Captured 2026-10-30T08:17:33.827Z from https://chromium.googlesource.com/chromium/src/+/refs/tags/64.0.3282.24/google_apis/google_api_keys.cc

google_api_keys.cc

blob: 2e8437ed136e682540835ef4738e3519f9cb6051 [file]

```
1  // Copyright (c) 2012 The Chromium Authors. All rights reserved.
2  // Use of this source code is governed by a BSD-style license that can be
3  // found in the LICENSE file.
4
5  #include "google_apis/google_api_keys.h"
6
7  // If you add more includes to this list, you also need to add them to
8  // google_api_keys_unittest.cc.
9
10 #include <stddef.h>
11
12 #include <memory>
13
14 #include "base/command_line.h"
15 #include "base/environment.h"
16 #include "base/lazy_instance.h"
17 #include "base/logging.h"
18 #include "base/strings/stringize_macros.h"
19 #include "google_apis/gaia/gaia_switches.h"
20
21 #if defined(OS_MACOSX)
22 #include "google_apis/google_api_keys_mac.h"
23 #endif
24
25 #if defined(GOOGLE_CHROME_BUILD) || defined(USE_OFFICIAL_GOOGLE_API_KEYS)
26 #include "google_apis/internal/google_chrome_api_keys.h"
27 #endif
28
29 // Used to indicate an unset key/id/secret.  This works better with
30 // various unit tests than leaving the token empty.
31 #define DUMMY_API_TOKEN "dummytoken"
32
33 #if !defined(GOOGLE_API_KEY)
34 #define GOOGLE_API_KEY DUMMY_API_TOKEN
35 #endif
36
37 #if !defined(GOOGLE_CLIENT_ID_MAIN)
38 #define GOOGLE_CLIENT_ID_MAIN DUMMY_API_TOKEN
39 #endif
40
```

```
#if !defined(GOOGLE_CLIENT_SECRET_MAIN)
#define GOOGLE_CLIENT_SECRET_MAIN DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_ID_CLOUD_PRINT)
#define GOOGLE_CLIENT_ID_CLOUD_PRINT DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_SECRET_CLOUD_PRINT)
#define GOOGLE_CLIENT_SECRET_CLOUD_PRINT DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_ID_REMOTING)
#define GOOGLE_CLIENT_ID_REMOTING DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_SECRET_REMOTING)
#define GOOGLE_CLIENT_SECRET_REMOTING DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_ID_REMOTING_HOST)
#define GOOGLE_CLIENT_ID_REMOTING_HOST DUMMY_API_TOKEN
#endif

#if !defined(GOOGLE_CLIENT_SECRET_REMOTING_HOST)
#define GOOGLE_CLIENT_SECRET_REMOTING_HOST DUMMY_API_TOKEN
#endif

// This is really the API key for non-stable channels on Android. It's
// named after the first feature that used it.
// TODO(jkrcal,rogerta): Rename this to GOOGLE_API_KEY_ANDROID_NON_STABLE.
#if !defined(GOOGLE_API_KEY_PHYSICAL_WEB_TEST)
#define GOOGLE_API_KEY_PHYSICAL_WEB_TEST DUMMY_API_TOKEN
#endif

// These are used as shortcuts for developers and users providing
// OAuth credentials via preprocessor defines or environment
// variables.  If set, they will be used to replace any of the client
// IDs and secrets above that have not been set (and only those; they
// will not override already-set values).
#if !defined(GOOGLE_DEFAULT_CLIENT_ID)
#define GOOGLE_DEFAULT_CLIENT_ID ""
#endif
#if !defined(GOOGLE_DEFAULT_CLIENT_SECRET)
#define GOOGLE_DEFAULT_CLIENT_SECRET ""
#endif

namespace google_apis {

```

```
const char kAPIKeysDevelopersHowToURL[] =
    "http://www.chromium.org/developers/how-tos/api-keys";

// This is used as a lazy instance to determine keys once and cache them.
class APIKeyCache {
 public:
  APIKeyCache() {
    std::unique_ptr<base::Environment> environment(base::Environment::Create());
    base::CommandLine* command_line = base::CommandLine::ForCurrentProcess();

    api_key_ = CalculateKeyValue(
        GOOGLE_API_KEY, STRINGIZE_NO_EXPANSION(GOOGLE_API_KEY), NULL,
        std::string(), environment.get(), command_line);

// A special non-stable key is at the moment defined only for Android Chrome.
#if defined(OS_ANDROID)
    api_key_non_stable_ = CalculateKeyValue(
            GOOGLE_API_KEY_PHYSICAL_WEB_TEST,
            STRINGIZE_NO_EXPANSION(GOOGLE_API_KEY_PHYSICAL_WEB_TEST), NULL,
            std::string(), environment.get(), command_line);
#else
    api_key_non_stable_ = api_key_;
#endif

    std::string default_client_id =
        CalculateKeyValue(GOOGLE_DEFAULT_CLIENT_ID,
                          STRINGIZE_NO_EXPANSION(GOOGLE_DEFAULT_CLIENT_ID),
                          NULL,
                          std::string(),
                          environment.get(),
                          command_line);
    std::string default_client_secret =
        CalculateKeyValue(GOOGLE_DEFAULT_CLIENT_SECRET,
                          STRINGIZE_NO_EXPANSION(GOOGLE_DEFAULT_CLIENT_SECRET),
                          NULL,
                          std::string(),
                          environment.get(),
                          command_line);

    // We currently only allow overriding the baked-in values for the
    // default OAuth2 client ID and secret using a command-line
    // argument, since that is useful to enable testing against
    // staging servers, and since that was what was possible and
    // likely practiced by the QA team before this implementation was
    // written.
    client_ids_[CLIENT_MAIN] = CalculateKeyValue(
        GOOGLE_CLIENT_ID_MAIN,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_ID_MAIN),
        switches::kOAuth2ClientID,
```

```
        default_client_id,
        environment.get(),
        command_line);
    client_secrets_[CLIENT_MAIN] = CalculateKeyValue(
        GOOGLE_CLIENT_SECRET_MAIN,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_SECRET_MAIN),
        switches::kOAuth2ClientSecret,
        default_client_secret,
        environment.get(),
        command_line);

    client_ids_[CLIENT_CLOUD_PRINT] = CalculateKeyValue(
        GOOGLE_CLIENT_ID_CLOUD_PRINT,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_ID_CLOUD_PRINT),
        NULL,
        default_client_id,
        environment.get(),
        command_line);
    client_secrets_[CLIENT_CLOUD_PRINT] = CalculateKeyValue(
        GOOGLE_CLIENT_SECRET_CLOUD_PRINT,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_SECRET_CLOUD_PRINT),
        NULL,
        default_client_secret,
        environment.get(),
        command_line);

    client_ids_[CLIENT_REMOTING] = CalculateKeyValue(
        GOOGLE_CLIENT_ID_REMOTING,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_ID_REMOTING),
        NULL,
        default_client_id,
        environment.get(),
        command_line);
    client_secrets_[CLIENT_REMOTING] = CalculateKeyValue(
        GOOGLE_CLIENT_SECRET_REMOTING,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_SECRET_REMOTING),
        NULL,
        default_client_secret,
        environment.get(),
        command_line);

    client_ids_[CLIENT_REMOTING_HOST] = CalculateKeyValue(
        GOOGLE_CLIENT_ID_REMOTING_HOST,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_ID_REMOTING_HOST),
        NULL,
        default_client_id,
        environment.get(),
        command_line);
    client_secrets_[CLIENT_REMOTING_HOST] = CalculateKeyValue(
```

```
        GOOGLE_CLIENT_SECRET_REMOTING_HOST,
        STRINGIZE_NO_EXPANSION(GOOGLE_CLIENT_SECRET_REMOTING_HOST),
        NULL,
        default_client_secret,
        environment.get(),
        command_line);
  }

  std::string api_key() const { return api_key_; }
#if defined(OS_IOS)
  void set_api_key(const std::string& api_key) { api_key_ = api_key; }
#endif
  std::string api_key_non_stable() const { return api_key_non_stable_; }

  std::string GetClientID(OAuth2Client client) const {
    DCHECK_LT(client, CLIENT_NUM_ITEMS);
    return client_ids_[client];
  }

#if defined(OS_IOS)
  void SetClientID(OAuth2Client client, const std::string& client_id) {
    client_ids_[client] = client_id;
  }
#endif

  std::string GetClientSecret(OAuth2Client client) const {
    DCHECK_LT(client, CLIENT_NUM_ITEMS);
    return client_secrets_[client];
  }

#if defined(OS_IOS)
  void SetClientSecret(OAuth2Client client, const std::string& client_secret) {
    client_secrets_[client] = client_secret;
  }
#endif

  std::string GetSpdyProxyAuthValue() {
#if defined(SPDY_PROXY_AUTH_VALUE)
    return SPDY_PROXY_AUTH_VALUE;
#else
    return std::string();
#endif
  }

 private:
  // Gets a value for a key.  In priority order, this will be the value
  // provided via a command-line switch, the value provided via an
  // environment variable, or finally a value baked into the build.
  // |command_line_switch| may be NULL. Official Google Chrome builds will not
```

```
  // use the value provided by an environment variable.
  static std::string CalculateKeyValue(const char* baked_in_value,
                                       const char* environment_variable_name,
                                       const char* command_line_switch,
                                       const std::string& default_if_unset,
                                       base::Environment* environment,
                                       base::CommandLine* command_line) {
    std::string key_value = baked_in_value;
    std::string temp;
#if defined(OS_MACOSX)
    // macOS and iOS can also override the API key with a value from the
    // Info.plist.
    temp = ::google_apis::GetAPIKeyFromInfoPlist(environment_variable_name);
    if (!temp.empty()) {
      key_value = temp;
      VLOG(1) << "Overriding API key " << environment_variable_name
              << " with value " << key_value << " from Info.plist.";
    }
#endif

#if !defined(GOOGLE_CHROME_BUILD)
    // Don't allow using the environment to override API keys for official
    // Google Chrome builds. There have been reports of mangled environments
    // affecting users (crbug.com/710575).
    if (environment->GetVar(environment_variable_name, &temp)) {
      key_value = temp;
      VLOG(1) << "Overriding API key " << environment_variable_name
              << " with value " << key_value << " from environment variable.";
    }
#endif

    if (command_line_switch && command_line->HasSwitch(command_line_switch)) {
      key_value = command_line->GetSwitchValueASCII(command_line_switch);
      VLOG(1) << "Overriding API key " << environment_variable_name
              << " with value " << key_value << " from command-line switch.";
    }

    if (key_value == DUMMY_API_TOKEN) {
#if defined(GOOGLE_CHROME_BUILD)
      // No key should be unset in an official build except the
      // GOOGLE_DEFAULT_* keys.  The default keys don't trigger this
      // check as their "unset" value is not DUMMY_API_TOKEN.
      CHECK(false);
#endif
      if (default_if_unset.size() > 0) {
        VLOG(1) << "Using default value \"" << default_if_unset
                << "\" for API key " << environment_variable_name;
        key_value = default_if_unset;
      }
```

```
286       }
287
288       // This should remain a debug-only log.
289       DVLOG(1) << "API key " << environment_variable_name << "=" << key_value;
290
291       return key_value;
292     }
293
294     std::string api_key_;
295     std::string api_key_non_stable_;
296     std::string client_ids_[CLIENT_NUM_ITEMS];
297     std::string client_secrets_[CLIENT_NUM_ITEMS];
298   };
299
300   static base::LazyInstance<APIKeyCache>::DestructorAtExit g_api_key_cache =
301       LAZY_INSTANCE_INITIALIZER;
302
303   bool HasKeysConfigured() {
304     if (GetAPIKey() == DUMMY_API_TOKEN)
305       return false;
306
307     for (size_t client_id = 0; client_id < CLIENT_NUM_ITEMS; ++client_id) {
308       OAuth2Client client = static_cast<OAuth2Client>(client_id);
309       if (GetOAuth2ClientID(client) == DUMMY_API_TOKEN ||
310           GetOAuth2ClientSecret(client) == DUMMY_API_TOKEN) {
311         return false;
312       }
313     }
314
315     return true;
316   }
317
318   std::string GetAPIKey() {
319     return g_api_key_cache.Get().api_key();
320   }
321
322   std::string GetNonStableAPIKey() {
323     return g_api_key_cache.Get().api_key_non_stable();
324   }
325
326   #if defined(OS_IOS)
327   void SetAPIKey(const std::string& api_key) {
328     g_api_key_cache.Get().set_api_key(api_key);
329   }
330   #endif
331
332   std::string GetOAuth2ClientID(OAuth2Client client) {
333     return g_api_key_cache.Get().GetClientID(client);
334   }
```

```
335
336 std::string GetOAuth2ClientSecret(OAuth2Client client) {
337   return g_api_key_cache.Get().GetClientSecret(client);
338 }
339
340 #if defined(OS_IOS)
341 void SetOAuth2ClientID(OAuth2Client client, const std::string& client_id) {
342   g_api_key_cache.Get().SetClientID(client, client_id);
343 }
344
345 void SetOAuth2ClientSecret(OAuth2Client client,
346                            const std::string& client_secret) {
347   g_api_key_cache.Get().SetClientSecret(client, client_secret);
348 }
349 #endif
350
351 std::string GetSpdyProxyAuthValue() {
352   return g_api_key_cache.Get().GetSpdyProxyAuthValue();
353 }
354
355 bool IsGoogleChromeAPIKeyUsed() {
356 #if defined(GOOGLE_CHROME_BUILD) || defined(USE_OFFICIAL_GOOGLE_API_KEYS)
357   return true;
358 #else
359   return false;
360 #endif
361 }
362
363 }  // namespace google_apis
```

Powered by Gitiles | Privacy | Terms

txt json