

National focus modding - Hearts of Iron 4 Wiki

This is a community maintained wiki. If you spot a mistake then you are welcome to fix it.

National focus trees are defined within `/Hearts of Iron IV/common/national_focus/*.txt` files. Like most other files, the filename is irrelevant and isn't used for anything other than organisation. One file can store more than one focus tree or none at all.



Focus tree[[编辑](#) | [编辑源代码](#)]

A focus tree is defined by using a `focus_tree = { ... }` block. The following arguments are used:

`id = my_focus_tree` decides the ID that the focus tree uses. This ID is never seen by the player in-game, used merely for the [has_focus_tree_trigger](#) and the [load_focus_tree_effect](#). However, it is still mandatory to define, and an overlap will result in an error.

`country = { ... }` is a [MTTH block](#) that assigns a score for the focus tree, deciding which one is used in-game. This is evaluated *before the game's start* and the check is essentially **never refreshed**^[a]. The focus tree with the highest score will be the one that gets loaded for the country. By default, the score starts with 1. A typical usage looks like this:

```
country = {
    factor = 0
    modifier = {
        add = 20
        original_tag = TRA
    }
}
```

In this case, countries originating from  [Transylvania](#) (i.e. the country itself and any civil war or collaboration government breakaways, as ensured by [the original_tag_trigger](#)) will have the score of 20, while every other country will have the score of 0. Assuming that there is no other focus tree where  [Transylvania](#) has a higher country score, this will ensure that this focus tree gets loaded for it.

`default = yes` sets the focus tree to be marked as default. In total, *there should be one total default focus tree*, no more, no less. A focus tree being marked as default means that if every other focus tree has a country score of 0, this tree will be chosen instead. Additionally, a country starting with a focus tree will fail to appear in the "minor countries" section within the "interesting countries" menu before the game's start. If this is left out from a focus tree, it gets assumed to be non-default.

`reset_on_civilwar = no` is not determined on its effect. Instead, this is how focus trees are handled in civil wars, regardless of if `reset_on_civilwar` is set or how it's set:

When a civil war starts, the original country will always continue using the focus tree. The focus it's doing will not be paused or cancelled by the civil war itself. The revolter will have the focus tree it's using evaluated when the civil war starts, assigning one depending on each tree's `country = { ... }` value. If the same focus tree gets used for the revolting country as the one that the original country used when the civil war started, every focus that the original country has completed will get completed for the revolting country, including setting the same focus progress for the one that's being completed by the original country at the moment. Otherwise, the focus progress will get lost.

`shared_focus = TAG_focusname` will set the focus tree to include the specified [shared focus](#) and every focus that is connected to it via prerequisites. **Setting this to a non-existing focus causes a game crash when loading into the main menu.**

`continuous_focus_position = { x = 1200 y = 100 }` is the position of the top left corner of the continuous focus menu *in pixels*. For comparison, by default^[b], the continuous focus palette has the position of 50 on the X axis and 1000 on the Y axis. If both x and y are set to 0 or the position is undefined for the tree, it resets to the default position.

`initial_show_position = { ... }` decides the initial position of the camera when the focus tree is first opened. There are 2 ways to arrange it:

`focus = TAG_focusname` will make the camera centre on the specified focus in particular. It'll be in the top centre of the screen exactly, taking offsets into consideration.

`x = 12 y = 0` decides the exact position of the top-centre of the camera. This uses the same coordinate system as regular focuses do, by default a unit of x being equal to 96 pixels and a unit of y being equal to 130 pixels^[c]

This also accepts `offset = { ... }`, adding the specified values to respective positions if the conditions within the `trigger = { ... }` trigger block are met for the country. For example, this will apply the modifier and result in a position of x = 13, y = 1 if the country is BHR:

```
initial_show_position = {
    x = 17
    y = 0
    offset = {
        x = -4
        y = 1
        trigger = {
            tag = BHR
        }
    }
}
```

`focus = { ... }` are the focuses themselves. Each focus that's put within the focus tree will. Of course, the focuses **have** to be within a `focus_tree = { ... }` in order to let the game know which focus tree exactly to assign them to. If a `focus = { ... }` blocks ends up outside of a `focus_tree = { ... }` or within another `focus = { ... }`, this gets marked within the [error log](#) as "focus" being an unexpected token, fixed by adjusting brackets as needed.

Examples[[编辑](#) | [编辑源代码](#)]

Pure minimum, with 2 focuses:

展开The text in this section has been collapsed by default.

Average tree, with 2 focuses:

展开The text in this section has been collapsed by default.

Focuses[[编辑](#) | [编辑源代码](#)]

The ID for the focus is defined using `id = TAG_focusname`. While it's optional to preface the focus' ID with the country's tag, doing so is preferred to avoid overlapping focus IDs from different focus trees. Having the same focus ID in different focus trees leads to errors, such as broken prerequisite lines and effects or triggers such as [complete_national_focus](#) or [has_completed_focus](#) not working correctly. Instead, it could be possible to use [shared focuses](#) to put the same focus in different focus trees.

Name and description[[编辑](#) | [编辑源代码](#)]

The name of the focus depending on the language that's turned on is defined within `/Hearts of Iron IV/localisation/`, using the ID of the focus as the localisation key. For English in particular, this is defined within any `/Hearts of Iron IV/localisation/english/*_1_english.yml` file with the UTF-8-BOM encoding. It is preferable to use new localisation files when possible rather than overwriting base game localisation in order to not have to change that for compatibility with recent versions, and to do so, the file should have a new name that doesn't exist in base game, but it must still end with `_1_english.yml` to be loaded properly.

Within a localisation file, as long as the first line is `1_english:`, a localisation key gets its value assigned using `TAG_focusname: "Focus' name'`. To assign the description, the focus' ID gets used with `_desc` appended to the end as `TAG_focusname_desc: "Focus' description"`

Position[\[编辑 | 编辑源代码\]](#)

The position of the focus is decided via `x = 5` and `y = 1` attributes. By default, a unit of `x` is equal to 96 pixels and a unit of `y` is equal to 130 pixels^[c]. In other words, a focus directly below another focus would have a unit difference of 1, while a focus directly to the right of another one would have a unit difference of 2. By default, this is relative to the top left corner of the tree: a larger `x` value moves the focus right, a larger `y` value moves the focus *down*.

It is also possible **and preferred** (on focuses with prerequisites) to make the focus' position be relative to another focus with doing `relative_position_id = TAG_other_focus`. This will position the focus relative to that focus, adding the `x` and `y` values to the other focus' position (after calculating that one's `relative_position_id` too). Doing so allows for more flexibility in the focus tree design by allowing to easily modify the position of the entire branch at the same time, due to updates to the children focuses' positioning. This also allows to only use the later `offset = { ... }` only in the top focus of each branch that requires to be moved.

The game can behave unstably with an incorrect relative position ID. A recursion (such as a focus being positioned relative to itself or focus A and focus B being positioned relative to each other) may cause a game crash since it is impossible to determine the exact position of the focus, and the focus must also be located in the same focus tree for the argument to work properly.

Offsets in particular are done with `offset = { ... }`. The `x = 10` and `y = -3` values will be added to the focus' position if the conditions within `trigger = { ... }` are met for the country *when the focus tree is loaded*. This looks like the following:

```
offset = {
  x = -5
  trigger = {
    has_dlc = "Poland: United and Ready"
  }
}
```

This in particular will move the focus 5 units to the left if the "Poland: United and Ready" DLC is turned on. This check also can be refreshed mid-session with the [mark focus tree layout dirty effect](#)^[d], applying the offset if true.

Interaction with other focuses[\[编辑 | 编辑源代码\]](#)

`prerequisite = { focus = TAG_other_focus }` decides the focuses necessary to complete for this focus to be available. At least one focus within a prerequisite has to be completed to mark the prerequisite as true, and each prerequisite much be completed to take the focus. In other words, an OR statement is done by putting 2 focuses inside a prerequisite as `prerequisite = { focus = TAG_other_focus_1 focus = TAG_other_focus_2 }`, while an AND statement is done by putting two different prerequisites like the following:

```
prerequisite = { focus = TAG_other_focus_1 }
prerequisite = { focus = TAG_other_focus_2 }
```

This system cannot represent every boolean logical arrangement, such as $A \text{ or } (B \text{ and } C)$ (Where A , B , and C represent whether a focus is complete) or with anything using negation. In this case, it can be possible to, instead, put an OR statement for either of the focuses necessary to complete this one and use the [has completed focus](#) trigger within the `available = { ... }` block with necessary flow control tools. [A custom trigger tooltip can be used to make it easier for the player to understand](#).

`mutually_exclusive = { focus = TAG_other_focus }` makes this focus impossible to select if the specified focus has been completed. If both focuses are mutually exclusive toward each other, then the mutually exclusive arrows will be shown in the focus tree view. Mutual exclusivity to multiple focuses is usually done by putting several of `focus = TAG_focusname` in the same `mutually_exclusive`, but defining several of `mutually_exclusive` is also possible.

Neither prerequisites nor mutual exclusivity require the other focus to be in the same focus tree. This means that it can be used with [shared focuses](#) to declare a regular, non-shared focus as mutually exclusive or a prerequisite without any errors even when used in a focus tree not containing that focus.

The difference between prerequisites and using [has completed focus](#) within the `available = { ... }` block is that the prerequisites show up as lines within the national focus tree view. There is an issue that leads to prerequisite lines not working properly: duplicate focus IDs within different focus trees. In this case, the game can take the position of the focuses as the ones within the different focus tree that contains the same focuses, leading to them appearing to link towards empty spaces or start inside of them. In order to avoid this, duplicate focus IDs must be avoided. A simple way to decrease the chance drastically is to preface the focus IDs with the country tag (such as `TAG_focus_name`) or something else that's unique for the focus tree (Such as `REGION_focus_name` for a shared regional tree). If the same focus tree branch should be used within several different focus trees, then [shared focuses](#) can achieve exactly that.

Icon[\[编辑 | 编辑源代码\]](#)

折叠General sprite overview

For loading GFX, the game uses the sprite system. Sprites are code definitions that attach a name to an image file, as well as optionally adding additional information, such as animation, the amount of frames, the way that the image will be loaded, and so on. This means **placing an image into the gfx folder isn't enough for it to work**, a sprite has to use that image file as well. Sprites are defined in any `/Hearts of Iron IV/interface/*.gfx` file (this is separate from `gfx/interface/`), opened with a text editor. To create a new `.gfx` file, a text file can be created and renamed to change the extension (on Windows, the [Windows Explorer needs to show the extensions, which it doesn't by default](#)). In particular, sprites are defined within a `spriteTypes = { ... }` block, as to separate from fonts and map arrows also defined in that folder, while the simplest sprite with the least mandatory properties is a `spriteType = { ... }`. The simplest sprite definition looks like the following:

```
spriteTypes = {
  spriteType = {
    name = GFX_first_sprite # In some cases, beginning with GFX_ is mandatory for it to work.
    texturefile = gfx/interface/folder/filename.dds # The folder and filename don't matter, as long as they are correct
  } # Only the forward slash '/' (can be doubled as '//') can be used to separate folders.
  spriteType = { # The image doesn't have to be .dds, as .tga and .png are acceptable.
    name = GFX_second_sprite
    texturefile = gfx/interface/folder2/filename2.dds
    noOfFrames = 2
  }
}
```

In this case, this creates a sprite with the name of `GFX_first_sprite` and attaches the `/Hearts of Iron IV/gfx/interface/folder/filename.dds` image to it, and a second sprite similarly. The second sprite will be split into 2 frames: this is decided by having the left half of the image as the first frame and the right half as the second frame (more frames would further split the image horizontally). This doesn't make the sprite animated, just turns on the option to switch between the two halves as needed. `GFX_second_sprite:1` serves as a reference to the first frame, and GUI can be set up to change the shown frame depending on context, such as with radio stations. In order to add animation, a [frameAnimatedSpriteType](#) is used.


It's never mandatory to copy a base game file to change a sprite. If there are duplicate definitions of a sprite with the same name in different files, the game will prioritise the one that would be [evaluated later, based on the filename](#), and the older sprite will be ignored in entirety. This can be ensured by beginning the replacement file's name with a symbol late in the ASCII character table. Typically the lowercase letter 'z' is used for this purpose. For example, to change the amount of frames in `GFX_idea_traits_strip` to 10, it is possible to define a sprite with that name with 10 frames in the mod's `modname/interface/zz_replace.gfx` file instead of copying over the base game file.

Since most `.gfx` files define integral parts of the user interface, copying them over can lead to the mod's loaded files missing sprites upon a major game update, which would appear in-game as the default image, which is the error dog by default. As to ease the burden of needing to check the interface files, it's best to never copy over `.gfx` files, unless more additions would be actively harmful to the mod, such as with `interface/subuniticons.gfx`

Within a focus, an icon is assigned with the line of `icon = GFX_focus_icon`. This assigns two sprites to the focus, in particular:

Regular sprite, with the name of `GFX_focus_icon`. This is used in the focus description view and in the focus tree view when the focus is unavailable, is being completed, or has been completed.

Sprite for the shine animation, with the name of `GFX_focus_icon_shine` (**Same name as the regular sprite, but with `_shine` at the end**). This is used in the focus tree view for focuses that are currently available and in the country politics and diplomacy views for the focus currently being complete.

If one of these is undefined or is defined incorrectly, the  missing focus icon will be used instead of the appropriate sprite, however the working sprite will continue to be used. Commonly, this is caused by the shine not being defined correctly, such as due to the name not ending with `_shine`.

If the texturefile links to a non-existing file, whether it's the folder path that's incorrect^{[[el](#)]} or the filename, including the extension, the focus icon will appear as fully transparent.

By default, the base game stores images for focus icons in the `/Hearts of Iron IV/gfx/interface/goals/` folder and sprites in the `interface/goals.gfx` and `interface/goals_shine.gfx` files. Since there's no reason to copy the files to the mod and doing so will lead to needing to update the file after a major update to use new sprites, it's best to create a new file in the folder for sprite definitions.

The following is an example of an interface file that defines both of the sprites:



展开

Collapsed example interface file

When copying from the template, note to change the `animationmaskfile` in each animation within the sprite with the shine alongside the `texturefile` and name.

Triggers^{[[编辑](#) | [编辑源代码](#)]}

See also: [Triggers](#)


In order to take the focus, aside from the focus prerequisites, the conditions within the `available = { ... }` block must be met. This functions as an [AND block](#), so each of the triggers must be true to fulfilled. [Scopes](#) can be used to check for conditions for other countries or within states. By default, the scope is of the country doing the focus. For example, this example requires the country to have more than 10%  [Stability](#) and for the state 294 to be owned by the  [Republic of Qatar](#):

```
available = {
    stability > 0.1
    294 = { is_owned_by = QAT }
}
```

`bypass = { ... }` is similar, but for bypassing the focus. Bypassing a focus marks the focus as complete, but does not grant its effects within the completion reward. The exact same applies as to `available`: it's an AND block that assumes the country doing the focus by default. `bypass_if_unavailable = yes` can be used to make the focus automatically bypass as soon as the `available = { ... }` block is not met without needing to port over the triggers.

`allow_branch = { ... }` is used to tell when the focus should be visible. **This is only checked when the focus tree is first loaded.** However, this check also can be refreshed mid-session with the [mark focus tree layout dirty effect](#)^{[[d\]](#)], making the focus visible or invisible depending on if it's true or not. By default, a focus will also be disallowed if either of the parent focuses (as set by prerequisites) is disallowed. However, a focus containing `allow_branch` within its definition will only check its own allowed status, still showing up if it has disallowed parents. This means that if a branch is set to be disallowed under certain conditions, the first focus in any sub-branch that has its own conditions for being allowed must also contain the parent branch's condition within of itself, while other focuses may not have any `allow_branch` defined at all due to it being inherited from parents.}

`available_if_capitulated = yes` sets the focus to be possible to complete while being capitulated. By default, this is set to false.



`cancel_if_invalid = no` and `continue_if_invalid = yes` decide how to treat the focus if the `available = { ... }` block becomes false while doing it. By default, these are true and false respectively. If both are set to false, the focus would pause when the `available = { ... }` block is false. This will not remove the gain `focus` [static modifier](#), which by default results in costing 1  [Political Power](#) per day when doing the focus.

`cancel = { ... }` decides additional conditions which would cancel the focus if met. This is usually paired with `cancel_if_invalid = no`.

`historical_ai = { ... }` decides when the AI is able to pick this focus with the historical focus turned on. This does not ensure it picks this focus, rather prevents it from picking it when false. This takes priority over the order of focuses granted within [AI strategy plans](#): if the AI were to do this focus next by the plan, yet `historical_ai = { ... }` is false and historical focus is turned on, then it won't be able to.

Effects^{[[编辑](#) | [编辑源代码](#)]}

See also: [Effects](#)

The primary reward of the focus is done with `completion_reward = { ... }`. This executes each [effect](#) within the specified [scopes](#) in order that they are put in the file. The assumed scope is the country doing the focus. For example, this would add 100  [Political Power](#) to the country doing the focus and fire the `my_event.0` country event to  [Oman](#):

```
completion_reward = {
    add_political_power = 100
    OMA = { country_event = my_event.0 }
}
```

The tooltip of the focus can be changed with `complete_tooltip = { ... }`, which is also an *effect block*. This would be equivalent to putting the contents of the reward inside of [hidden effect](#) and using [effect tooltip](#) in the same reward. This can be useful if the tooltip of the reward appears cluttered. For instance, using [random owned controlled state](#) thrice with the same effect in each one can result in a cluttered tooltip, as each state and its effects would appear individually. But if, instead, the code [sets a state flag](#) for each state in the reward and, in the tooltip, uses [every state](#) limited to the ones that have the state flag, it'll show the same effect being executed for 3 states at the same time, cutting it into a third of what it was.

Additionally, `select_effect = { ... }` is used to execute an effect when the focus is *selected*. This has no tooltip shown to the player. This also automatically makes the focus impossible to cancel manually. It may still be cancelled automatically if set to do so, so in most cases it'd be preferable to set it to not cancel if invalid in order to prevent the effects within from firing more than once, as there is no way to execute an effect when the focus gets cancelled.

The focus is not yet marked as complete when the effects are being executed. What this means is that any [has completed focus check](#) ran within will return as false. This can be a hinderance in some cases, most commonly when using `allow_branch` (as the effect to refresh the check will not work properly when put within a focus). Some alternatives can be considered:

[Country flags to track that a focus has been completed](#). Since country flags are set immediately, as long as the setting precedes the requirement, this will work.

Delaying the effects that would require the focus to be complete. This is usually done by firing a hidden event. If done with no delay, the event will be fired *right after* the focus is complete, resulting in no noticeable delay for the player, but still executing the effects in order.

Forcefully marking the focus to be complete. This, for example, can be done by using [load focus tree](#) for the same focus tree with the completed focuses being kept. Doing so will not interrupt the completion reward's execution.

Search filters^{[[编辑](#) | [编辑源代码](#)]}

Search filters for each focus are set with `search_filters = { ... }`. Within that block, each search filter that the focus has is put inside, separated by whitespace characters. For instance, the following will assign `FOCUS_FILTER_MANPOWER` and `FOCUS_FILTER_POLITICAL` to the focus: `search_filters = { FOCUS_FILTER_MANPOWER FOCUS_FILTER_POLITICAL }`. These filters get used in the search menu in the top right of the national focus tree view.

A focus filter is not defined in any file, but instead they are created dynamically for each focus tree. Each one uses the sprite the same as its name but with `GFX_` inserted in the beginning. For instance, this definition within any `/Hearts of Iron IV/interface/*.gfx` file would be used for `FOCUS_FILTER_MY_MOD`:

```
spriteType = {
    name = GFX_FOCUS_FILTER_MY_MOD
    texturefile = gfx/interface/focusview/filter/my_mod_icon.dds
}
```

As before with regular focus icon, the exact folder where filter icons are stored is irrelevant, as long as the texturefile specified within the sprite is correct.

The localisation key used for the focus filter is the same as its name. For example, with the prior example of FOCUS_FILTER_MY_MOD, this would get defined for the English language in any /Hearts of Iron IV/localisation/english/*_1_english.yml file as FOCUS_FILTER_MY_MOD: "My mod"

A list
of
every
base
game
focus
filter:
展开

In addition, it is possible to set the priority of focus filters. This is done with the `search_filter_prios = { ... }` block *outside of the focus_tree = { ... }*. This priority is impossible to set to be focus tree-specific and instead is global. An entry within that block is done in the format of `FOCUS_FILTER_MY_MOD = 1200`, where the first part decides the filter and the second part decides the priority. Focus filters with the higher priority appear earlier in the top view. By default, the game does it in the file containing the generic focus tree: /Hearts of Iron IV/common/national_focus/generic.txt.

AI will do[\[编辑 | 编辑源代码\]](#)

`ai_will_do = { ... }` is a **MTTH block** that decides the likelihood for the AI to do this focus if [an AI strategy plan](#) is not set.

By default, each focus has a score of 1. The arguments of `base` (changing the value), `add`, and `factor` (multiplying it) can be used to modify it.

Within the `ai_will_do = { ... }` block, `modifier = { ... }` functions as a trigger block where the prior three value-modifying arguments are also supported. The value will be modified if the triggers are true. For example, the following will result in the value of 15 for POL and a value of 5 for every other country:

```
ai_will_do = {
    base = 5
    modifier = {
        factor = 3
        tag = POL
    }
}
```

An arbitrarily large amount of modifiers is possible to add to an `ai_will_do`, and they will apply in the order they're put in the code. It is also possible to use [variables](#) within a modifier of the `ai_will_do` value.

The way that the value is evaluated for AI picking the focus is that, when picking a focus to do, it generates a random decimal value between 0 and the `ai_will_do` value for each of the focuses. If the evaluated focus has a prerequisite focus that the AI has just completed, the generated `ai_will_do` value gets multiplied by 1.5 before the game picks a focus.^[1] If the [AI strategy plan](#) that the country is currently following has `focus_factors = { ... }` defined for this focus, the value gets multiplied by the specified value. Afterwards, the game picks the focus that has the highest generated value. If neither of the focuses has a value above 0, the AI will not pick any of them, instead going into continuous focuses if possible or not doing any otherwise.

Due to that algorithm, low values are less likely to be picked than intuition suggests.

To reiterate, **this is only evaluated if the [AI strategy plan](#) for the country doesn't have the order of the focuses set** or if none of the focuses in that order can be followed. Comparing the chances between focuses is the following:

Formulas for chance calculation[\[编辑 | 编辑源代码\]](#)

As the general formula can be complex to calculate without a special tool, simpler calculations for special cases can provide quite useful: whether for approximating a chance by substituting similar numbers or for calculating the exact chance if the numbers align. Each case will be provided with three paragraphs — The formula in the first paragraph, an example in the second paragraph, and a general explanation of why it applies (though not necessarily a rigorous proof) in the third paragraph. The total chance will be given on the scale of 0–1; focuses are assumed each to have a positive value, as negatives are unintended and a focus with a chance of zero will never get picked by the game's AI meaning they can be excluded from the calculation entirely; and the modifiers applying to the focus' AI will do value (e.g. from AI strategy plans or the bonus for continuing the same branch) have already been applied, as multiplying the result of the rolled dice by a number would be the same as multiplying the ends of a range by that number.

In shortened form, the `ai_will_do` value of a focus will be referred simply as the "focus' value".

展开One focus with a large value and other focuses with the same smaller value

展开Focuses split across two different values

展开General formula

Other[\[编辑 | 编辑源代码\]](#)

Another important aspect of the focus is `cost = 8`. This sets how long the focus takes to complete. By default, a cost of 1 is taken to be 7 "points"^[2], of which by default 1 is completed daily, although it's possible to set different speeds depending if the country is at war or at peace^[3]. In other words, a cost of 1 represents a week by default. Decimals within cost are supported, and it will get rounded down to a whole day in the game.

`will_lead_to_war_with = TAG` is used to mark the focus as leading to a war towards the specified country. This will show up for the specified country and its allies (subjects, overlord, and/or fellow faction members) as the country doing the focus justifying a wargoal on them in the alerts topbar. This also makes the AI prepare for a potential war, both for the country doing the focus and for the country on whom the focus is set to declare war on.

`cancelable = no` will set the focus to be impossible to cancel manually without defining a `select_effect`. This does not prevent the focus from cancelling automatically. However, `cancelable = yes` *does not* make the focus possible to manually cancel if the focus also contains `select_effect = { ... }`.

`dynamic = yes` allows dynamic localisation ([which includes namespaces and scripted localisation](#)) to update within the focus' *title*. If false (including the default value) — a value is generated at the game's start, which remains unchanging until the next reload (either of the savefile or the focus tree). A focus' description always updates localisation dynamically, regardless if this is present or not.

Examples[\[编辑 | 编辑源代码\]](#)

Pure minimum (at the top of the branch):

展开The text in this section has been collapsed by default.

Average focuses:

展开The text in this section has been collapsed by default.

展开The text in this section has been collapsed by default.

An allow_branch focus:

展开The text in this section has been collapsed by default.

Shared focuses[[编辑](#) | [编辑源代码](#)]

Shared focuses are those that can be used in multiple focus trees. Examples of these include the ["Invite foreign investors" branch](#) present in focus trees for  [China](#),  [Communist China](#), and  [Manchukuo](#) or the ["Rejoin the railways"](#) and ["Restore the worker's republic"](#) national focus tree branches for  [Estonia](#),  [Latvia](#), and  [Lithuania](#).

A shared focus is defined with a `shared_focus = { ... }` block, **not** inside of any `focus_tree = { ... }` block. Within that block, the focus is defined in *exactly* the same way as a regular focus within a focus tree: each of the arguments that can be used in a regular `focus = { ... }` can be used with a `shared_focus = { ... }` without any changes.

A shared focus is added to a focus tree by adding the `shared_focus = my_shared_focus` argument within that focus tree. This will add the specified shared focus **and every shared focus that's connected to it via prerequisites**, assuming that `allow_branch` is true. Even if the shared focus has a different focus as a prerequisite that's not loaded in this focus tree, it will appear as visible. **Setting this to a non-existing focus causes a game crash when loading into the main menu.**

In addition, the [load_focus_tree effect](#) can be used with kept completed focuses as such in order to have shared focuses be kept as complete when switching to a different tree:

```
load_focus_tree = {
    tree = my_focus_tree
    keep_completed = yes
}
```

This can serve as an alternative to using `allow_branch = { ... }` in order to swap out focus tree branches for the same country from being visible or not.

Shared focuses can be defined in any `/Hearts of Iron IV/common/national_focus/*.txt` file. Usually they're kept in different files from the focus trees using them, but that's not necessary.

Example[[编辑](#) | [编辑源代码](#)]

展开The text in this section has been collapsed by default.

Continuous focuses[[编辑](#) | [编辑源代码](#)]

Continuous focuses are defined in `/Hearts of Iron IV/common/continuous_focus/*.txt` files. These are the focuses that can be selected after having completed 10 national focuses^[4] and which last eternally without being able to complete them entirely.

Focus palettes[[编辑](#) | [编辑源代码](#)]

A continuous focus palette is incredibly similar to a definition to a national focus tree, but with less arguments.

`id = my_focus_palette` decides the ID of the focus palette. Mandatory.

`country = { ... }` is a [MTTH block](#) that assigns scores to each country for picking a continuous focus palette, in the exact same manner as it's done with national focus trees.

`default = yes` assigns a focus palette as default. As with national focus trees, only one palette should be default total: no more, no less. Defaults to false.

`reset_on_civilwar = no` similarly as with national focuses, has an unknown effect.

`position = { x = 100 y = 1230 }` assigns the default position of the continuous focus palette, measuring in the pixel position of the top left corner. This will get used if the national focus tree of the country doesn't specify a different position that is not `x = 0 y = 0`

`focus = { ... }` are the continuous focuses themselves.

Example:

展开The text in this section has been collapsed by default.

Focuses[[编辑](#) | [编辑源代码](#)]

Continuous focuses are also fairly similar to national focuses in definition, but there are substantial differences.

`id = TAG_focus_name` is the continuous focus' ID. There must be no overlap across different palettes. This also gets used as the localisation key for the focus' name and appending `_desc` is used to get the focus' description.

`icon = GFX_focus_icon_name` is the sprite of the icon used by the continuous focus. Similarly to national focus, there must be both a regular icon and a shine for it to always work properly.

`available = { ... }` is a trigger block required to be met for the focus to be *visible*, unlike national focuses where this makes it possible to pick. Instead, the trigger block used for making the focus be possible to pick is `enable = { ... }`. `available_if_capitulated = yes` will also make the focus possible to select while capitulated. If false, the focus will still be visible, but can't be picked.

`modifier = { ... }` is a modifier block that details the list of [modifiers](#) and their values that are added by having the focus selected. However, there are other ways to apply continuous bonuses aside from modifiers, such as research bonuses for specific categories and equipment bonuses. For these, `idea = idea_name` can be used, which'll add the idea to the country when the focus is selected and remove it when the focus is cancelled. This is recommended to do with hidden ideas, as it shows the effects of the idea when hovering over the focus.

`select_effect = { ... }` and `cancel_effect = { ... }` are effects executed when selecting and unselecting this focus respectively. Since it's possible to select and unselect the focus at any time with no cost, it's recommended to make these be completely the opposite of each other so that doing so will not grant the player any benefit.

`daily_cost = 0.4` is the cost in daily political power gain to take this focus compared to not having any focus selected, can be decimal.

`ai_will_do = { ... }` is a [MTTH block](#), working in the exact same way as [ai_will_do in national focuses](#). Additionally, `supports_ai_strategy = AI_focus` makes the focus be possible to pick by AI if [it is following the specified focus](#). Without that, AI would never pick this continuous focus.

Examples:

展开The text in this section has been collapsed by default.

展开The text in this section has been collapsed by default.

AI strategy plans[[编辑](#) | [编辑源代码](#)]

Main article: [AI modding#AI Strategy Plans](#)

AI strategy plans to tell AI what to prioritise depending on circumstances: which advisors to pick, which technologies to research, which AI strategies to apply, and, most importantly here, which

focuses to pick. This is a short overview of AI strategy plans purely for national focus prioritising, full detail being in the [AI modding](#) article. Several AI strategy plans can be enabled at the same time.

AI strategy plans are defined within /Hearts of Iron IV/common/ai_strategy_plans/*.txt files. Within these files, a new strategy plan is done as a new block, the name of which must be the same as the internal ID of the plan.

Within that plan, `name = "AI plan's name"` and `desc = "AI plan's description"` decide the name and the description of the strategy plan. *This is never intended to be shown to the player*, so localising it into different languages is never needed. Instead, this is used within the `aiview` console command, which tells info to the developer about what AI wants to prioritise.

`allowed = { ... }` is, similarly to decisions or ideas, is a trigger block **checked only at the game's start**. This is primarily used to tell which country and DLCs to restrict the strategy plan to.

`enable = { ... }` is checked each day if the allowed is met. If `enable = { ... }` is met, then the AI strategy plan will be assigned to the AI regardless of whether `enable = { ... }` turns false later or not. Commonly, [Triggers#has_game_rule](#) is used to make it work with custom game rules deciding what path the AI will pick. [is_historical_focus_on](#) is commonly used with the default AI game rule, and country flags can be used for randomisation, by setting up an [on_startup](#) to set a random one using [random_list](#)

`abort = { ... }` is checked every day in order to make the AI *stop* using this AI strategy plan if `enable = { ... }` is met. Additionally, it must be false in order for the AI strategy plan to be possible to be picked.

`ai_national_focuses = { TAG_focus_name_1 TAG_focus_name_2 }` is a list of national focuses, separated by whitespaces, in the order that the AI should take them. In this example, the AI will try to take TAG_focus_name_1 first if possible. If it's already taken or TAG_focus_name_1 is impossible to take, then AI will try to take TAG_focus_name_2. If both of the focuses are impossible to take due to being completed or unavailable, then it will move on to other focuses, taking `ai_will_do = { ... }` into consideration. While following a focus order, it ignores `ai_will_do = { ... }` values.

`focus_factors = { ... }` assigns a multiplier to `ai_will_do` values of the specified focus. An entry in this block looks like `TAG_focus_name = 3`. In this case, this will make the `ai_will_do` value of the focus be multiplied by 3, assuming AI strategy plan's weight of 1. If the focus has an `ai_will_do` value of 4 after applying modifiers, it'll become 12 if AI is following this strategy plan, and get treated as such. And, of course, a factor of 0 will make the focus be never picked without specification in `ai_national_focuses`. This can serve as a faster-to-write or a more randomised way to make AI follow a political path by making focuses it should never pick have a value of 0.

`weight = { ... }` is a [MTTH block](#) assigning an overall weight to the plan. Defined in the same way as [a national focus's ai_will_do](#), this multiplies each factor within the AI strategy plan by the weight before applying them. A weight of 1.25 will turn a focus factor of 4 into 5 before applying it, for instance. This can be used to make the AI follow the strategy plan more strictly in some cases and less strictly in others.

Example[\[编辑\]](#) [\[编辑源代码\]](#)

展开The text in this section has been collapsed by default.

Notes[\[编辑\]](#) [\[编辑源代码\]](#)

^ a: Dynamic countries, when created, will go through the check assigning a focus tree again. As well as that, reloading focuses by saving over a focus tree file while the debug mode is turned on via launch settings will also refresh this check.

^ b: The default position is defined within the pallette's definition in a /Hearts of Iron IV/common/continuous_focus/*.txt file

^ c: The exact size of a single x and y coordinate unit uses the `focus_spacing` positionType within /Hearts of Iron IV/interface/nationalfocusview.gui

^ d: If [mark_focus_tree_layout_dirty](#) is put within a focus reward, it wouldn't be marked as complete for the [has_completed_focus](#) trigger at the time it's executed. This can be avoided by using a hidden event, fired immediately, which has the effect to refresh the check within its immediate. Alternatively, the [load_focus_tree](#) effect, set to have the focuses kept complete, can be used to mark the focus as complete before doing the refreshing.

^ e: Within folder paths, a backslash (\) can result in the game not being able to read the folder, since it's expected to be used as an [escape character](#). Instead, it's preferable to use forward slashes, as in `texturefile = gfx/interface/goals/my_file.dds`.

References[\[编辑\]](#) [\[编辑源代码\]](#)

↑ NDefines.NAI.FOCUS_TREE_CONTINUE_FACTOR = 1.5 in [Defines](#)

↑ NDefines.NFocus.FOCUS_POINT_DAIS = 7 in [Defines](#)

↑ NDefines.NFocus.FOCUS_PROGRESS_PEACE = 1 and NDefines.NFocus.FOCUS_PROGRESS_WAR = 1 in [Defines](#)

↑ NDefines.NCountry.MIN_FOCUSES_FOR_CONTINUOUS = 10

Note that while modifying defines to use a define override file rather than copying over the entire file, as even otherwise 'minor' updates can add new defines causing potential crashes.