Prototyping Proposal: Migrating `base::Feature`-Based Feature Definitions from C++ to JSON5

blee@igalia.com, Feb. 19th 2026

Note: The direction shifted from deep-diving into `base::Feature` to exploring broadly across configuration systems. The following summarizes the progress made before this decision.

Characteristics of `base::Feature`

- **Immutable at runtime**: State is determined at process startup and cannot change during the process lifetime, unlike prefs or RuntimeEnabledFeatures which can be toggled dynamically.
- **Process-wide**: A single value applies to the entire process, unlike origin trials or permissions which are per-context.
- **Controlled via field trials**: Finch can remotely enable/disable features and segment users into experiment groups for metrics, without shipping a new binary.
- **Declarative**: Defined statically in source code with a default state, unlike prefs which require registration, storage, and UI.
- **Cross-process**: Accessible from any process (browser, renderer, GPU), unlike RuntimeEnabledFeatures which only exist in the renderer.

Current Status

- Chrome (`chrome/common/chrome_features`): 284 features
- Content (`content/public/common/content_features`): 191 features
- Generator, templates, and build integration working for both modules
- PoC CL link: https://crrev.com/c/7638000

Variations in Feature Definitions

Refer to Sections 2.1, 2.2, and 2.3 for details.

Migration Considerations

- **Scale**: A large number of `base::Feature` definitions are spread across many modules throughout the codebase, making incremental migration a significant effort.
- **Comments**: Original comments carry important context: design docs, bug references, platform rationale, kill-switch notes. Pre-feature comment blocks must be fully transferred.
- **Condition merging**: Avoid merging separate conditions (e.g., `IS_WIN || IS_MAC`) when the original code has distinct branches with different comments or rationale. Keep them as separate keys in the status dict, with inline comments placed on the specific condition they belong to.
- **Conflicts with upstream commits**: Build flags may be added or removed between migration and landing, requiring features to be updated accordingly.

---

1. Overview

Bug 469041913 proposed consolidating the various feature-controlling systems in the Chromium project (Runtime Enabled Features, `base::Feature`, Command Line Switches, Prefs, etc.) into a single set of JSON files for centralized feature control.

Given the large number of features in Chromium and that each feature-controlling system has its own purpose, requirements, and design that introduce different kinds of variations, unifying them into a single framework will be a very large task.

To gather initial insights, such as what the baseline should be, what kinds of variations we need to handle, and which can be simplified or eliminated, we propose prototyping with a narrower scope as the first step: migrating `base::Feature`-based feature definitions from C++ to JSON5. While the ultimate goal is to unify all feature-controlling systems into a single framework, starting with a narrower scope allows us to validate the approach and identify potential challenges early.

As the feature definitions in the C++ files of `chrome_features` (`chrome/common/chrome_features.h/.cc`) and `content_features` (`content/public/common/content_features.h/.cc`) vary in their patterns, and the files contain enough variations to serve as a meaningful starting point, this prototyping identifies chrome/content feature variations first and defines a JSON5 schema to handle them.

Once we have the JSON5 schema and a script that generates the C++ files for the chrome/content features from JSON5 definitions, we plan to expand it to the other `base::Feature` definitions across the codebase to capture the full range of variations.

2. Variations in `chrome_features` and `content_features`

2.1. Basic Pattern of Feature Declaration and Definition

Every feature requires a `name` and a `status`. The feature status can be either `enabled` or `disabled`. In most cases, the C++ identifier of the feature is a camel-cased string with `k` prepended, derived from the feature name.

- `CryptographyComplianceCnsa` feature, defined with `k`-prefixed feature name `kCryptographyComplianceCnsa`, disabled by default:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kCryptographyComplianceCnsa);

// chrome_features.cc
BASE_FEATURE(kCryptographyComplianceCnsa, base::FEATURE_DISABLED_BY_DEFAULT);
```

- `ChromeStructuredMetrics` feature, defined with `k`-prefixed feature name `kChromeStructuredMetrics`, enabled by default:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kChromeStructuredMetrics);

// chrome_features.cc
BASE_FEATURE(kChromeStructuredMetrics, base::FEATURE_ENABLED_BY_DEFAULT);
```

2.2. Mismatch Between Feature Name and C++ Identifier

A feature may have a C++ identifier that doesn't match the feature name, usually for code readability.

- `AbusiveOriginNotificationPermissionRevocation` feature, defined with `kAbusiveNotificationPermissionRevocation` instead of the `k`-prefixed feature name, to use a shorter identifier:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kAbusiveNotificationPermissionRevocation);

// chrome_features.cc
BASE_FEATURE(kAbusiveNotificationPermissionRevocation,
             "AbusiveOriginNotificationPermissionRevocation",
             base::FEATURE_ENABLED_BY_DEFAULT);
```

2.3. Conditional Feature Definition with Preprocessor

2.3.1. Exclusive Feature

A feature can be defined exclusively when a specific preprocessor condition is met.

- `Borealis` feature, defined only for ChromeOS:

```
// chrome_features.h
#if BUILDFLAG(IS_CHROMEOS)
COMPONENT_EXPORT(CHROME_FEATURES) BASE_DECLARE_FEATURE(kBorealis);
#endif

// chrome_features.cc
#if BUILDFLAG(IS_CHROMEOS)
BASE_FEATURE(kBorealis, base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

2.3.2. Conditional Feature Status

A feature can have a different status depending on the build configuration.

- `DesktopPWAsPreventClose` feature, enabled for ChromeOS, otherwise disabled:

```
// chrome_features.cc
BASE_FEATURE(kDesktopPWAsPreventClose,
#if BUILDFLAG(IS_CHROMEOS)
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

In cases where each condition needs to be commented differently, there can be multiple preprocessor conditional branches for the same status.

- `WebAppSystemMediaControls` feature, enabled for Windows and macOS, otherwise disabled, with different comments for Windows and macOS:

```
// content_features.cc
BASE_FEATURE(kWebAppSystemMediaControls,
#if BUILDFLAG(IS_WIN)
             // Windows enabled since 124.
             base::FEATURE_ENABLED_BY_DEFAULT);
#elif BUILDFLAG(IS_MAC)
             // macOS enabled in 130. If a kill switch is needed, it should be
             // safe to only disable the failing platform (ie. macOS here).
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

2.3.3. Logical Expressions in Preprocessor Condition

The preprocessor condition can be a logical expression.

- `ChromeAppsDeprecation` feature, defined for Windows, macOS, and Linux:

```
// chrome_features.h
#if BUILDFLAG(IS_WIN) || BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kChromeAppsDeprecation);
```

- `PreinstalledWebAppInstallation` feature, defined for all platforms except Android:

```
// chrome_features.h
```

```
#if !BUILDFLAG(IS_ANDROID)
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kPreinstalledWebAppInstallation);
```

- `AudioServiceOutOfProcess` feature, enabled for Windows, macOS, and Linux, otherwise disabled:

```
// content_features.cc

BASE_FEATURE(kAudioServiceOutOfProcess,
#if BUILDFLAG(IS_WIN) || BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
             base::FEATURE_ENABLED_BY_DEFAULT
#else
             base::FEATURE_DISABLED_BY_DEFAULT
#endif
);
```

2.3.4. Exclusive Feature with Conditional Feature Status

A feature defined exclusively can also have a conditional status.

- `GlicGuestContentsVisibilityState` feature, defined only when glic is enabled, enabled for macOS and Linux, otherwise disabled:

```
// chrome_features.h
#if BUILDFLAG(ENABLE_GLIC)
...
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kGlicGuestContentsVisibilityState);

// chrome_features.cc
#if BUILDFLAG(ENABLE_GLIC)
...
BASE_FEATURE(kGlicGuestContentsVisibilityState,
#if BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

2.3.5. Non-BUILDFLAG Preprocessor Condition

In most cases, the preprocessor condition checks a BUILDFLAG status. But it is also allowed to use a non-BUILDFLAG status as well.

- `WebAssemblyTrapHandler` feature, enabled for Linux, ChromeOS, Windows, and macOS when ARCH_CPU_X86_64 is defined, or enabled for Linux, ChromeOS, and macOS when ARCH_CPU_ARM64 is defined, otherwise disabled:

```
// content_features.cc
BASE_FEATURE(kWebAssemblyTrapHandler,
#if ((BUILDFLAG(IS_LINUX) || BUILDFLAG(IS_CHROMEOS) || BUILDFLAG(IS_WIN) ||  \
      BUILDFLAG(IS_MAC)) &&                                                  \
     defined(ARCH_CPU_X86_64)) ||                                            \
    ((BUILDFLAG(IS_LINUX) || BUILDFLAG(IS_CHROMEOS) || BUILDFLAG(IS_MAC)) && \
      defined(ARCH_CPU_ARM64))
             base::FEATURE_ENABLED_BY_DEFAULT
#else
             base::FEATURE_DISABLED_BY_DEFAULT
#endif
);
```

2.4. Basic Pattern of Feature Parameter Declaration and Definition

A feature may have parameters defined with `base::FeatureParam`. Every parameter requires a name, a type, and a default value. The default value is used when the parameter is not set or contains an invalid value.

- The `IgnoreDuplicateNavs` feature has a boolean-type parameter `skip_ignore_renderer_initiated_navs` with a default value of `false`:

```
// content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE(kIgnoreDuplicateNavs);
...
CONTENT_EXPORT BASE_DECLARE_FEATURE_PARAM(bool,
                                          kSkipIgnoreRendererInitiatedNavs);

// content_features.cc
BASE_FEATURE(kIgnoreDuplicateNavs, base::FEATURE_DISABLED_BY_DEFAULT);
...
BASE_FEATURE_PARAM(bool,
                   kSkipIgnoreRendererInitiatedNavs,
                   &kIgnoreDuplicateNavs,
                   "skip_ignore_renderer_initiated_navs",
                   false);
```

2.5. Parameter Declaration and Definition Without Macros

A feature parameter can be defined using the macros `BASE_DECLARE_FEATURE_PARAM` and `BASE_FEATURE_PARAM`. But there are many feature parameter definitions without the macros, and the C++ specifier varies in some cases.

- The `glic-allowed-origins-override` parameter of the `GlicCSPConfig` feature is defined using `base::FeatureParam` directly with the `const` specifier:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kGlicAllowedOriginsOverride;

// chrome_features.cc
const base::FeatureParam<std::string> kGlicAllowedOriginsOverride{
    &kGlicCSPConfig, "glic-allowed-origins-override",
    "https://gemini.google.com https://www.google.com"};
```

- The `moderate_level` parameter of the `LinuxLowMemoryMonitor` feature is defined using `base::FeatureParam` directly with the `const` specifier for declaration and the `constexpr` specifier for definition:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<int> kLinuxLowMemoryMonitorModerateLevel;

// chrome_features.cc
constexpr base::FeatureParam<int> kLinuxLowMemoryMonitorModerateLevel{
    &kLinuxLowMemoryMonitor, "moderate_level", 50};
```

2.6. Mismatch Between Feature Parameter Name and C++ Identifier

In most cases, the C++ identifier of the feature parameter is a camel-cased string with `k`-prepended, derived from the parameter name.

- The `glic-side-panel-min-width` parameter of the `GlicMultiInstance` feature has the `k`-prefixed/camel-cased parameter name as its C++ identifier:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<int> kGlicSidePanelMinWidth;

// chrome_features.cc
const base::FeatureParam<int> kGlicSidePanelMinWidth{
    &kGlicMultiInstance, "glic-side-panel-min-width", 384};
```

A feature parameter may have a C++ identifier that doesn't match the parameter name, usually for code readability.

- The `allowed_tiers` parameter of the `GlicWebActuationSetting` feature is defined with `kGlicWebActuationAllowedTiers` instead of the `k`-prefixed/camel-cased parameter name, for clarity:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kGlicWebActuationAllowedTiers;

// chrome_features.cc
const base::FeatureParam<std::string> kGlicWebActuationAllowedTiers{
    &kGlicWebActuationSetting, "allowed_tiers", ""};
```

2.7. Conditional Parameter Default Value with Preprocessor

A feature parameter can have a different default value depending on the build configuration. The preprocessor condition can be a logical expression, similar to the conditional feature definitions.

- The `GlicUserStatusCheck` feature has a `std::string`-type parameter `glic-user-status-url`. The default value of the parameter is `https://geminiweb-pa.googleapis.com/v1/glicStatus` for Google Chrome branding, otherwise an empty string:

```
// chrome_features.cc
BASE_FEATURE_PARAM(std::string,
                   kGlicUserStatusUrl,
                   &kGlicUserStatusCheck,
                   "glic-user-status-url",
#if BUILDFLAG(GOOGLE_CHROME_BRANDING)
                   "https://geminiweb-pa.googleapis.com/v1/glicStatus"
#else
                   ""
#endif
);
```

2.8. Feature Parameter Types

`base::FeatureParam` supports five built-in types (`bool`, `int`, `size_t`, `double`, `std::string`), `base::TimeDelta`, and enum types. (Ref. `base/metrics/field_trial_params.h`)

2.8.1. `base::TimeDelta`-type Feature Parameter Value

A feature can have a `base::TimeDelta`-type parameter. The parameter value specifies the unit and the value of the time delta using the helpers defined in `base/time/time.h`.

- The `GlicActorMoveBeforeClick` feature has a `base::TimeDelta`-type parameter `glic-actor-move-before-click-delay` with a default value of 5 milliseconds:

```
// chrome_features.h
extern const base::FeatureParam<base::TimeDelta> kGlicActorMoveBeforeClickDelay;

// chrome_features.cc
const base::FeatureParam<base::TimeDelta> kGlicActorMoveBeforeClickDelay{
    &kGlicActorMoveBeforeClick, "glic-actor-move-before-click-delay",
    base::Milliseconds(5)};
```

2.8.2. Feature Parameter with Fixed Set of Valid Values

A feature parameter can have a fixed set of valid values. In this case, the feature parameter is defined using an enumeration type, and the valid value set is defined using an array of `base::FeatureParameter::Option`, which contains a mapping from each enumerator to an actual parameter value. The array is typically defined with the `constexpr` specifier.

In most cases, the enumeration type name is the camel-case conversion of the parameter name, and the enumerator name is the `k`-prefixed camel-case conversion of the parameter value.

- The `GlicActor` feature has a `GlicActorEnterprisePrefDefault`-type parameter `glic_actor_enterprise_pref_default`. The valid values are [`enabled_by_default`, `disabled_by_default`, `forced_disabled`], with `disabled_by_default` as the default:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES) BASE_DECLARE_FEATURE(kGlicActor);
...
enum class GlicActorEnterprisePrefDefault {
  ...
};
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<GlicActorEnterprisePrefDefault>(
    kGlicActorEnterprisePrefDefault);

// chrome_features.cc
constexpr base::FeatureParam<GlicActorEnterprisePrefDefault>::Option
    kGlicActorEnterprisePrefDefaultOptions[] = {
        {GlicActorEnterprisePrefDefault::kEnabledByDefault,
         "enabled_by_default"},
        {GlicActorEnterprisePrefDefault::kDisabledByDefault,
         "disabled_by_default"},
```

```
        {GlicActorEnterprisePrefDefault::kForcedDisabled, "forced_disabled"},
};
BASE_FEATURE_ENUM_PARAM(GlicActorEnterprisePrefDefault,
                        kGlicActorEnterprisePrefDefault,
                        &kGlicActor,
                        "glic_actor_enterprise_pref_default",
                        GlicActorEnterprisePrefDefault::kDisabledByDefault,
                        &kGlicActorEnterprisePrefDefaultOptions);
```

2.9. Enum-Related Variations

2.9.1. Mismatch Between Feature Parameter Name and Enumeration Type

The enumeration type may be defined with a name that doesn't match the parameter name, usually for code readability.

- The `DisableBoostPriority` feature has an enumeration type parameter `exempt_processes`. The enumeration type name is not `ExemptProcesses` but `DisableBoostPriorityExemption`, for clarity:

```
// chrome_features.h
enum class DisableBoostPriorityExemption {
  kBrowserNetwork,
  kGpuBrowserNetwork,
  kLoadingBrowserNetwork,
  kForegroundBrowserNetwork
};

// chrome_features.cc
constinit const base::FeatureParam<DisableBoostPriorityExemption>
    kDisableBoostPriorityExemption{
        &kDisableBoostPriority, "exempt_processes",
        DisableBoostPriorityExemption::kForegroundBrowserNetwork,
        ...
```

2.9.2. C++ Specifier for Feature Parameter Option Array Definition

The C++ specifiers used for parameter option array definitions vary across the codebase (`constexpr`, `static constexpr`, `const`). Since these variations have no practical impact on the feature parameter initialization, they could potentially be unified to `static constexpr`.

- The option array of the `exempt_process` parameter is defined with the `static constexpr` specifier:

```
// chrome_features.cc
static constexpr base::FeatureParam<DisableBoostPriorityExemption>::Option
  kDisableBoostPriorityOptions[] = {
      ...
  };
constinit const base::FeatureParam<DisableBoostPriorityExemption>
    kDisableBoostPriorityExemption{
        &kDisableBoostPriority, "exempt_processes",
        ...
```

2.9.3. Mismatch Between Valid Parameter Value and Enumerator

The enumerator may be defined with a name that doesn't match the parameter value, usually for code readability.

- The enumerator for the parameter value `per_render_process_host` is defined as `kEnabledPerRenderProcessHost` instead of `kPerRenderProcessHost`, for clarity:

```
// content_features.h
enum class MBIMode {
  kLegacy,
  kEnabledPerRenderProcessHost,
  kEnabledPerSiteInstance,
};

// content_features.cc
const base::FeatureParam<MBIMode>::Option mbi_mode_types[] = {
    {MBIMode::kLegacy, "legacy"},
    {MBIMode::kEnabledPerRenderProcessHost, "per_render_process_host"},
    {MBIMode::kEnabledPerSiteInstance, "per_site_instance"}};
```

2.9.4. Explicit Enumerator Value

An enumerator can be defined with an explicit value.

- The enumerator `GlicActorEnterprisePrefDefault::kEnabledByDefault` has an explicit value of 0.

```
// chrome_features.h
enum class GlicActorEnterprisePrefDefault {
  kEnabledByDefault = 0,
  kDisabledByDefault,
  kForcedDisabled,
};
```

2.9.5. Use of Enum Defined in a Separate Header

In most cases, the enumeration is defined in the feature header file. But a feature can use an enumeration defined in a different header.

- The `triggering_action` parameter uses the `BtmTriggeringAction` enumeration defined in `btm_utils.h`:

```
// btm_utils.h
enum class BtmTriggeringAction { kNone, kBounce };

// content_features.h
#include "content/public/common/btm_utils.h"

// content_features.cc
constexpr base::FeatureParam<content::BtmTriggeringAction>::Option
    kBtmTriggeringActionOptions[] = {
        {content::BtmTriggeringAction::kNone, "none"},
        {content::BtmTriggeringAction::kBounce, "bounce"}};
const base::FeatureParam<content::BtmTriggeringAction> kBtmTriggeringAction{
    &kBtm, "triggering_action", content::BtmTriggeringAction::kBounce,
    &kBtmTriggeringActionOptions};
```

2.10. Use of `char[]` Strings

2.10.1. Exported Parameter Name String

A parameter name can be defined and exported in the form of a `char[]` string to avoid hardcoding it.

- The parameter name `glic-actor-ui-task-icon` is defined as a `const char[]` string `kGlicActorUiTaskIconName` and exported:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kGlicActorUiTaskIconName[];
...
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<bool> kGlicActorUiTaskIcon;

// chrome_features.cc
const char kGlicActorUiTaskIconName[] = "glic-actor-ui-task-icon";
...
const base::FeatureParam<bool> kGlicActorUiTaskIcon{
```

```
    &kGlicActorUi,
    kGlicActorUiTaskIconName,
    true
};
```

- The parameter name `boarding_pass_detector_urls` is defined as a `const char[]` string `kBoardingPassDetectorUrlParamName` and exported:

```
// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam;
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kBoardingPassDetectorUrlParamName[];

// chrome_features.cc
const char kBoardingPassDetectorUrlParamName[] = "boarding_pass_detector_urls";
const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam(
    &kBoardingPassDetector,
    kBoardingPassDetectorUrlParamName,
    "");
```

2.10.2. Exported Parameter Value String

A parameter value can be defined and exported in the form of a `char[]` string to avoid hardcoding it.

- The parameter values `after-loading`, `after-first-paint`, and `delayed-during-loading` are exported as `constexpr const char[]` strings for the parameter `spare_renderer_creation_timing`:

```
// content_features.h
CONTENT_EXPORT extern const base::FeatureParam<std::string>
    kAndroidSpareRendererCreationTiming;
inline constexpr const char kAndroidSpareRendererCreationAfterLoading[] =
    "after-loading";
inline constexpr const char kAndroidSpareRendererCreationAfterFirstPaint[] =
    "after-first-paint";
inline constexpr const char
    kAndroidSpareRendererCreationDelayedDuringLoading[] =
        "delayed-during-loading";

// content_features.cc
const base::FeatureParam<std::string> kAndroidSpareRendererCreationTiming{
    &kAndroidWarmUpSpareRendererWithTimeout, "spare_renderer_creation_timing",
    kAndroidSpareRendererCreationAfterLoading};
```

2.10.3. Field-Trial Parameter Name

When a feature has a field-trial parameter, the parameter name is defined in the form of a `char[]` string and exported.

- The `IsolateOrigins` feature has a field-trial parameter `OriginsList`. The parameter name is defined as a `const char[]` string `kIsolateOriginsFieldTrialParamName`:

```
// content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE(kIsolateOrigins);
CONTENT_EXPORT extern const char kIsolateOriginsFieldTrialParamName[];

// content_features.cc
BASE_FEATURE(kIsolateOrigins, base::FEATURE_DISABLED_BY_DEFAULT);
const char kIsolateOriginsFieldTrialParamName[] = "OriginsList";
```

2.11. File-Level Variations

There are other variations that differ across files:

- Generated header and source file name and path (e.g. `out/Debug/gen/chrome/common/chrome_features.h`)
- Header guard (e.g. `#ifndef CHROME_COMMON_CHROME_FEATURES_H_`)
- Header includes (e.g. `#include ...`)
- Export expression (e.g. `COMPONENT_EXPORT(CHROME_FEATURES)`)
- Helper methods

3. Variation Handling in JSON5

The following JSON5 schema is proposed to handle the variations identified in Section 3.

3.1. Basic Pattern of Feature Declaration and Definition

Each JSON5 feature definition file has a `features` field that contains feature definitions.

```
// json5
{
  features: [
    {
      // feature definition
    },
```

```
    ...
  ]
}
```

Each feature definition has a `name` field and a `status` field as mandatory fields. `status` can be `"disabled"` or `"enabled"`. The C++ identifier of the feature is derived from the feature name with `k` prepended. `"enabled"` and `"disabled"` are converted to `base::FEATURE_ENABLED_BY_DEFAULT` and `base::FEATURE_DISABLED_BY_DEFAULT` respectively. The feature is declared using the `BASE_DECLARE_FEATURE` macro, and defined using the `BASE_FEATURE` macro.

- `CryptographyComplianceCnsa` feature, disabled by default:

```
// chrome_features.json5
{
  name: "CryptographyComplianceCnsa",
  status: "disabled"
},

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kCryptographyComplianceCnsa);

// generated chrome_features.cc
BASE_FEATURE(kCryptographyComplianceCnsa, base::FEATURE_DISABLED_BY_DEFAULT);
```

3.2. Mismatch Between Feature Name and C++ Identifier

A feature definition has an optional `cpp_identifier` field for specifying a custom feature identifier that differs from the derived one.

- `AbusiveOriginNotificationPermissionRevocation` feature, defined with `kAbusiveNotificationPermissionRevocation` instead of the `k`-prefixed feature name, to use a shorter identifier:

```
// chrome_features.json5
{
  name: "AbusiveOriginNotificationPermissionRevocation",
  status: "enabled",
  cpp_identifier: "kAbusiveNotificationPermissionRevocation",
},

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kAbusiveNotificationPermissionRevocation);
```

```
// generated chrome_features.cc
BASE_FEATURE(kAbusiveNotificationPermissionRevocation,
             "AbusiveOriginNotificationPermissionRevocation",
             base::FEATURE_ENABLED_BY_DEFAULT);
```

3.3. Conditional Feature Definition with Preprocessor

3.3.1. Exclusive Feature

The `status` field can be a map, where the key is a preprocessor conditional expression, and the value is the feature status.

When the `status` is a map with a single condition, the feature declaration and definition are guarded behind the preprocessor condition, making it an exclusive feature. The `BUILDFLAG()` macro can be omitted to simplify the condition.

- `Borealis` feature, defined only for ChromeOS:

```
// chrome_features.json5
{
  name: "Borealis",
  status: {
    "IS_CHROMEOS": "disabled"
  }
},

// generated chrome_features.h
#if BUILDFLAG(IS_CHROMEOS)
COMPONENT_EXPORT(CHROME_FEATURES) BASE_DECLARE_FEATURE(kBorealis);
#endif

// generated chrome_features.cc
#if BUILDFLAG(IS_CHROMEOS)
BASE_FEATURE(kBorealis, base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

3.3.2. Conditional Feature Status

When the `status` field of a feature has more than one condition and one of them has the `default` key, the status of the feature becomes conditional.

- `DesktopPWAsPreventClose` feature, enabled for ChromeOS, otherwise disabled:

```
// chrome_features.json5
{
  name: "DesktopPWAsPreventClose",
  status: {
    "IS_CHROMEOS": "enabled",
    "default": "disabled"
  }
},

// generated chrome_features.cc
BASE_FEATURE(kDesktopPWAsPreventClose,
#if BUILDFLAG(IS_CHROMEOS)
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

Multiple conditions can map to the same status, allowing a separate comment for each.

- `WebAppSystemMediaControls` feature, enabled for Windows and macOS, otherwise disabled. The comments for Windows and macOS are adopted from `content_features.cc`:

```
// content_features.json5
{
  name: "WebAppSystemMediaControls",
  status: {
    // Windows enabled since 124.
    "IS_WIN": "enabled",
    // macOS enabled in 130. If a kill switch is needed, it should be
    // safe to only disable the failing platform (ie. macOS here).
    "IS_MAC": "enabled",
    "default": "disabled",
  },
},

// generated content_features.cc
BASE_FEATURE(kWebAppSystemMediaControls,
#if BUILDFLAG(IS_WIN)
             base::FEATURE_ENABLED_BY_DEFAULT);
#elif BUILDFLAG(IS_MAC)
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
```

3.3.3. Logical Expressions in Preprocessor Condition

The `status` map can have a logical expression as a key, allowing logical expressions in the preprocessor condition.

- `ChromeAppsDeprecation` feature, defined for Windows, macOS, and Linux:

```
// chrome_features.json5
{
  name: "ChromeAppsDeprecation",
  status: {
    "IS_WIN || IS_MAC || IS_LINUX": "enabled"
  }
},

// generated chrome_features.h
#if BUILDFLAG(IS_WIN) || BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kChromeAppsDeprecation);
#endif
```

- `PreinstalledWebAppInstallation` feature, defined for all platforms except Android:

```
// chrome_features.json5
{
  name: "DefaultWebAppInstallation",
  cpp_identifier: "kPreinstalledWebAppInstallation",
  status: {
    "!IS_ANDROID": "enabled"
  }
},

// generated chrome_features.h
#if !BUILDFLAG(IS_ANDROID)
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kPreinstalledWebAppInstallation);
#endif
```

- `AudioServiceOutOfProcess` feature, enabled for Windows, macOS, and Linux, otherwise disabled:

```
// content_features.json5
{
  name: "AudioServiceOutOfProcess",
  status: {
```

```
    "IS_WIN || IS_MAC || IS_LINUX": "enabled",
    "default": "disabled",
  },
},

// generated content_features.cc
BASE_FEATURE(kAudioServiceOutOfProcess,
#if BUILDFLAG(IS_WIN) || BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
             base::FEATURE_ENABLED_BY_DEFAULT
#else
             base::FEATURE_DISABLED_BY_DEFAULT
#endif
);
```

3.3.4. Exclusive Feature with Conditional Feature Status

When an exclusive feature needs to have a conditional status, the exclusive expression and the conditional expression can be joined with `&&`. Given that `default` is a predefined key for `#else` case, we can simply specify the exclusive expression for the default case.

By using the exclusive expression instead of "default" for the fallback status, the code generator can infer that the entire feature should be guarded behind the exclusive condition.

- `GlicGuestContentsVisibilityState` feature, defined only when glic is enabled, enabled for macOS and Linux, otherwise disabled:

```
// chrome_features.json5
{
  name: "GlicGuestContentsVisibilityState",
  status: {
    "ENABLE_GLIC && (IS_MAC || IS_LINUX)": "enabled",
    "ENABLE_GLIC": "disabled"
  }
},

// generated chrome_features.h
#if BUILDFLAG(ENABLE_GLIC)
...
COMPONENT_EXPORT(CHROME_FEATURES)
BASE_DECLARE_FEATURE(kGlicGuestContentsVisibilityState);
...
#endif

// generated chrome_features.cc
#if BUILDFLAG(ENABLE_GLIC)
...
```

```
BASE_FEATURE(kGlicGuestContentsVisibilityState,
#if BUILDFLAG(IS_MAC) || BUILDFLAG(IS_LINUX)
             base::FEATURE_ENABLED_BY_DEFAULT);
#else
             base::FEATURE_DISABLED_BY_DEFAULT);
#endif
...
#endif
```

3.3.5. Non-BUILDFLAG Preprocessor Condition

The exclusive and conditional expressions may use non-BUILDFLAG conditions. In the `chrome_features` and `content_features`, the preprocessor `defined()` operator is the only pattern that uses a non-BUILDFLAG condition, so we can simply use the operator as-is for these cases.

- `WebAssemblyTrapHandler` feature, enabled for Linux, ChromeOS, Windows, and macOS when ARCH_CPU_X86_64 is defined, or enabled for Linux, ChromeOS, and macOS when ARCH_CPU_ARM64 is defined, otherwise disabled:

```
// content_features.json5
{
  name: "WebAssemblyTrapHandler",
  status: {
    "((IS_LINUX || IS_CHROMEOS || IS_WIN || IS_MAC) && defined(ARCH_CPU_X86_64))
|| ((IS_LINUX || IS_CHROMEOS || IS_MAC) && defined(ARCH_CPU_ARM64))": "enabled",
    "default": "disabled",
  },
},

// generated content_features.cc
BASE_FEATURE(kWebAssemblyTrapHandler,
#if ((BUILDFLAG(IS_LINUX) || BUILDFLAG(IS_CHROMEOS) || BUILDFLAG(IS_WIN) ||
BUILDFLAG(IS_MAC)) && defined(ARCH_CPU_X86_64)) ||
((BUILDFLAG(IS_LINUX) || BUILDFLAG(IS_CHROMEOS) || BUILDFLAG(IS_MAC)) &&
defined(ARCH_CPU_ARM64))
             base::FEATURE_ENABLED_BY_DEFAULT
#else
             base::FEATURE_DISABLED_BY_DEFAULT
#endif
);
```

3.4. Basic Pattern of Feature Parameter Declaration and Definition

When a feature has parameters, a `params` array field can be added to specify each parameter with the mandatory fields `name`, `default_value`, and `type`. Similar to the feature identifier, the parameter identifier is derived from the parameter name by camel-casing it and prepending `k`. The parameter is declared using the `BASE_DECLARE_FEATURE_PARAM` macro, and defined using the `BASE_FEATURE_PARAM` macro.

- The `IgnoreDuplicateNavs` feature has a boolean-type parameter `skip_ignore_renderer_initiated_navs` with a default value of `false`:

```
// content_features.json5
{
  name: "IgnoreDuplicateNavs",
  ...
  params: [
    ...
    {
      name: "skip_ignore_renderer_initiated_navs",
      default_value: "false",
      type: "bool",
    },
    ...
  ],
},

// generated content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE(kIgnoreDuplicateNavs);
...
CONTENT_EXPORT BASE_DECLARE_FEATURE_PARAM(bool,
                                          kSkipIgnoreRendererInitiatedNavs);

// generated content_features.cc
BASE_FEATURE(kIgnoreDuplicateNavs, base::FEATURE_DISABLED_BY_DEFAULT);
...
BASE_FEATURE_PARAM(bool,
                   kSkipIgnoreRendererInitiatedNavs,
                   &kIgnoreDuplicateNavs,
                   "skip_ignore_renderer_initiated_navs",
                   false);
```

3.5. Parameter Declaration and Definition Without Macros

A parameter may have an optional `cpp_specifier` field, allowing the parameter to be declared and defined directly using `base::FeatureParam` with the specified C++ specifier.

- The `glic-allowed-origins-override` parameter of the `GlicCSPConfig` feature is defined using `base::FeatureParam` directly with the `const` specifier:

```
// chrome_features.json5
{
  name: "glic-allowed-origins-override",
  default_value: "https://gemini.google.com https://www.google.com",
  type: "std::string",
  cpp_specifier: "const",
}

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kGlicAllowedOriginsOverride;

// generated chrome_features.cc
const base::FeatureParam<std::string> kGlicAllowedOriginsOverride{
    &kGlicCSPConfig, "glic-allowed-origins-override",
    "https://gemini.google.com https://www.google.com"};
```

The `cpp_specifier` can also be specified selectively for the declaration and the definition using an object with `h` (header/declaration) and `cc` (source/definition) keys.

- The `moderate_level` parameter of the `LinuxLowMemoryMonitor` feature is defined using `base::FeatureParam` directly with the `const` specifier for declaration and the `constexpr` specifier for definition:

```
// chrome_features.json5
{
  name: "moderate_level",
  default_value: "50",
  type: "int",
  cpp_specifier: { cc: "constexpr", h: "const" },
  cpp_identifier: "kLinuxLowMemoryMonitorModerateLevel",
},

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<int> kLinuxLowMemoryMonitorModerateLevel;
```

```
// generated chrome_features.cc
constexpr base::FeatureParam<int> kLinuxLowMemoryMonitorModerateLevel{
    &kLinuxLowMemoryMonitor, "moderate_level", 50};
```

3.6. Mismatch Between Feature Parameter Name and C++ Identifier

Similar to the feature definition, a parameter may have a `cpp_identifier` field to specify a parameter identifier that differs from the derived one.

- The `allowed_tiers` parameter of the `GlicWebActuationSetting` feature is defined with `kGlicWebActuationAllowedTiers` instead of the `k`-prefixed/camel-cased parameter name, for clarity:

```
// chrome_features.json5
{
  name: "allowed_tiers",
  default_value: "",
  type: "std::string",
  cpp_specifier: "const",
  cpp_identifier: "kGlicWebActuationAllowedTiers",
}

// chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kGlicWebActuationAllowedTiers;

// chrome_features.cc
const base::FeatureParam<std::string> kGlicWebActuationAllowedTiers{
    &kGlicWebActuationSetting, "allowed_tiers", ""};
```

3.7. Conditional Parameter Default Value with Preprocessor

Similar to the `status` field of a feature, the default value of a parameter becomes conditional when the `default_value` field has more than one condition and one of them has the `default` key.

The `default_value` field follows the same rules for logical expressions (3.3.3.) and non-BUILDFLAG conditions (3.3.5.).

- The `GlicUserStatusCheck` feature has a `std::string`-type parameter `glic-user-status-url`. The default value of the parameter is `https://geminiweb-pa.googleapis.com/v1/glicStatus` for Google Chrome branding, otherwise an empty string:

```
// chrome_features.json5
{
  name: "glic-user-status-url",
  default_value: {
    "GOOGLE_CHROME_BRANDING": "https://geminiweb-pa.googleapis.com/v1/glicStatus",
    "default": ""
  },
  type: "std::string",
  ...
},

// chrome_features.cc
BASE_FEATURE_PARAM(std::string,
                   kGlicUserStatusUrl,
                   &kGlicUserStatusCheck,
                   "glic-user-status-url",
#if BUILDFLAG(GOOGLE_CHROME_BRANDING)
                   "https://geminiweb-pa.googleapis.com/v1/glicStatus"
#else
                   ""
#endif
);
```

3.8. Feature Parameter Types

3.8.1. `base::TimeDelta` Feature Parameter Value

When the type of a parameter is `base::TimeDelta`, the value is specified with a number and a unit to represent the time delta.

There are seven available units, which are compatible with the helpers in `base/time/time.h`:

Unit String	Time Helper
`ns`	`base::Nanoseconds()`
`us`	`base::Microseconds()`
`ms`	`base::Milliseconds()`
`s`	`base::Seconds()`
`m`	`base::Minutes()`
`h`	`base::Hours()`

d	base::Days()

- The `GlicActorMoveBeforeClick` feature has a `base::TimeDelta`-type parameter `glic-actor-move-before-click-delay` with a default value of 5 milliseconds:

```
// chrome_features.json5
{
  name: "glic-actor-move-before-click-delay",
  default_value: "5ms",
  type: "base::TimeDelta",
  cpp_specifier: "const",
}

// generated chrome_features.h
extern const base::FeatureParam<base::TimeDelta> kGlicActorMoveBeforeClickDelay;

// generated chrome_features.cc
const base::FeatureParam<base::TimeDelta> kGlicActorMoveBeforeClickDelay{
    &kGlicActorMoveBeforeClick, "glic-actor-move-before-click-delay",
    base::Milliseconds(5)};
```

3.8.2. Feature Parameter with Fixed Set of Valid Values

When a parameter has a fixed set of valid values, the `type` field is specified as `"enum"`, and the valid value set is specified in the `valid_values` field. In this case, the `default_value` must be one of the valid values.

- The `GlicActor` feature has a `GlicActorEnterprisePrefDefault`-type parameter `glic_actor_enterprise_pref_default`. The valid values are [`enabled_by_default`, `disabled_by_default`, `forced_disabled`], with `disabled_by_default` as the default:

```
// chrome_features.json5
{
  name: "glic_actor_enterprise_pref_default",
  default_value: "disabled_by_default",
  type: "enum",
  valid_values: [
    "enabled_by_default",
    "disabled_by_default",
    "forced_disabled",
  ],
},

// generated chrome_features.h
```

```
COMPONENT_EXPORT(CHROME_FEATURES) BASE_DECLARE_FEATURE(kGlicActor);
...
enum class GlicActorEnterprisePrefDefault {
  kEnabledByDefault,
  kDisabledByDefault,
  kForcedDisabled,
};
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<GlicActorEnterprisePrefDefault>(
    kGlicActorEnterprisePrefDefault);

// generated chrome_features.cc
constexpr base::FeatureParam<GlicActorEnterprisePrefDefault>::Option
    kGlicActorEnterprisePrefDefaultOptions[] = {
        {GlicActorEnterprisePrefDefault::kEnabledByDefault,
         "enabled_by_default"},
        {GlicActorEnterprisePrefDefault::kDisabledByDefault,
         "disabled_by_default"},
        {GlicActorEnterprisePrefDefault::kForcedDisabled, "forced_disabled"},
};
BASE_FEATURE_ENUM_PARAM(GlicActorEnterprisePrefDefault,
                        kGlicActorEnterprisePrefDefault,
                        &kGlicActor,
                        "glic_actor_enterprise_pref_default",
                        GlicActorEnterprisePrefDefault::kDisabledByDefault,
                        &kGlicActorEnterprisePrefDefaultOptions);
```

3.9. Enum-Related Variations

3.9.1. Mismatch Between Feature Parameter Name and Enumeration Type

When an enumeration for a parameter needs to be defined with a name that differs from the derived one, the name can be specified under the `valid_values` field. In this case, the `valid_values` field is not an array but an object that has an `options` field for the value set. The enumeration name is specified in the `enum_type` field of the valid values object.

- The `DisableBoostPriority` feature has an enumeration type parameter `exempt_processes`. The enumeration type name is not `ExemptProcesses` but `DisableBoostPriorityExemption`, for clarity:

```
// chrome_features.json5
{
  name: "exempt_processes",
  default_value: "ForegroundBrowserNetwork",
  type: "enum",
```

```
  ...
  valid_values: {
    enum_type: "DisableBoostPriorityExemption",
    options: [
      "BrowserNetwork",
      "GpuBrowserNetwork",
      "LoadingBrowserNetwork",
      "ForegroundBrowserNetwork"
    ],
  },
}

// generated chrome_features.h
enum class DisableBoostPriorityExemption {
  kBrowserNetwork,
  kGpuBrowserNetwork,
  kLoadingBrowserNetwork,
  kForegroundBrowserNetwork
};

// generated chrome_features.cc
...
constinit const base::FeatureParam<DisableBoostPriorityExemption>
    kDisableBoostPriorityExemption{
        &kDisableBoostPriority, "exempt_processes",
        DisableBoostPriorityExemption::kForegroundBrowserNetwork,
        ...
```

3.9.2. C++ Specifier for Feature Parameter Option Array Definition

To use a C++ specifier other than `constexpr` for the option array, the valid values object has an optional `cpp_specifier` field.

- The option array of the `exempt_process` parameter is defined with the `static constexpr` specifier:

```
// chrome_features.json5
{
  name: "exempt_processes",
  default_value: "ForegroundBrowserNetwork",
  type: "enum",
  ...
  valid_values: {
    ...
    cpp_specifier: "static constexpr",
    options: [
```

```
      ...
    ],
  },
}

// generated chrome_features.cc
static constexpr base::FeatureParam<DisableBoostPriorityExemption>::Option
  kDisableBoostPriorityOptions[] = {
      ...
  };
constinit const base::FeatureParam<DisableBoostPriorityExemption>
    ...
```

3.9.3. Mismatch Between Valid Parameter Value and Enumerator

To use an enumerator name other than the derived one, the `valid_values.options` can have an object with an `enumerator` field specifying the name. In this case, the option value is specified in the `value` field.

- The enumerator for the parameter value `per_render_process_host` is defined as `kEnabledPerRenderProcessHost` instead of `kPerRenderProcessHost`, for clarity:

```
// content_features.json5
{
  name: "mode",
  type: "enum",
  ...
  valid_values: {
    enum_type: "MBIMode",
    options: [
      "legacy",  // kLegacy
      {value: "per_render_process_host", enumerator:
"kEnabledPerRenderProcessHost"},
      {value: "per_site_instance", enumerator: "kEnabledPerSiteInstance"},
    ],
  },
},

// generated content_features.h
enum class MBIMode {
  kLegacy,
  kEnabledPerRenderProcessHost,
  kEnabledPerSiteInstance,
};
```

```
// generated content_features.cc
const base::FeatureParam<MBIMode>::Option mbi_mode_types[] = {
    {MBIMode::kLegacy, "legacy"},
    {MBIMode::kEnabledPerRenderProcessHost, "per_render_process_host"},
    {MBIMode::kEnabledPerSiteInstance, "per_site_instance"}};
```

3.9.4. Explicit Enumerator Value

To specify a value for an enumerator, the `valid_values.options` can have an object with an `enumerator_value` field specifying the value. Similar to the above, the option value is specified in the `value` field.

- The enumerator `GlicActorEnterprisePrefDefault::kEnabledByDefault` has an explicit value of 0.

```
// chrome_features.json5
{
  name: "glic_actor_enterprise_pref_default",
  type: "enum",
  ...
  valid_values: [
    {value: "enabled_by_default", enumerator_value: "0"},
    "disabled_by_default",
    "forced_disabled",
  ],
},

// generated chrome_features.h
enum class GlicActorEnterprisePrefDefault {
  kEnabledByDefault = 0,
  kDisabledByDefault,
  kForcedDisabled,
};
```

3.9.5. Use of Enum Defined in a Separate Header

To use an enumeration defined in a different header file, the valid values object has an optional boolean `external_enum` field. When the field is true, the generated feature header doesn't include an enumeration definition. In this case, the `enum_type` field needs to include the namespace if the enumeration is defined in a namespace other than `features`.

- The `triggering_action` parameter uses the `BtmTriggeringAction` enumeration defined in `btm_utils.h`:

```
// content_features.json5
{
  name: "triggering_action",
  type: "enum",
  ...
  valid_values: {
    enum_type: "content::BtmTriggeringAction",
    external_enum: true,
    options: [
      "none",  // kNone
      "bounce",  // kBounce
    ],
  },
},

// btm_utils.h
enum class BtmTriggeringAction { kNone, kBounce };

// generated content_features.h
#include "content/public/common/btm_utils.h"

// generated content_features.cc
constexpr base::FeatureParam<content::BtmTriggeringAction>::Option
    kBtmTriggeringActionOptions[] = {
        {content::BtmTriggeringAction::kNone, "none"},
        {content::BtmTriggeringAction::kBounce, "bounce"}};
const base::FeatureParam<content::BtmTriggeringAction> kBtmTriggeringAction{
    &kBtm, "triggering_action", content::BtmTriggeringAction::kBounce,
    &kBtmTriggeringActionOptions};
```

3.10. Use of `char[]` Strings

3.10.1. Exported Parameter Name String

A parameter definition has an optional boolean `export_name` field. When the field is set, the parameter name is defined and exported as a constant char array variable. The C++ identifier of the variable is derived from the parameter name by camel-casing it, with `k` prepended and `Name` appended.

- The parameter name `glic-actor-ui-task-icon` is defined as a `const char[]` string `kGlicActorUiTaskIconName` and exported:

```
// chrome_features.json5
{
  name: "glic-actor-ui-task-icon",
  default_value: "true",
```

```
  type: "bool",
  cpp_specifier: "const",
  export_name: true,
},

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kGlicActorUiTaskIconName[];
...
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<bool> kGlicActorUiTaskIcon;

// generated chrome_features.cc
const char kGlicActorUiTaskIconName[] = "glic-actor-ui-task-icon";
...
const base::FeatureParam<bool> kGlicActorUiTaskIcon{
    &kGlicActorUi,
    kGlicActorUiTaskIconName,
    true
};
```

When a parameter specifies the `cpp_identifier` field, the identifier of the parameter name variable is the `cpp_identifier` with `Name` appended.

- The parameter name `boarding_pass_detector_urls` is defined as a `const char[]` string `kBoardingPassDetectorUrlParamName` and exported:

```
// chrome_features.json5
{
  name: "boarding_pass_detector_urls",
  default_value: "",
  type: "std::string",
  cpp_specifier: "const",
  cpp_identifier: "kBoardingPassDetectorUrlParam",
  export_name: true,
}

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam;
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kBoardingPassDetectorUrlParamName[];

// generated chrome_features.cc
const char kBoardingPassDetectorUrlParamName[] = "boarding_pass_detector_urls";
const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam(
    &kBoardingPassDetector,
```

```
    kBoardingPassDetectorUrlParamName,
    "");
```

A parameter name variable can have a different identifier from the derived one by using an object with a `cpp_identifier` field.

- The parameter name `boarding_pass_detector_urls` is defined as a `const char[]` string `kSomeOtherParamNameIdentifier` and exported:

```
// chrome_features.json5
{
  name: "boarding_pass_detector_urls",
  default_value: "",
  type: "std::string",
  cpp_specifier: "const",
  cpp_identifier: "kBoardingPassDetectorUrlParam",
  export_name: { cpp_identifier: "kSomeOtherParamNameIdentifier" },
}

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam;
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kSomeOtherParamNameIdentifier[];

// generated chrome_features.cc
const char kSomeOtherParamNameIdentifier[] = "boarding_pass_detector_urls";
const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam(
    &kBoardingPassDetector,
    kSomeOtherParamNameIdentifier,
    "");
```

The parameter name can be exported in the form of `inline constexpr const char[]` without a separate definition in the source file, by using an object with the boolean `inline_constexpr` field set.

- The parameter name `boarding_pass_detector_urls` is defined as an `inline constexpr const char[]` string `kBoardingPassDetectorUrlParamName` and exported:

```
// chrome_features.json5
{
  name: "boarding_pass_detector_urls",
  default_value: "",
  type: "std::string",
  cpp_specifier: "const",
  cpp_identifier: "kBoardingPassDetectorUrlParam",
  export_name: { inline_constexpr: true },
```

```
}

// generated chrome_features.h
COMPONENT_EXPORT(CHROME_FEATURES)
extern const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam;
COMPONENT_EXPORT(CHROME_FEATURES)
inline constexpr const char kBoardingPassDetectorUrlParamName[] =
    "boarding_pass_detector_urls";

// generated chrome_features.cc
const base::FeatureParam<std::string> kBoardingPassDetectorUrlParam(
    &kBoardingPassDetector,
    kBoardingPassDetectorUrlParamName,
    "");
```

3.10.2. Exported Parameter Value String

A parameter definition has an optional `export_values` array field to specify the parameter values to be exported. Similar to the parameter name, each parameter value in the array is defined and exported as a constant char array variable. The C++ identifier of the variable is derived from the parameter name and the value by camel-casing them, with `k` prepended and `Value` inserted between the name and the value.

- The parameter values `option-a`, `option-b`, and `option-c` are exported for the `test-parameter` parameter:

```
// content_features.json5
{
  name: "test-parameter",
  default_value: "option-a",
  type: "std::string",
  export_values: [ "option-a", "option-b", "option-c" ]
}

// generated content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE_PARAM(std::string, kTestParameter);
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kTestParameterValueOptionA[];
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kTestParameterValueOptionB[];
COMPONENT_EXPORT(CHROME_FEATURES)
extern const char kTestParameterValueOptionC[];

// generated content_features.cc
const char kTestParameterValueOptionA[] = "option-a";
const char kTestParameterValueOptionB[] = "option-b";
```

```
const char kTestParameterValueOptionC[] = "option-c";
BASE_FEATURE_PARAM(std::string,
                   kTestParameter,
                   &kTestFeature,
                   "test-parameter",
                   kTestParameterValueOptionA);
```

Similar to the parameter name, the `cpp_identifier` and `inline_constexpr` fields can be specified in an object for each parameter value, with a `value` field holding the actual parameter value.

- The parameter values `after-loading`, `after-first-paint`, and `delayed-during-loading` are exported as `constexpr const char[]` strings for the parameter `spare_renderer_creation_timing`:

```
// content_features.json5
{
  name: "spare_renderer_creation_timing",
  default_value: "after-loading",
  type: "std::string",
  cpp_specifier: "const",
  cpp_identifier: "kAndroidSpareRendererCreationTiming",
  export_values: [
    {value: "after-loading",
     cpp_identifier: "kAndroidSpareRendererCreationAfterLoading",
     inline_constexpr: true},
    {value: "after-first-paint",
     cpp_identifier: "kAndroidSpareRendererCreationAfterFirstPaint",
     inline_constexpr: true},
    {value: "delayed-during-loading",
     cpp_identifier: "kAndroidSpareRendererCreationDelayedDuringLoading",
     inline_constexpr: true},
  ],
},

// generated content_features.h
CONTENT_EXPORT extern const base::FeatureParam<std::string>
    kAndroidSpareRendererCreationTiming;
inline constexpr const char kAndroidSpareRendererCreationAfterLoading[] =
    "after-loading";
inline constexpr const char kAndroidSpareRendererCreationAfterFirstPaint[] =
    "after-first-paint";
inline constexpr const char
    kAndroidSpareRendererCreationDelayedDuringLoading[] =
        "delayed-during-loading";

// generated content_features.cc
const base::FeatureParam<std::string> kAndroidSpareRendererCreationTiming{
```

```
    &kAndroidWarmUpSpareRendererWithTimeout, "spare_renderer_creation_timing",
    kAndroidSpareRendererCreationAfterLoading};
```

3.10.3. Field-Trial Parameter Name

A feature definition has an optional `field_trial_params` array field to specify field-trial parameter names. Similar to the other exported string values, each field-trial parameter name in the array is defined and exported as a constant char array variable. The C++ identifier of the variable is derived from the feature name and the parameter name by camel-casing them, with `k` prepended and `FieldTrialParam` inserted between the feature name and the parameter name.

- The `IsolateOrigins` feature has a field-trial parameter `OriginsList`. The parameter name is defined as a `const char[]` string `kIsolateOriginsFieldTrialParamOriginsList`:

```
// content_features.json5
{
  name: "IsolateOrigins",
  status: "disabled",
  field_trial_params: [ "OriginsList" ]
},

// generated content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE(kIsolateOrigins);
CONTENT_EXPORT extern const char kIsolateOriginsFieldTrialParamOriginsList[];

// generated content_features.cc
BASE_FEATURE(kIsolateOrigins, base::FEATURE_DISABLED_BY_DEFAULT);
const char kIsolateOriginsFieldTrialParamOriginsList[] = "OriginsList";
```

Similar to the parameter value, the `cpp_identifier` and `inline_constexpr` fields can be specified in an object for each field-trial parameter name, with a `name` field holding the actual parameter name.

- The `IsolateOrigins` feature has a field-trial parameter `OriginsList`. The parameter name is defined as a `const char[]` string `kIsolateOriginsFieldTrialParamName`:

```
// content_features.json5
{
  name: "IsolateOrigins",
  status: "disabled",
  field_trial_params: [
    {name: "OriginsList",
     cpp_identifier: "kIsolateOriginsFieldTrialParamName"},
  ],
},
```

```
// generated content_features.h
CONTENT_EXPORT BASE_DECLARE_FEATURE(kIsolateOrigins);
CONTENT_EXPORT extern const char kIsolateOriginsFieldTrialParamName[];

// generated content_features.cc
BASE_FEATURE(kIsolateOrigins, base::FEATURE_DISABLED_BY_DEFAULT);
const char kIsolateOriginsFieldTrialParamName[] = "OriginsList";
```

3.11. File-Level Variations

The root object has the following fields to handle file-level variations:

- `header_includes`: An array field for the header file paths to be included in the `.h` file.
- `header_system_includes`: An array field for the system headers to be included in the `.h` file.
- `cc_includes`: An array field for the header file paths to be included in the `.cc` file.
- `cc_system_includes`: An array field for the system headers to be included in the `.cc` file.
- `export_expression`: A string field to specify the export expression.

```
// content_features.json5
{
  export_expression: "CONTENT_EXPORT",
  header_system_includes: ["string"],
  header_includes: [
    "base/feature_list.h",
    "base/metrics/field_trial_params.h",
    "base/time/time.h",
    "build/build_config.h",
    "content/common/content_export.h",
    "content/public/common/btm_utils.h",
    "content/public/common/buildflags.h",
    "tools/v8_context_snapshot/buildflags.h",
  ],
  cc_system_includes: ["string"],
  cc_includes: [
    "base/feature_list.h",
    "base/time/time.h",
    "build/android_buildflags.h",
    "build/build_config.h",
    "build/config/chromebox_for_meetings/buildflags.h",
    "content/common/buildflags.h",
    "content/public/common/btm_utils.h",
    "content/public/common/buildflags.h",
    "media/base/media_switches.h",
```

```
  ],

  ...

}
```

Since the generated file path/name and the header guard can be derived from the JSON5 file path/name, they do not need to be specified in the JSON5.

The helper methods declared and defined in the feature header/source file need to be relocated to separate files. (e.g., the helper methods in `content_features.h/.cc` would need to be relocated to `content_features_helpers.h/.cc`)

4. Examples

Examples with the first 10 features in `chrome_features.h/.cc` and `content_features.h/.cc`. These demonstrate several of the variations described in Sections 2 and 3.

4.1 `chrome/common/chrome_features.json5`

```
// Copyright 2026 The Chromium Authors
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

// This file defines all the public base::FeatureList features for the chrome
// module.
{
  export_expression: "COMPONENT_EXPORT(CHROME_FEATURES)",
  header_includes: [
    "base/component_export.h",
    "base/feature_list.h",
    "base/metrics/field_trial_params.h",
    "base/time/time.h",
    "build/build_config.h",
    "build/buildflag.h",
    "chrome/common/buildflags.h",
    "device/vr/buildflags/buildflags.h",
    "extensions/buildflags/buildflags.h",
    "pdf/buildflags.h",
    "ui/base/buildflags.h",
  ],
  cc_includes: [
    "base/command_line.h",
    "base/feature_list.h",
    "base/strings/string_split.h",
    "base/time/time.h",
    "build/branding_buildflags.h",
```

```
    "build/build_config.h",
    "chrome/common/chrome_switches.h",
    "pdf/buildflags.h",
  ],

  features: [
    // When enabled, notifications from PWA's will use the PWA icon and name,
    // as long as the PWA is on the start menu.  b/40285965.
    {
      name: "AppSpecificNotifications",
      status: {
        "IS_WIN": "enabled"
      }
    },

    // When enabled, invokes `SetProcessPriorityBoost` to disable priority
boosting
    // when a thread is taken out of the wait state. The default Windows behavior
is
    // to boost when taking a thread out of waking state. On other platforms, the
    // default is not to boost and implementing boosting regresses input and page
    // load metrics. Therefore, this experiment on Windows to evaluates if
operating
    // without boosting improves these metrics. This is a field-sampling
experiment
    // and is not intended to be shipped as is regardless of the outcome but
rather
    // to gather data before the design phase of enhanced cross-platform
scheduling
    // primitives.
    {
      name: "DisableBoostPriority",
      status: {
        "IS_WIN": "disabled"
      },
      params: [
        {
          name: "exempt_processes",
          default_value: "ForegroundBrowserNetwork",
          type: "enum",
          cpp_specifier: { cc: "constinit const" },
          cpp_identifier: "kDisableBoostPriorityExemption",
          valid_values: {
            enum_type: "DisableBoostPriorityExemption",
            cpp_specifier: "static constexpr",
            options: [
              // Priority boosting is disabled for all processes except Browser
and Network.
```

```
              "BrowserNetwork",  // kBrowserNetwork
              // Priority boosting is disabled for all processes except GPU,
Browser, and
              // Network.
              "GpuBrowserNetwork",  // kGpuBrowserNetwork
              // Priority boosting is disabled for all processes except Browser,
Renderers
              // that are currently loading, and Network while there is at least
one
              // renderer loading.
              "LoadingBrowserNetwork",  // kLoadingBrowserNetwork
              // Priority boosting is disabled for all processes except Browser,
Network,
              // and Foreground renderers.
              "ForegroundBrowserNetwork"  // kForegroundBrowserNetwork
            ],
          },
        }
      ]
    },

    // Can be used to disable RemoteCocoa (hosting NSWindows for apps in the app
    // process). For debugging purposes only.
    {
      name: "AppShimRemoteCocoa",
      status: {
        "IS_MAC": "enabled"
      }
    },

    // This is used to control the new app close behavior on macOS wherein closing
    // all windows for an app leaves the app running.
    // https://crbug.com/1080729
    {
      name: "AppShimNewCloseBehavior",
      status: {
        "IS_MAC": "disabled"
      }
    },

    // When enabled, app shims try to launch chrome silently if chrome isn't
already
    // running, rather than have chrome launch visibly with a new tab/profile
    // selector.
    // https://crbug.com/1205537
    {
      name: "AppShimLaunchChromeSilently",
      status: {
```

```
        "IS_MAC": "enabled"
      }
    },

    // When enabled, notifications coming from PWAs will be displayed via their
app
    // shim processes, rather than directly by chrome.
    // https://crbug.com/40616749
    {
      name: "AppShimNotificationAttribution",
      status: {
        "IS_MAC": "disabled"
      }
    },

    // When enabled, app shims used by PWAs will be signed with an ad-hoc
signature
    // https://crbug.com/40276068
    {
      name: "UseAdHocSigningForWebAppShims",
      status: {
        "IS_MAC": "enabled"
      }
    },

    // When enabled, the KeychainKeyProvider is used to provide the OS Crypt async
    // key.
    {
      name: "UseKeychainKeyProvider",
      status: {
        "IS_MAC": "enabled"
      }
    },

    // Enables or disables the Autofill survey triggered by opening a prompt to
    // save address info.
    {
      name: "AutofillAddressSurvey",
      status: {
        "IS_WIN || IS_MAC || IS_LINUX || IS_CHROMEOS": "disabled"
      }
    },

    // Enables or disables the Autofill survey triggered by opening a prompt to
    // save credit card info.
    {
      name: "AutofillCardSurvey",
      status: {
```

```
        "IS_WIN || IS_MAC || IS_LINUX || IS_CHROMEOS": "disabled"
      }
    },

    ...

}
```

4.2 content/public/common/content_features.json5

```
// Copyright 2026 The Chromium Authors
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

// This file defines all the public base::FeatureList features for the content
// module.
//
// BEFORE MODIFYING THIS FILE: If your feature is only used inside content/, add
// your feature in `content/common/features.json5` instead.
{
  export_expression: "CONTENT_EXPORT",
  header_system_includes: ["string"],
  header_includes: [
    "base/feature_list.h",
    "base/metrics/field_trial_params.h",
    "base/time/time.h",
    "build/build_config.h",
    "content/common/content_export.h",
    "content/public/common/btm_utils.h",
    "content/public/common/buildflags.h",
    "tools/v8_context_snapshot/buildflags.h",
  ],
  cc_system_includes: ["string"],
  cc_includes: [
    "base/feature_list.h",
    "base/time/time.h",
    "build/android_buildflags.h",
    "build/build_config.h",
    "build/config/chromebox_for_meetings/buildflags.h",
    "content/common/buildflags.h",
    "content/public/common/btm_utils.h",
    "content/public/common/buildflags.h",
    "media/base/media_switches.h",
  ],

  features: [
    // All features in alphabetical order.
```

```
    // Marks navigations as aborted when the NavigationHandle is destroyed mid
    // navigation, likely due to a tab closure. This is a kill switch.
    {
      name: "AbortNavigationsFromTabClosures",
      status: "enabled",
    },

    // Capture Android key event objects to send them to the web contents when the
    // IME sends composition texts.
    {
      name: "AndroidCaptureKeyEvents",
      status: "disabled",
    },

    // Enables the caret browsing a11y feature - can use arrow keys to navigate
    // through web pages.
    {
      name: "AndroidCaretBrowsing",
      status: "disabled",
    },

    // DevTools frontend for Android.
    {
      name: "AndroidDevToolsFrontend",
      status: "disabled",
    },

    // Enables media capturing to continue in the background.
    {
      name: "AndroidEnableBackgroundMediaCapturing",
      status: "disabled",
    },

    // Enables media to continue playing in the background.
    {
      name: "AndroidEnableBackgroundMediaLargeFormFactors",
      status: "enabled",
    },

    // Fallback to next named service slot if launching a privileged service
process
    // hangs. In practice, this means if GPU launch hanges, then retry it once.
    {
      name: "AndroidFallbackToNextSlot",
      status: "enabled",
    },

    // Enables IMEs to insert media content such as images, gifs and stickers.
```

```
    {
      name: "AndroidMediaInsertion",
      status: "disabled",
    },

    // Enables the physical keyboard autocorrect underline feature.
    {
      name: "AndroidPkAutocorrectUnderline",
      status: "disabled",
    },

    // Enables the spelling underline in composition mode.
    {
      name: "AndroidSpellingUnderlineInCompositionMode",
      status: "enabled",
    },

    ...

}
```