This is useful for looping animations or abilities like firing a machine-gun or casting a quick projectile.
It's also possible to setup this type of abilities with a trigger that continously is triggered since the actual values are cached at startup the cost of setting a parameter is greatly reduced.

Animations
Add the clips here to automatically use the clip length in Timings. Note that these are only used in edit-mode so you have to manually press 'Set timings from animations' to use these. The animations are never used at runtime.

Area of effect settings

Settings for how an ability should be applied in an area. These differ slighty if the ability is a AbilityTypes/Directional ability.

Effect radius (Non-directional Only)
The radius of the sphere that's used to check which tagets the effect should be applied to.

Area of Effect Type
If a box or sphere should be used for checking the area of effect.

Angle (Directional only)
The angle of the effect. If set to f.eg. 180 the ability will hit in a 180 degree wide circle.

Distance (Directional only)
The distance of the effect. This in combination with angle will create a wedge/pie slice which gives the total area an ability will look for targets.

Max effect targets
The number of maximum targets the ability can affect each cast.

Line of sight check
If a line of sight raycast should be used from the ability impact point to the target to decide if the target should be affected or not.

Los blocking layers
The layers that will block the ability's line of sight and prevent the effect to be applied to a given target.

Effect layers
The layers that holds the potential ability targets.

Effect application
Example: Ability has two effects and five targets.
Per effect
Each effect will be added to each target in order.

The first effect will be applied to all five targets and then the next effect will be applied to all five targets.
This could be viewed as a standard behaviour where a fireball explodes and damages all targets simultaneously.

Per target
The first target will have each effect applied in order, then the second target will have each effect applied in order and so on until all effects are applied.
This is usefull if you for example spawn particles on each target as an effect (say lightning strike), which would give the ability a wave-like application where there would be 5 hits in succession with a delay in between instead of 5 immediate lightning strikes.

Per target application delay
Only visible when effect application is set to "Per Target". This would be set to the delay between each lightning strike in the example above, for example 0.2s

SFX Settings

This is where you setup the SFX (Sound Effects) for the different stages of an ability.

Warmup SFX
The SFX to be played on the start of Warmup. Is always played from the Ability invoker's `AudioSource`.

Cast SFX
The SFX to be played at the start of Cast. Is always played from the Ability invoker's `AudioSource`.

On Hit SFX
The SFX to be played when an ability hits a target. If you supply multiple audio clips here a random one will be chosen to be played.
The target's `AudioSource` will be used if applicable, else the ability invoker's `AudioSource` will be used.

VFX Settings

This is where you setup the VFX (Visual Effects) for an ability. An ability's VFX is handled in `AbilityVFXHandler`.

Warmup

Warmup VFX
The VFX to spawn at the beginning of the Warmup stage. This VFX is cached and will be reused on every cast.
The spawn position for the warmup is set though the actual ability component that's attached to a GameObject.

Warmup lifetime
Determines for how long the warmup vfx should be active for after it's spawned.

Warmup custom scale
Since the warmup VFX gets its spawn position from its ability component, and you might want the warmup VFX to be spawned deep in a hierarchy to make it automatically move with an animation, you can set a custom scale to be used here.
This is useful if you for example attach the spawn position transform as a child of a staff, (which in turn is attached to a character's hands etc.) to make it automatically follow the staff when the character is animating, but for some reason any transform in the hierarchy needs a scale that isn't 1. You can then set a custom scale here to make your warmup-vfx have a scale that matches your needs.

On hit

On hit VFX
The on hit VFX to be spawned when an ability hits a target or position.

Num of On Hit Instances
Since the on hit VFX is cached and reused, you have to set the number of instances you want to pre-spawn at startup. This setting can be either same as effect targets or custom.

Same as effect targets
This will spawn as many VFX instances as the maximum effect targets of the ability. Useful if you want to display a quick hit effect when firing a shotgun for example.

Custom
This will spawn the given value of VFX instances. Useful when you want a projectile that explodes at the impact point or an ability that fires rapidly like a machine-gun.

On hit spawn point
Can be set to either Targets or Impact point

Targets
Will spawn the VFX at the center of the target if not 'Spawn on hit at target base' is

checked.

Impact point
Will spawn the VFX at the point of impact, i.e. the cast point or where a projectile hits.

On hit position offset
The offset to be applied to the spawned VFX.

Life time handling
Can be set to either automatic or manual.

Automatic
The VFX handles it's own lifetime. Useful when you have a one-shot VFX/particle effect that doesn't loop and can be left alive after it's done.

Manual
This setting will allow you to set a custom lifetime for the VFX. If you set this to 0 it will have the same effect as Automatic.

Ability Types

There are four different types of abilities in the SPAM-framework.

- Targeted ability
- Projectile ability
- Directional ability
- Raycast ability

See also: General settings

Targeted ability

An ability that's cast either directly on an `IAbilityTarget` or at a given point, provided that *Requires ability target* set to false

See also: General settings, Ability component settings

Specific settings

Requires ability target
Check this is the ability requires a valid `IAbilityTarget` to be casted.

Projectile ability

A projectile ability will have it's effects delayed and instead spawn a projectile on Cast that will apply all Ability effects to the target or area it hits. Can be either targeted or directional. When the ability is targeted, it can be set to require an Ability target to be cast.

See also: General settings, Ability component settings

Additional settings

Projectile prefab
The projectile to be spawned. Note that this prefab needs to have a `Projectile` script attached. See (or use) the proved examples in Prefabs/Projectiles.

Time to live
The time until a projectile despawns if it doesn't hit a target.

In flight sound
The sound to be played from the Projectiles `AudioSource` when it's traveling.

On hit action
What the projectile should do when it hit's a target. Values are *None*, *Disable*, and *Deactivate*.

None
Only apply effects and VFS/SFX, but don't deactivate the projectile. This is useful if you want a piercing projectile that doesn't stop at impact.

Disable
Stop the projectile's movement at the point of impact and disable it's collider so it can't affect more targets.

Deactivate
Deactivates the projectiles visuals and colliders, and then the projectile completely after it played its on hit SFX.

Directional projectile settings

These settings apply (and are only visible) if a projectile is meant to be a directional projectile that doesn't need a target to be cast, i.e. when requires ability target is set to off.

Distance check range
How close to the target point the projectile will deactivate and apply it's On hit effects. This should be set to a value greater than 0 to not have the projectile accidentally travel past it's target.

Spawned projectiles

This is where you set which projectiles should be spawned when the ability is cast.

Spawn time
How much delay there should be before the projectile is spawned. Can either be a constant value or a random value between two constants that's evaluated on each cast.

Movement Behaviour
This is where you add the movement behaviour the spawned projectile should have. The + button will create a new movement asset that you can configure. Note that the movement asset is currently only editable in the inspector and not in a SPAM-Window.
See also: Projectile movement behaviour

Projectile movement behaviour

Each projectile can have individual (or shared) movement behaviour. A movement behaviour is an asset that you configure through the inspector after it's created.

Base speed
The base speed of the projectile.

Desired rotation
The desired rotation the projectile should have in flight.

Movement type
Projectile can currently have straight movement (directly towards target) or curved movement. If you select curved movement, more settings will be displayed (see below).

Curved Movement Settings

Here you can set how the projectiles movement should be offset while it's traveling towards its target.

When the given setting is applicable and configured correctly, a green dot will be displayed next to its header. Both a curve and strength must be set for a settings to be applicable.

Horizontal movement
Horizontal movement can be though of as seeing the projectile from a top-down perspective. If you add a positive value, the projectile will be offset to the left of it's current direction, and if you set a negative value, it will be offset to the right (when seen from above).

Vertical movement
This is how the projectile should travel in the Y axis when traveling towards it's target. A value of 0 will not affect it's height, and positive/negative values will raise/lower the projectiles height during flight.

Speed
If you with the projectile to have different speeds at different times of it's path, you can set a curve for it here.

Strength
All curves can either have a constant strength (same movement every time) or random between two constants strength (random strength between given values each time the projectile is spawned). Strength determines how much the curve should affect the projectile's movement direction.

Directional ability

An ability that's cast in a given direction from the caster.

See also: General settings, Ability component settings

Raycast ability

An ability that's invoked by shooting a ray from a given position in a given direction to check for a potential target.

This is useful for abilities like a shot in First-Person games, as you don't need to continually raycast every frame but instead only when you want to fire your ability or weapon. The ability will not continously fire rays as it will return early if the ability is on cooldown.

See also: General settings, Ability component settings

Additional settings

Raycast length
The max-length of the ray that casted as part of invoking the ability.

Raycast Layers
The layers that the ability will check if it hits something on.

Requires ability target
Check this is the ability requires a valid `IAbilityTarget` to be casted, i.e. the ray must hit an object that has an `IAbilityTarget component`.

Ability stages

There are three main stages an ability goes through when cast: Warmup, Cast, Cooldown.

Warmup

Cast

Cooldown

Warmup

Warmup is the stage when the ability is prepared to be casted. Use this to set a delay before an ability is cast and its cooldown begins.

Cast

The stage that comes after Warmup. This is the stage when the ability is Invoked and effects are applied (if it's not a AbilityTypes/Projectile ability, projectiles apply their effects when they collide).
When an ability enter this stage, it's Ability Cooldown will also start.

Cooldown

The last stage that comes after Cast. This is a stage where you could have wind-down animations.

Effect settings

General settings

Name
The name of your effect

Description
The description of your effect.

Included in ability effects
Check this if you want to include the effect in `AbilityBase.Effects`.
Turning this off is useful if you want to exclude the effect from an ability's public list of effects, meaning you can safely list all effects and their name/description in the UI.

Effect specific settings

The specific settings for the effect type will show up here, depending on the currently selected effect.

For example a **Damage** effect will show *Damage value*, and an **Explosion** effect will show *force* and *up modifier*.

If you create your own effect, all its serialized fields will be shown here.

General ability component settings

These settings are common for all `ability components`, regardless of type.

See also: Directional ability component, Projectile ability component, Raycast ability component, Targeted ability component, Components

Invoker
The Ability invoker that will cast the ability. This will try to automatically resolve itself from the hierarchy of the `GameObject` the `ability component` is added to. If no Ability invoker is set a button will appear to try to resolve this automatically from the hierarchy again.

Event Controls
Raise cooldown event
If the the cooldown event should be raised in `AbilitySystemEvents`. Use this if you have custom code listening for this event. Should most oftenly be turned of if an ability has a short cooldown.

Raise used event
If the used event should be raised in `AbilitySystemEvents`. Use this if you have custom code listening for this event. Should most oftenly be turned of if an ability has a short cooldown.
This event is also used for automatically showing an event-driven Telegraph

Warmup VFX spawn point
The transform where warmup VFX (if any) will spawn at.

Ability invoker

A base `component` which is required for casting abilities. This `component` handles the general flow of casting abilities, i.e. triggering the different stages at the correct timings.

Only one Invoker is required per GameObject-hierarchy (caster).

Ability target

Pre-supplied example component to quickly enable the GameObject to be the target of Ability effects. You can also implement `IAbilityTarget` yourself to enable this behaviour.

The framework checks for this interface before applying effects.

Audio source
An optional audio source that will be used when playing either custom Ability effect SFX or on hit SFX (see also: SFX Settings).

Targeted ability component

The actual component that holds a reference to a Targeted ability.

Targeted ability
The ability that this component controls.

Visualize AOE
Check this to display a gizmo that represents the area-of-effect of the ability. This is done only in editor and can be changed live.
If the ability is not an area-of-effect ability, this checkbox has no effect.

See also: Ability component settings

Directional ability component

The actual component that holds a reference to a Directional ability.

Directional ability
The ability that this component controls.

Visualize AOE
Check this to show a gizmo that represents the area-of-effect of the ability. This is updated in real-time so if you change the angle or range the gizmo will also update. This visualization is in-editor only.

See also: Ability component settings

Projectile ability component

Note: the following also applies to "Targeted Ability Component"

The actual component that holds a reference to a Projectile ability via it's Projectile Pool. The pool will be automatically created when adding a projectile ability to a GameObject.

Projectile pool
The pool that handles the projectiles of this ability. The actual projectile-data for the ability is resolved through this pool. This is automatically created to the root of the scene when adding an ability to a GameObject. If you wish to reuse a pool, simple change the reference and remove the newly created pool.

Spawn point
Where the projectile should be spawned from.

See also: Ability component settings

Raycast ability component

The actual component that holds a reference to a Raycast ability.

Raycast ability
The ability that this component controls.

Raycast from
Where to raycast from. This will resolve to Camera.main (cached on start) if not set.

See also: Ability component settings

Projectile Pool

A pool of projectiles. Will automatically spawn the given projectile on start.
This is used to prevent garbage from being generated at runtime, since each Projectile ability requires a pool of projectiles.

Projectile in pool
The projectile that lives in the pool, and will be spawned on Start.

No in pool
The number of projectile to spawn in the pool. This value is static during runtime.

Keep alive
Check this if you have multiple abilities referencing this pool.
When an ability is destroyed (f.eg when a caster with an ability that references the pool is destroyed) it will call `Destroy` on the pool, effectively breaking all other references to the pool. This could lead to null reference exceptions or other unintended behaviour if you still have references to the pool itself or any of its projectiles.

Please note that if you decide to keep a pool alive you'll have to handle it's lifetime manually.

Telegraph

A visual representation of the area where an ability will apply it's effects. There are two pre-supplied telegraphs, Circle and Wedge, which can be found under the *Prefabs* folder. If you use events to handle telegraphing of abilities you would theoretically only need one telegraph since it will update itself automatically (both size and position) when `AbilityEventSystem.AbilityUsedEvent` is raised.
This has some limitations as there can only be only telegraph active in the scene at one time, so it's not applicable if you need multiple telegraphs for abilities. For this use-case you will have to write your own custom code to handle a pool of telegraphs. Note that projectile abilities can also be telegraphed by assigning a telegraph to the ability.

Event driven
Check this on the telegraph to enable it to be controlled by events from the AbilitySystemEvent-Mediator. If this is set to false then the telegraph has to be controlled manually.

Circle telegraph

This telegraph works by scaling two separate quads with a custom shader applied. The outer quad will automatically be set to a correct size depending on the ability's AOE-radius settings.
If the ability is set to be telegraphed during warmup then the inner quad will automatically scale itself up for the Warmup-duration until it reaches the same size as the outer quad when Warmup has elapsed just before Cast

Scale factor
Take a quad and apply the given material needed to display only the telegraph's circular texture. Then take a normal sphere and position it at the exact same position of the quad so they overlap. Then scale the sphere **uniformly on all axes** until it exactly lines up with the telegraph's texture. The scale factor will then be 0.5 divided the scale of the sphere.
Ex: Sphere scale after following above steps: 1.2
Scale factor: 0.5/1.2 = 0.416

Wedge telegraph

The telegraph works effectively the same as the cricle telegraph, but it renders the wedge as a mesh at runtime.

Update frequency
The delay between mesh-redraws.

Setting this to a value higher than 0 is most often preferred since a small delay in redrawing is rarely noticeable and increases performance.

Conditions

See also: Conditional effects

Conditions, also known as status effects, DOT/HOT, states, and more, can be applied to a target when an ability hits. Conditions can do various things, such as apply additional effects too the target, mark it, and/or act as rules (Ability pre-conditions) for when an ability's effects should be applied. Conditions were added in SPAM version 1.2.0

You can open the conditions window by going to *Tools->Spam Framework->Conditions* or *Tools->Spam Framework->All Windows*.

For a target to be affected by conditions it needs both an Ability target- and an Ability Conditions Target-component. If you want condition VFX to be displayed on the target it also need an Ability Conditions VFX-component.

To apply or remove conditions with abilities, use the add- and remove condition ability effect.

General settings

Name
The name of the condition

Description
A description of the condition.

Add Multiple Behaviour
How the condition behaves when it's applied to a target that already has the condition.

Do nothing
The condition is not applied, i.e the existing condition is kept and the new one discarded

Extend
Add the applied condition's lifetime (if any) to the existing condition's lifetime.

Overwrite
Overwrites the previous condition and its (if any) lifetime. Note that will fire the *added* event again.

Stack
Adds this condition to the target, resulting in the target having two instances of the same condition (but with separate lifetimes, if any).

VFX settings

VFX
The Condition VFX that should be displayed on the target while it's affected by the condition.

Num Pooled
All condition VFX is pooled per condition. This values determines how many instances of the VFX that should be available. Set this number as low as possible to save memory, but as high as needed to display at all affected targets.

Spawn at target base
If the VFX should spawn at the target's base. If this is not checked the VFX will spawn at the targets position (often its center)

Spawn offset
The offset from the spawn position that the VFX should spawn. This is the general offset of the VFX. Each target can set it's own VFX position in the Ability Conditions VFX component, which will override this setting.

Play On Events
When the VFX should be played. When one of the selected events are fired, the framework will call Stop() and then Play() on the particle system of the given Condition VFX.

Secondary effects (and events)

Conditions can apply Ability effects during their lifetime. An effect can be applied at multiple events, and possible values are:

Added
The effect is applied when the target doesn't already have the condition, or when it has the condition but it's Add Multiple Behaviour is set to *Overwrite* or *Stack*.

Ticked
The effect is applied on each "tick". See "time between ticks" below.

Extended
The effect is applied when the condition's lifetime was extended.

Expired
The effect is applied when the conditions lifetime ended.

Removed
The effect is applied when the condition is removed before its lifetime ended (or if it was permanent and removed by another ability). See also Adding and removing conditions.

Time between ticks
If this is set to a number higher than 0, all effects with the *ticked*-event will be applied at this interval. For example, a values of 1 means ticked effects are applied every second.

Adding and removing conditions

To add conditions to a target, it needs both an Ability target- and an Ability Conditions Target-component.

Conditions are added and removed with an either Ability effects (AddCondition and RemoveCondition) or through code. Adding a condition will apply its secondary effects which are set to fire on the *added*-event, and removing them will apply effects which are set to fire on the *removed*-event (see Condition settings for more info about events). This can make for some powerful and interesting conditions, since you could make a damage-over-time condition (ticked damage effect) that should not be removed since that will trigger f.eg. massive damage, forcing the player to decide if it's better to let it run to its end to spread out the damage over time, or to remove it and apply lots of damage at once.

Conditions without casting abilities

Conditions can be added and removed outside of abilities.
You can add a condition to a target by calling `AbilityConditionsTarget.AddCondition(condition)`, and remove it by calling `AbilityConditionsTarget.RemoveCondition(condition)` on it's Ability Conditions Target-component.
This opens up a lot of possibilities since the world, equipment, actions etc. could add and remove conditions to a character. Below is an example of a small component that set a target to be in a "wet" state when entering an area where it rains.
Other systems in your game could react to these changes in character conditions (or interact with conditions directly) to create very interesting mechanics along with emergent gameplay.

```
        public class WetZone : MonoBehaviour
        {
                [SerializeField] private AbilityConditionSO
_wetCondition;

                private void OnTriggerEnter(Collider other)
                {
                    if (!other.TryGetComponent<AbilityConditionsTarget>
(out var target)) return;
                    target.AddCondition(_wetCondition,0, null);
                }

                private void OnTriggerExit(Collider other)
                {
                    if (!other.TryGetComponent<AbilityConditionsTarget>
(out var target)) return;
                    target.RemoveCondition(_wetCondition);
                }
        }
```

Conditional effects

An ability can have effects that are only applied when the target has or lacks certain conditions. Conditional effects are created as a separate object and can be reused between different abilities.
You can open the conditional effects window by going to *Tools->Spam Framework->Conditional effects or* Tools->Spam Framework->All Windows* and selecting the Conditional Effects tab.

Conditional effects can be though of as "effect groups", where f.eg. fire abilities could deal normal damage to targets, but also set them on fire if they're affected by a "flammable" condition, or have an ability heal the caster if the caster has the condition "vampire" and the target has the condition "living".

Since the underlying object is reusable, it can easily be attached to multiple abilities, making them behave similarly on targets that satisfies the given pre-conditions. You can be very creative and abilities can have a large range of effects given different circumstances with conditional effects.

General settings

Name
The name of the conditional effect.

Pre-conditions

A list of pre-conditions that needs to be satisfied for the effects to be applied.

When you add and remove pre-conditions, a written form of the full criteria for the conditional effect will be displayed under it's name at the top, f.eg "If target has [reflect] apply [minor damage] to caster".

Effects

The effects that are applied if the pre-conditions are met. These can be applied to either the target or the caster. These are called `Secondary Effects` in code.

Ability pre-conditions

Pre-conditions are condition-rules thats needs to be satisfied for an ability's effects to apply. The rules are checked before the ability is cast or when a projectile hits something. If pre-conditions aren't satisfied then **no** effects (including conditional effects) will be applied.
If you wish to check if pre-conditions are satisfied eariler than that (f.eg. you don't want the caster to be able to cast a certain projectile if pre-conditions aren't met) you'll have to manually call `Ability.CanCastAbility(Vector3 target)` from another script.

To add pre-conditions to an ability, click "Add pre-condition".

The settings are, from left to right:

Constraint
Should the target have or lack the given condition?

Condition
Which conditions should be checked for?

On caster
Should the caster be checked for the given condition?

Ability Conditions Target

A component that enables a target to be affected by conditions. The target's immunities and active conditions will be shown at the bottom at runtime.

Max Conditions
The maximum number of conditions that can affect this target simultaneously. A fixed maximum prevents garbage from being generated at runtime which reduces stress on the garbage collector.
Set this as low as possible for your game to prevent memory from being needlessly allocated.

Time handling
Automatic: The component will decrease the lifetime of conditions in it's update loop.
Manual: The component won't decrease the lifetime of conditions. You have to manually call `TickLifetimes(deltaTime)` to decrease condition lifetimes.

Immunities

Immunites are conditions that won't affect the target. Depending in your game these can either be static or dynamic.
Immunities can be changed at runtime by calling `AddImmunity(condition)` and `RemoveImmunity(condition)` from another script.

Max Num Immunities
Just as *Max conditions* you have to set max immunities to pre-allocate memory for a target's immunities. This avoids garbage generation at runtime.

Valid conditions

A list of which conditions that can affect the target. Use these when only a few select conditions can affect this target. You can set these as opposed to adding a lot of immunities

Note that having both valid conditions and immunities simultaneously is not supported. Valid conditions are immutable at runtime. If you add Valid conditions, immunities will be disregarded.

Ability conditions VFX component

A component that enables Condition VFX to be display at the target. See also Conditions VFX Settings.

Spawn point override
Set this to a child-transform of the game object to override the given spawn-position settings on Condition VFX.

Condition VFX component

A component that's required for VFX (particle systems) that should display when a target is affected by a certain condition.

Keep rotation
Enabling this will force the VFX to keep its initial rotation, even if the target rotates.