Vareiko Foundation

Comprehensive Usage Guide

Version: 1.0.1

Date: 2026-02-28

Zenject-first architecture for fast, stable Unity project starts.

Foundation Usage Guide (1.0+)

This is a practical, end-to-end guide for using com.vareiko.foundation as a starter architecture in Unity.

1. What You Get

The package provides:

1. deterministic bootstrap and service composition (ProjectContext + FoundationProjectInstaller);
1. scene-local composition (SceneContext + FoundationSceneInstaller);
1. modular runtime services (save/settings, analytics, backend, monetization, push, attribution, UI, observability);
1. adapter-oriented integrations (safe null/simulated providers + bridge entry points);
1. built-in runtime tests and validation tooling.

Use it as your baseline layer, then add game-specific domain modules above it.

2. Minimal Integration (Fresh Project)

1. Add ProjectContext to your bootstrap scene.
2. Attach FoundationProjectInstaller.
3. Add SceneContext to gameplay scenes.
4. Attach FoundationSceneInstaller.
5. Create and assign these configs first:
6. EnvironmentConfig
7. ObservabilityConfig
8. SaveSecurityConfig
9. AutosaveConfig

For the first run keep optional providers in fallback mode:

1. backend: None
1. IAP: None or Simulated
1. ads: None or Simulated
1. push: None or Simulated
1. attribution: None

3. Installation Dependencies

Required:

1. `com.cysharp.unitask`
1. `net.bobbo.extenject`
1. `com.unity.inputsystem`

Optional production dependencies:

1. Unity IAP (`UNITY_PURCHASING`)
1. host push SDK + `FOUNDATION_UNITY_NOTIFICATIONS`
1. PlayFab SDK + `PLAYFAB_SDK`

4. How Runtime Is Composed

At project startup, `FoundationRuntimeInstaller.InstallProjectServices(...)` installs modules in a strict order (time/common/env/app/bootstrap/config/.../observability).

The order is documented in `Documentation~/ARCHITECTURE.md`.

`FoundationProjectInstaller` exposes serialized config slots and forwards them to the runtime installer.

5. First Play Mode Checklist

1. Enter Play Mode in bootstrap scene.
2. Verify zero compile errors.
3. Verify no startup validation `Error`.
4. Verify boot transitions to expected app state.
5. Verify diagnostics snapshot is available via `IDiagnosticsService`.

6. Core Service Usage Patterns

Save/Load

```
using Cysharp.Threading.Tasks;
using Vareiko.Foundation.Save;
using Zenject;

public sealed class ProfileFacade
{
    private readonly ISaveService _save;

    [Inject]
    public ProfileFacade(ISaveService save)
    {
        _save = save;
    }

    public UniTask SaveProfileAsync(PlayerProfile profile)
    {
        return _save.SaveAsync("main", "profile", profile);
    }

    public UniTask<PlayerProfile> LoadProfileAsync()
```

```
    {
        return _save.LoadAsync("main", "profile", new PlayerProfile());
    }
}
```

Settings

```
using Vareiko.Foundation.Settings;
using Zenject;

public sealed class AudioSettingsFacade
{
    private readonly ISettingsService _settings;

    [Inject]
    public AudioSettingsFacade(ISettingsService settings)
    {
        _settings = settings;
    }

    public void SetMusicVolume(float volume01)
    {
        var next = _settings.Current;
        next.MusicVolume = volume01;
        _settings.Apply(next, saveImmediately: true);
    }
}
```

Scene Flow

```
using Cysharp.Threading.Tasks;
using UnityEngine.SceneManagement;
using Vareiko.Foundation.SceneFlow;
using Zenject;

public sealed class SceneRouter
{
    private readonly ISceneFlowService _sceneFlow;

    [Inject]
    public SceneRouter(ISceneFlowService sceneFlow)
    {
        _sceneFlow = sceneFlow;
    }

    public UniTask OpenGameplayAsync()
    {
        return _sceneFlow.LoadSceneAsync("Gameplay", LoadSceneMode.Single, setActive: true);
    }
}
```

Feature Flags

```
using Cysharp.Threading.Tasks;
using Vareiko.Foundation.Features;
using Zenject;

public sealed class FeatureGate
{
    private readonly IFeatureFlagService _flags;

    [Inject]
```

```
    public FeatureGate(IFeatureFlagService flags)
    {
        _flags = flags;
    }

    public async UniTask<bool> IsNewEconomyEnabledAsync()
    {
        await _flags.RefreshAsync();
        return _flags.IsEnabled("economy.v2", false);
    }
}
```

7. Custom Bootstrap Tasks

Add game-specific boot logic by implementing `IBootstrapTask`.

```
using System.Threading;
using Cysharp.Threading.Tasks;
using Vareiko.Foundation.Bootstrap;
using Vareiko.Foundation.Save;
using Zenject;

public sealed class LoadProfileBootstrapTask : IBootstrapTask
{
    private readonly ISaveService _save;

    public int Order => 100;
    public string Name => "Load Player Profile";

    [Inject]
    public LoadProfileBootstrapTask(ISaveService save)
    {
        _save = save;
    }

    public async UniTask ExecuteAsync(CancellationToken cancellationToken)
    {
        _ = await _save.LoadAsync("main", "profile", new PlayerProfile(), cancellationToken);
    }
}
```

Bind it in your project installer or a domain installer.

8. Monetization and Comms Stack

IAP

Main contract: `IInAppPurchaseService`

1. `InitializeAsync()`
1. `GetCatalogAsync()`
1. `PurchaseAsync(productId)`
1. `RestorePurchasesAsync()`

Ads

Main contract: `IAdsService`

1 `InitializeAsync()`

1 `LoadAsync(placementId)`

1 `ShowAsync(placementId)`

Use `IMonetizationPolicyService` to gate ad/purchase actions before calling providers.

Push

Main contract: `IPushNotificationService`

1 `InitializeAsync()`

1 `RequestPermissionAsync()`

1 `GetDeviceTokenAsync()`

1 `SubscribeAsync(topic)`

1 `UnsubscribeAsync(topic)`

9. External Bridge Integrations

External Ads Bridge

For non-Unity mediation SDKs:

```
using Cysharp.Threading.Tasks;
using Vareiko.Foundation.Ads;

public static class AdsSdkBridgeInstaller
{
    public static void Install()
    {
        ExternalAdsBridge.SetInitializeHandler(_ => UniTask.FromResult(AdsInitializeResult.Succeed()));
        ExternalAdsBridge.SetLoadHandler((placementId, _) => UniTask.FromResult(AdLoadResult.Succeed(pl
        ExternalAdsBridge.SetShowHandler((placementId, _) => UniTask.FromResult(AdShowResult.Succeed(pl
    }
}
```

External Attribution Bridge

For AppsFlyer/Adjust/other attribution SDKs:

```
using System.Collections.Generic;
using Cysharp.Threading.Tasks;
using Vareiko.Foundation.Attribution;

public static class AttributionSdkBridgeInstaller
{
    public static void Install()
    {
        ExternalAttributionBridge.SetTrackEventHandler((eventName, properties, _) =>
        {
            IReadOnlyDictionary<string, string> payload = properties;
            // Forward payload to your attribution SDK here.
            return UniTask.FromResult(AttributionTrackResult.Succeed(eventName));
        });

        ExternalAttributionBridge.SetTrackRevenueHandler((data, _) =>
        {
```

```
            // Forward revenue to your attribution SDK here.
            return UniTask.FromResult(AttributionRevenueTrackResult.Succeed(data.ProductId, data.Curren
        });
    }
}
```

At shutdown/domain reload call:

1. `ExternalAdsBridge.ClearHandlers()`
1. `ExternalAttributionBridge.ClearHandlers()`

10. Observability and Diagnostics

Use:

1. `IDiagnosticsService` for live snapshot access;
1. `IDiagnosticsSnapshotExportService.ExportAsync(label)` for QA/support exports.

Monitor key signals in `SignalBus` for:

1. startup validation;
1. monetization decisions/failures;
1. push permission/topic flow;
1. attribution and ads bridge failures.

11. Testing Strategy

Recommended levels:

1. Runtime unit tests per module (service contracts + provider mapping).
2. Integration smoke tests for bootstrap + scene flow + save/settings.
3. PlayMode smoke for scene wiring and audio/UI critical path.

Use package test assembly `Tests/Runtime` as the default pattern for new modules.

12. Release Checklist

1. Run `Tools/Vareiko/Foundation/Validate Project`.
2. Ensure package version and changelog are aligned.
3. Ensure no compile errors on clean import.
4. Ensure required runtime tests pass.
5. Tag release (`vX.Y.Z`) after commit.

13. Common Pitfalls

1. Missing provider define/sdks with production provider selected.
1. Enabling consent-required modules without loaded consent state.
1. Registering bridge handlers too late (after provider init).

1. Skipping `SceneContext`/`FoundationSceneInstaller` in gameplay scenes.

If you follow the starter flow and keep provider-heavy modules in fallback mode first, integration is usually stable from day one.