SDK_PATTERNS_REFERENCE.md

Version: 1.0.0 **Last Updated:** 2024-12-19 **Status:** Approved **Dependencies:** None (Reference Document)

Change Log

- v1.0.0 (2024-12-19): Consolidated from spacetimedb_rust and csharp pattern docs

---

4.1 SpacetimeDB Rust Patterns

Reducer Requirements

✅ CORRECT Reducer Signatures

```rust
```

```rust
#[spacetimedb::reducer]
pub fn my_reducer(ctx: &ReducerContext) -> Result<(), String> {
    // Simple reducer
    Ok(())
}

#[spacetimedb::reducer]
pub fn with_params(
    ctx: &ReducerContext,
    param1: u64,
    param2: String,
) -> Result<(), String> {
    // Parameters must implement SpacetimeType
    Ok(())
}
```

❌ INCORRECT Patterns

```rust
```

```rust
// Wrong: Mutable references not allowed
#[spacetimedb::reducer]
pub fn bad_reducer(
    ctx: &ReducerContext,
    session: &mut MiningSession,  // ❌ Invalid
) -> Result<(), String>

// Wrong: Complex borrowed types
#[spacetimedb::reducer]
pub fn bad_reducer2(
    ctx: &ReducerContext,
    data: &ComplexStruct,  // ❌ Invalid
) -> Result<(), String>

// Wrong: Async not supported
#[spacetimedb::reducer]
pub async fn bad_reducer3(  // ❌ No async
    ctx: &ReducerContext,
) -> Result<(), String>
```

Table Operations

Finding Records

```rust
```

```
// ✅ CORRECT: Find with supported types
let player = ctx.db.player()
    .identity().find(&identity);  // Identity supported
let account = ctx.db.account()
    .username().find(&username);   // String supported

// ❌ WRONG: Complex types don't implement FilterableValue
let world = ctx.db.world()
    .world_coords().find(&coords); // WorldCoords not supported

// ✅ CORRECT: Use iteration for complex types
let world = ctx.db.world()
    .iter()
    .find(|w| w.world_coords == coords);
```

Delete Operations

```rust
// ❌ WRONG: delete() expects owned value
let session: &PlayerSession = get_session();
ctx.db.player_session().delete(session); // Error: expected owned

// ✅ CORRECT: Clone or move
ctx.db.player_session().delete(session.clone());
```

Update Pattern

```rust
```

```
// SpacetimeDB has no in-place updates
// ✅ CORRECT: Delete + Insert
let mut updated = original.clone();
updated.field = new_value;

ctx.db.my_table().delete(original);
ctx.db.my_table().insert(updated);
```

Type Requirements

Hash Trait for HashMap Keys

```rust
// ❌ WRONG: Missing Hash
#[derive(SpacetimeType, Clone, PartialEq, Eq)]
pub enum MyEnum { A, B, C }

let mut map = HashMap::new();
map.insert(MyEnum::A, 42);  // Error: Hash not implemented

// ✅ CORRECT: Include Hash
#[derive(SpacetimeType, Clone, PartialEq, Eq, Hash)]
pub enum MyEnum { A, B, C }
```

State Management

Static State Pattern

```rust
```

```rust
// For complex state not in tables
static GAME_STATE: OnceLock<Mutex<GameState>> = OnceLock::new();

fn get_game_state() -> &'static Mutex<GameState> {
    GAME_STATE.get_or_init(|| Mutex::new(GameState::default()))
}

#[spacetimedb::reducer]
pub fn use_state(ctx: &ReducerContext) -> Result<(), String> {
    let mut state = get_game_state().lock().unwrap();
    state.update();
    Ok(())
}
```

4.2 SpacetimeDB C# Patterns

Connection Patterns

✅ CORRECT Connection Building

```csharp
var conn = DbConnection.Builder()
    .WithUri("http://localhost:3000")
    .WithModuleName("my_module")
    .OnConnect((connection, identity, token) => { })
    .OnConnectError(error => { })
    .OnDisconnect((connection, error) => { })
    .Build();
```

❌ INCORRECT Patterns

```csharp
// Wrong: These methods don't exist
conn.Connect(host);          // ❌ No Connect method
conn.RemoteTables.Player;    // ❌ No RemoteTables
DbConnection.Builder()
    .WithCredentials(token);  // ❌ No WithCredentials
```

Table Access

✅ CORRECT Iteration

```csharp
// Use Iter() for enumeration
foreach (var player in conn.Db.Player.Iter())
{
    if (player.Identity == targetIdentity)
        return player;
}

// Use index accessors for unique columns
var player = conn.Db.Player.Identity.Find(identity);
```

❌ INCORRECT Patterns

```csharp
```

```
// Wrong: No LINQ support
var players = conn.Db.Player.Where(p => p.Active);  // ❌

// Wrong: Direct enumeration
foreach (var player in conn.Db.Player) { }  // ❌ Need Iter()

// Wrong: Find with predicate
var player = conn.Db.Player.Find(p => p.Name == "Bob"); // ❌
```

Event Handlers

Reducer Events

```csharp
// ✅ CORRECT: ReducerEventContext + direct arguments
conn.Reducers.OnStartMining += HandleStartMining;

private void HandleStartMining(
    ReducerEventContext ctx,
    ulong orbId  // Direct arguments, not wrapped
)
{
    // Note: NO .Status or .Message on ctx
    Debug.Log($"Mining started: {orbId}");
}
```

Table Events

```csharp
```

```csharp
// ✅ CORRECT: EventContext + row data
conn.Db.Player.OnInsert += (EventContext ctx, Player player) =>
{
    Debug.Log($"Player joined: {player.Name}");
};

conn.Db.Player.OnUpdate += (ctx, oldPlayer, newPlayer) =>
{
    Debug.Log($"Player updated: {newPlayer.Name}");
};
```

Unity Integration

MonoBehaviour Pattern

csharp

```csharp
public class SpacetimeManager : MonoBehaviour
{
    private DbConnection conn;

    void Start()
    {
        InitConnection();
        SubscribeEvents();
    }

    void OnDestroy()
    {
        // ✅ ALWAYS unsubscribe
        if (conn != null)
        {
            conn.Db.Player.OnInsert -= HandlePlayerInsert;
            conn.Reducers.OnStartMining -= HandleMining;
        }
    }
}
```

Coroutine Connection

```csharp
```

```
IEnumerator ConnectToSpacetimeDB()
{
    bool connected = false;

    conn = DbConnection.Builder()
        .WithUri(serverUrl)
        .OnConnect((connection, identity, token) => {
            connected = true;
        })
        .Build();

    // Wait for connection
    float timeout = 5f;
    while (!connected && timeout > 0)
    {
        timeout -= Time.deltaTime;
        yield return null;
    }

    if (connected)
        Debug.Log("Connected!");
    else
        Debug.LogError("Connection timeout!");
}
```

4.3 Unity Integration Patterns

Singleton Pattern

```csharp
// Game uses singleton pattern consistently
public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }
    public DbConnection conn { get; private set; }

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }
}

// Usage
GameManager.Instance.conn.Reducers.StartMining(orbId);
```

Caching Pattern

```csharp
public class DataCache : MonoBehaviour
{
    private Dictionary<ulong, WavePacketOrb> orbCache = new();

    void Start()
    {
        var conn = GameManager.Instance.conn;

        conn.Db.WavePacketOrb.OnInsert += (ctx, orb) =>
            orbCache[orb.OrbId] = orb;

        conn.Db.WavePacketOrb.OnDelete += (ctx, orb) =>
            orbCache.Remove(orb.OrbId);
    }

    public WavePacketOrb GetOrb(ulong id) =>
        orbCache.TryGetValue(id, out var orb) ? orb : null;
}
```

Object Pooling

```csharp
```

```
public class OrbPoolManager : MonoBehaviour
{
    private Queue<GameObject> pool = new();
    public GameObject orbPrefab;

    public GameObject GetOrb()
    {
        if (pool.Count > 0)
        {
            var orb = pool.Dequeue();
            orb.SetActive(true);
            return orb;
        }
        return Instantiate(orbPrefab);
    }

    public void ReturnOrb(GameObject orb)
    {
        orb.SetActive(false);
        pool.Enqueue(orb);
    }
}
```

4.4 Common Pitfalls & Solutions

Database Pitfalls

1. No Async/Await in Reducers

```rust
// ❌ WRONG
#[spacetimedb::reducer]
pub async fn bad_reducer(ctx: &ReducerContext) -> Result<(), String>

// ✅ CORRECT - Synchronous only
#[spacetimedb::reducer]
pub fn good_reducer(ctx: &ReducerContext) -> Result<(), String>
```

2. Partial Moves

```rust
// ❌ WRONG - Partial move of String field
let player = Player {
    name: logged_out.name,  // String moved
};
ctx.db.logged_out_player().delete(logged_out); // Error

// ✅ CORRECT - Clone the field
let player = Player {
    name: logged_out.name.clone(),
};
```

Client Pitfalls

1. Null Reference on Tables

```csharp
```

```
// ❌ WRONG - No null check
var player = conn.Db.Player.Identity.Find(identity);
player.Name = "New";  // Might be null!

// ✅ CORRECT - Always check
var player = conn.Db.Player.Identity.Find(identity);
if (player != null)
{
    // Safe to use
}
```

2. Memory Leaks from Events

```csharp
// ❌ WRONG - Not unsubscribing
void Start()
{
    conn.Db.Player.OnInsert += HandleInsert;
}
// Memory leak - handler never removed

// ✅ CORRECT - Unsubscribe in OnDestroy
void OnDestroy()
{
    if (conn != null)
        conn.Db.Player.OnInsert -= HandleInsert;
}
```

3. Immediate Data Assumption

```csharp
// ❌ WRONG - Data not ready immediately
conn = DbConnection.Builder().Build();
var players = conn.Db.Player.Iter(); // Empty!

// ✅ CORRECT - Wait for sync
bool dataReady = false;
conn.OnConnect += (connection, identity, token) => {
    StartCoroutine(WaitForInitialSync());
};

IEnumerator WaitForInitialSync()
{
    yield return new WaitForSeconds(0.5f);
    dataReady = true;
}
```

4.5 Testing Strategies

Local Testing Setup

```bash
```

```
# Start local SpacetimeDB
spacetime start

# Generate code
spacetime generate --lang=rust --out-dir=src/autogen
spacetime generate --lang=csharp --out-dir=Assets/Scripts/autogen

# Publish module
spacetime publish my_module --clear-database

# Watch logs
spacetime logs my_module -f
```

Debug Logging

Rust Side

```rust
log::info!("=== REDUCER START ===");
log::debug!("Player: {:?}, Orb: {}", player, orb_id);
log::warn!("Unexpected state");
log::error!("Critical: {}", error);
```

Unity Side

```csharp
```

```
#if UNITY_EDITOR
Debug.Log($"[MINING] Started on orb {orbId}");
Debug.LogWarning($"[NETWORK] High latency: {ping}ms");
Debug.LogError($"[ERROR] Failed to connect: {error}");
#endif
```

Performance Testing

```csharp
```

```csharp
public class PerformanceMonitor : MonoBehaviour
{
    void Start()
    {
        StartCoroutine(MonitorPerformance());
    }

    IEnumerator MonitorPerformance()
    {
        while (true)
        {
            yield return new WaitForSeconds(1f);

            var orbCount = conn.Db.WavePacketOrb.Count();
            var playerCount = conn.Db.Player.Count();
            var fps = 1f / Time.deltaTime;

            Debug.Log($"[PERF] FPS:{fps:F1} Orbs:{orbCount} Players:{playerCount}");
        }
    }
}
```

Mock Testing

```csharp
```

```
#if UNITY_EDITOR
public static class MockData
{
    public static WavePacketOrb CreateMockOrb(ulong id)
    {
        return new WavePacketOrb
        {
            OrbId = id,
            Position = Random.insideUnitSphere * 100f,
            Frequency = (FrequencyBand)Random.Range(0, 6),
            PacketsRemaining = 100
        };
    }
}
#endif
```