Migration to IdentityManager **PUBLIC @chromium**

Last updated: December, 2018
Tracking bug: crbug.com/796544
Tracking sheet:

Goal

Migrate usage of all signin classes that will become part of the Identity Service implementation to instead be usage of IdentityManager.

Out of Scope

In this phase, we are not migrating the codebase to use the Identity Service, either directly or by backing IdentityManager by the Identity Service. Those projects are targeted for later stages.

Accessing IdentityManager

Like the classes that it replaces, IdentityManager is a KeyedService. It can be accessed as follows:

- In //chrome: Via IdentityManagerFactory::GetForProfile().
- In //ios/chrome: Via IdentityManagerFactory::GetForBrowserState().
- In //ios/web_view: Via ios_web_view::WebViewIdentityManagerFactory::GetForBrowserState().
- In //components: Via dependency injection (i.e., however the classes whose usage it is replacing were being injected).

Like SigninManager(Base) and ProfileOAuth2TokenService, IdentityManager is null in incognito mode. Hence, checks for either of those objects being null can be replaced by checks for IdentityManager being null.

Conversion of SigninManager(Base) Usage

Tracking bug

SigninManagerBase::Observer

The interface is replaced by IdentityManager::Observer with the corresponding method mapping:

- GoogleSigninFailed → OnPrimaryAccountSigninFailed
  - This is only implemented by SyncEngine, TestSigninManagerObserver, SigninTracker and AboutSigninInternals
- GoogleSigninSucceeded → OnPrimaryAccountSet
- GoogleSignedOut → OnPrimaryAccountCleared
- GoogleSigninSucceededWithPassword → TODO
  - This is only implemented by PasswordStoreSigninNotifierImpl; probably better to implement and port ourselves instead of delegating to contractors

SigninManagerBase API

SigninManagerBase is the class that manages the state of the primary Google account (the account blessed for synchronisation). It is the super class of SigninManager and is usable on Chrome and Chrome OS. Here are conversion strategies for its various APIs:

- Initialize / IsInitialized
  - Should only be called from tests / factory; in production code, the service returned by the KeyedServiceFactory should already have been initialized, so there is no need to call those methods directly
- RegisterProfilePrefs / RegisterPrefs
  - Should only be called from tests / factory; in production code, the preferences are registered as part of the Profile creation, so there is no need to call those methods directly
- Shutdown()
  - Should only be called from tests / KeyedService infrastructure; in production code, the Shutdown method will be called as part of Profile destruction
- IsSigninAllowed
  - Code in chrome/ should read the preference directly prefs::kSigninAllowed
- GetAuthenticatedAccountInfo()
  - IdentityManager::GetPrimaryAccountInfo()
- GetAuthenticatedAccountId()
  - IdentityManager::GetPrimaryAccountId()
- IsAuthenticated()
  - IdentityManager::HasPrimaryAccount()
- AuthInProgress()
  - PrimaryAccountMutator::LegacyIsPrimaryAccountAuthInProgress()
- AddObserver/RemoveObserver
  - Port the code to implement IdentityManager::Observer and call IdentityManager::AddObserver / IdentityManager::RemoveObserver respectively
- AddSigninDiagnosticsObserver/RemoveSigninDiagnosticsObserver

    - Port the code to implement IdentityManager::DiagnosticsObserver and call IdentityManager::AddDiagnosticsObserver / IdentityManager::RemoveDiagnosticsObserver respectively
- signin_client()
    - TODO?
    - Should not be necessary
- RegisterOnShutdownCallback()
    - TODO?
    - Should not be necessary

The following methods are reserved to tests:
- SetAuthenticatedAccountInfo
    - Use MakePrimaryAccountAvailable() from identity_test_utils.h

SigninManager API

SigninManager is the class that manages the state of the primary Google account (the account blessed for synchronisation). It is unavailable on ChromeOS, but for all other platforms, SigninManagerFactory returns an instance of SigninManager, not SigninManagerBase. Here are conversion strategies for its various APIs:
- IsUsernameAllowedByPolicy / IsAllowedUsername
    - identity::LegacyIsUsernameAllowedByPatternFromPrefs()
- FromSigninManagerBase
    - Implementation detail to convert a SigninManagerBase* to SigninManager*; no need to port as there is no IdentityManagerBase*
- StartSignInWithRefreshToken
    - PrimaryAccountMutator::LegacyStartSigninWithRefreshTokenForPrimaryAccount()
- CopyCredentialsFrom
    - TODO https://crbug.com/889902
- SignOut(source_metric, delete_metric)
    - PrimaryAccountMutator::ClearPrimaryAccount(PrimaryAccountMutator::ClearAccountsAction::kDefault, source_metric, delete_metric)
- SignOutAndKeepAllAccounts(source_metric, delete_metric)
    - PrimaryAccountMutator::ClearPrimaryAccount(PrimaryAccountMutator::ClearAccountsAction::kKeepAll, source_metric, delete_metric)
- SignOutAndRemoveAllAccounts(source_metric, delete_metric)
    - PrimaryAccountMutator::ClearPrimaryAccount(PrimaryAccountMutator::ClearAccountsAction::kRemoveAll, source_metric, delete_metric)
- Initialize
    - Should only be called from tests / factory; in production code, the service returned by the KeyedServiceFactory should already have been initialized, so there is no need to call those methods directly

- Shutdown
  - Should only be called from tests / KeyedService infrastructure; in production code, the Shutdown method will be called as part of Profile destruction
- IsSigninAllowed
  - Only used in //chrome. Convert as outlined here.
- MergeSigninCredentialInCookieJar
  - TODO https://crbug.com/889902
- CompletePendingSignin
  - PrimaryAccountMutator::LegacyCompletePendingPrimaryAccountSignin()
- OnExternalSigninCompleted
  - PrimaryAccountMutator::SetPrimaryAccount()
- GetAccountIdForAuthInProgress
  - PrimaryAccountMutator::LegacyPrimaryAccountForAuthInProgress().account_id
- GetGaiaIdForAuthInProgress
  - PrimaryAccountMutator::LegacyPrimaryAccountForAuthInProgress().gaia
- GetUsernameForAuthInProgress
  - PrimaryAccountMutator::LegacyPrimaryAccountForAuthInProgress().email
- ~~DisableOneClickSignIn~~
  - ~~TODO? https://crbug.com/889908~~
  - ~~Sets the preference prefs::kReverseAutologinEnabled that is never read. Probably obsolete, should be removed.~~
- ~~ProhibitSignout~~
  - ~~TODO? https://crbug.com/889903~~
- ~~IsSignoutProhibited~~
  - ~~TODO? https://crbug.com/889903~~

Conversion of ProfileOAuth2TokenService Usage

Tracking bug

ProfileOAuth2TokenService is the class that manages the set of Google accounts for which this Profile has OAuth2 refresh tokens. It allows the user to interact with those refresh tokens and to fetch access tokens for those accounts.

Interacting with Access Tokens

- OAuth2TokenService::StartRequest(): If the requests are being made for the primary account only, use PrimaryAccountAccessTokenFetcher. Otherwise use AccessTokenFetcher.
  - How to tell whether requests for being made for the primary account only? Determine where the account ID being passed to O2TS::StartRequest() is coming from. If it only comes from SigninManagerBase::GetAuthenticatedAccount*() or

    IdentityManager::GetPrimaryAccountInfo(), then the requests are being made for the primary account only.
    - Example of migration to PAATF
    - Example of migration to ATF
- OAuth2TokenService::InvalidateAccessToken(): Use IdentityManager::RemoveAccessTokenFromCache().
    - Example

Interacting with Refresh Tokens

- OAuth2TokenService::GetAccounts(): Use IdentityManager::GetAccountsWithRefreshTokens().
- OAuth2TokenService::RefreshTokenIsAvailable(): Use IdentityManager::HasAccountWithRefreshToken().
- OAuth2TokenService::RefreshTokenHasError(): Use IdentityManager::HasAccountWithRefreshTokenInPersistentErrorState().
- OAuth2TokenService::GetAuthError(): Use IdentityManager::GetErrorStateOfRefreshTokenForAccount().

Porting OAuth2TokenService::Observer implementations:

- OnRefreshTokenAvailable(): Use IdentityManager::Observer::OnRefreshTokenUpdatedForAccount().
- OnRefreshTokenRevoked(): Use IdentityManager::Observer::OnRefreshTokenRemovedForAccount().
- OnRefreshTokensLoaded(): Use IdentityManager::Observer::OnRefreshTokensLoaded().

Missing APIs

TODO: Link to tracking bug and mention that bugs blocking tracking bug cover APIs known to be missing and needed. If you encounter another, add it as a bug blocking the tracking bug and CC {blundell, sdefresne}@chromium.org.

Out of Scope

We are not converting other subclasses of OAuth2TokenService, e.g., DeviceOAuth2TokenService.

Converting Tests

When converting a feature, both its production code and its tests should get converted to use only IdentityManager rather than directly using SigninManager and ProfileOAuth2TokenService. In general, we recommend separation of concerns:

- Convert the production code, making the minimal test changes necessary to have them continue to build and pass.

- Fully convert the tests, eliminating their usage of SigninManager and ProfileOAuth2TokenService.

The key infrastructure for converting tests is IdentityTestEnvironment, which provides mechanisms for constructing and interacting with IdentityManager objects in test contexts.

When converting a test, there are two questions to answer first:

- Does the test interact with fakes or with the real objects?
- Does the test construct the objects with which it interacts or does it obtain them (and potentially inject fake implementations of them) from the Profile (or in //ios, the ChromeBrowserState)?
    - We call the former **standalone tests**, and the latter **Profile-based tests**.

Converting standalone tests that interact with fakes

To convert incrementally (e.g., if an IdentityManager instance now needs to be passed into production code from the test): construct an IdentityTestEnvironment ivar in the test, passing it all of the backing objects that the test is currently using (i.e., FakeSigninManager and FakeProfileOAuth2TokenService). You can then access IdentityTestEnvironment::identity_manager(). See this CL for an example of such incremental conversion.

Once all test usage has been converted, you can eliminate the tests' knowledge of FakeSigninManager and FakePO2TS, constructing IdentityTestEnvironment via its no-parameters constructor.

Hints on most common recipes for converting test usage:

- You might encounter issues with the test abusing an email address as an account ID; in that case you will need to first clean up the test to properly disambiguate email addresses and account IDs. See this CL for a complex example of such cleanup, and if you have any questions about this issue, contact blundell@chromium.org.
- Using SigninManager(Base) to sign the user in/set the authenticated account: Use SetPrimaryAccount().
- Using SigninManager and PO2TS to sign the user in with a refresh token: Use MakePrimaryAccountAvailable().
    - For example, you can use MakePrimaryAccountAvailable(email) to replace flows like the following:
        - account_id = account_tracker_service->SeedAccountInfo(gaia, email);
        - token_service->UpdateCredentials(account_id, …);
        - signin_manager->SignIn(gaia, email, …);
- Signing the user out: Use ClearPrimaryAccount()

- Adding secondary accounts via AccountTrackerService (potentially with refresh tokens via PO2TS): Use MakeAccountAvailable()
- Interacting directly with refresh tokens for the primary/secondary accounts via PO2TS (e.g., to update or revoke credentials): use the various *RefreshToken*() APIs
- Responding to production requests for access tokens via PO2TS: Use the appropriate variant of WaitForAccessTokenRequestIfNecessaryAndRespondWithToken()

Converting standalone tests that interact with real objects

So far we have not seen any instances in this category. If you find such an instance, reach out to {blundell, sdefresne}@chromium.org so that we can prioritize developing a solution for this case.

Converting Profile-based tests that interact with fakes

These tests inject the fakes into the Profile via testing factories and use the IdentityManager instance that is constructed by its (production) factory. IdentityTestEnvironment cannot be constructed directly in these tests, as by default it constructs its own IdentityManager instance. However, it can be used via IdentityTestEnvironmentProfileAdaptor, which glues an IdentityTestEnvironment instance to the Profile via the fakes that are used by the Profile. The specifics of how IdentityTestEnvironmentProfileAdaptor is instantiated depend on how exactly the TestingProfile is instantiated by the test:

- TestingProfile::Builder: See an example here.
- TestingProfileManager: See an example here.
- Mechanisms that supply the set of testing factories to some opaque builder of the TestingProfile (e.g., a superclass): See an example here.

For advice on how to convert usage of the fakes, see the advice in the section above.

Converting Profile-based tests that interact with real objects

In this case, you should simply be able to change the tests to interact with the IdentityManager instance that is accessible via the Profile or BrowserState.

Converting ChromeBrowserState-based tests

The approach is conceptually the same as that of converting Profile-based tests, and similar infrastructure is present.