# Data Repository Assembly – Functional Description

## Overview

The DataRepo assembly provides a flexible, thread-safe, in-memory key-value storage system designed to manage different typed data values. It supports dynamic creation of keys, type enforcement, access control, and efficient retrieval and modification of stored values. The system is implemented as a reusable module for integration into larger applications that require structured, permission-controlled, and strongly-typed data management.

## Core Features

### 1. Concurrent & Thread-Safe Storage

Utilizes a ConcurrentDictionary<string, RepoValueBase> to store all key-value pairs, ensuring safe, multi-threaded access for high-performance environments.

### 2. Dynamic Key Creation

Allows creation of new keys at runtime with the CreateKey method. Supports multiple data types via the RepoValueType enum:  
- Integer (RepoValueInteger)  
- Float (RepoValueFloat)  
- String (RepoValueString)  
- Boolean (RepoValueBool)  
- JSON (RepoValueJson)  
- UTC Timestamp (RepoValueTimestampUtc)  
Also provides bulk creation through CreateKeyMulti.

### 3. Type Enforcement

Optional type enforcement ensures stored values match the declared type, preventing accidental mismatches.

### 4. Access Control

Supports per-key read, write, and delete passwords, enabling fine-grained permission control for sensitive data.

### 5. Value Writing & Reading

Methods are provided for setting and retrieving values while respecting type and access rules. Supports creating and writing keys in a single operation via CreateAndWriteKey.

### 6. Structured Change Tracking

Maintains a LastStructureChangeUtc timestamp to indicate when the repository structure changes, useful for synchronization and caching mechanisms.

### 7. Extensible Design

Values are represented by a polymorphic base class RepoValueBase with specialized derived classes for each supported type, facilitating future expansion with new data types or custom storage behaviors.

## Example Use Cases

- Configuration Management – Store application-wide settings with access restrictions.  
- Real-time Data Exchange – Share typed data safely between different threads or components.  
- IoT / Industrial Control Systems – Maintain a live, in-memory store for sensor values, states, or commands.  
- Temporary Caching – Hold runtime values without persisting them to disk.

## Technical Summary

Language: C# (.NET)  
Thread Safety: Yes (via ConcurrentDictionary)  
Storage Model: In-memory key-value store with typed entries  
Security: Optional per-key password protection  
Extensibility: Easy to add new value types via RepoValueBase inheritance