# **ASCDataAccessLibrary: Getting Started Guide**

This guide provides a quick introduction to working with the ASCDataAccessLibrary, a powerful .NET library for managing Azure Table Storage operations with enhanced capabilities for large data fields and simplified session management.

## **Table of Contents**

1. [Installation](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#installation)
2. [Basic Concepts](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#basic-concepts)
3. [Working with Data Access](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#working-with-data-access)
4. [Using the Session Manager](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#using-the-session-manager)
5. [Examples](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#examples)
6. [Best Practices](https://claude.ai/chat/7c7b6411-a0aa-47ce-b40c-ec7a6ec98508#best-practices)

## **Installation**

Install the package from NuGet:

Install-Package ASCDataAccessLibrary

Or using the .NET CLI:

dotnet add package ASCDataAccessLibrary

## **Basic Concepts**

### **Key Components**

* **DataAccess<T>**: Core component for CRUD operations with Azure Table Storage
* **Session**: Session management utility for maintaining stateful web applications
* **TableEntityBase**: Base class that automatically handles large data fields by chunking
* **ITableExtra**: Interface that your model classes must implement

### **Creating a Model**

Your model classes need to inherit from TableEntityBase and implement ITableExtra:

using AZTableStorage.Models;  
  
public class Customer : TableEntityBase, ITableExtra  
{  
 // Primary key field - stored in PartitionKey  
 public string CompanyId  
 {  
 get { return this.PartitionKey; }  
 set { this.PartitionKey = value; }  
 }  
   
 // Unique identifier - stored in RowKey  
 public string CustomerId  
 {  
 get { return this.RowKey; }  
 set { this.RowKey = value; }  
 }  
   
 // Your custom properties  
 public string Name { get; set; }  
 public string Email { get; set; }  
 public string PhoneNumber { get; set; }  
 public string LargeDataField { get; set; } // Will be automatically chunked if needed  
   
 // Required by ITableExtra  
 public string TableReference => "Customers";  
   
 // Required by ITableExtra - defines the unique identifier  
 public string GetIDValue() => this.CustomerId;  
}

## **Working with Data Access**

### **Initialization**

Create a new instance of DataAccess<T> for each entity type:

// Initialize data access with your Azure Storage credentials  
var customerAccess = new DataAccess<Customer>("your-azure-account-name", "your-azure-account-key");

**Important**: Create a separate DataAccess instance for each table you need to work with.

### **Basic CRUD Operations**

#### **Create or Update a Single Entity**

// Create a new customer  
var newCustomer = new Customer  
{  
 CompanyId = "company123",  
 CustomerId = "cust456",  
 Name = "Acme Corporation",  
 Email = "[contact@acme.com](mailto:contact@acme.com)"  
};  
  
// Insert or replace the entity  
customerAccess.ManageData(newCustomer);  
  
// To update with a merge operation instead  
customerAccess.ManageData(newCustomer, TableOperationType.InsertOrMerge);

#### **Retrieve a Single Entity**

// Get by RowKey  
var customer = customerAccess.GetRowObject("cust456");  
  
// Get with a custom query  
var customerByEmail = customerAccess.GetRowObject("Email", ComparisonTypes.eq, "[contact@acme.com](mailto:contact@acme.com)");

#### **Retrieve Multiple Entities**

// Get all entities with the same partition key  
var companyCustomers = customerAccess.GetCollection("company123");  
  
// Get all entities in the table  
var allCustomers = customerAccess.GetAllTableData();

#### **Advanced Querying**

// Create a complex query with multiple conditions  
var queryItems = new List<DBQueryItem>  
{  
 new DBQueryItem   
 {   
 FieldName = "Name",   
 FieldValue = "Acme",   
 HowToCompare = ComparisonTypes.eq   
 },  
 new DBQueryItem   
 {   
 FieldName = "Email",   
 FieldValue = "contact@",   
 HowToCompare = ComparisonTypes.ne   
 }  
};  
  
// Get customers matching all conditions (using AND)  
var customers = customerAccess.GetCollection(queryItems, QueryCombineStyle.and);

#### **Batch Operations**

// Update multiple entities at once  
List<Customer> customersToUpdate = GetCustomersForUpdate();  
customerAccess.BatchUpdateList(customersToUpdate);  
  
// Delete multiple entities  
List<Customer> customersToDelete = GetCustomersForDeletion();  
customerAccess.BatchUpdateList(customersToDelete, TableOperationType.Delete);

## **Using the Session Manager**

### **Initialization**

// Initialize a new session  
var session = new Session("your-azure-account-name", "your-azure-account-key", "user-session-id");

### **Working with Session Data**

// Store a value in session  
session["UserName"] = "John Doe";  
session["LastLogin"] = DateTime.Now.ToString();  
  
// Read a value from session  
string userName = session["UserName"];  
  
// Commit changes to database  
session.CommitData();

### **Session Management**

// Restart/clear a session  
session.RestartSession();  
  
// Clean up stale session data  
session.CleanSessionData();  
  
// Get stale session data (e.g., for abandoned carts)  
var staleSessions = session.GetStaleSessions();

## **Examples**

### **Complete Customer Management Example**

using AZTableStorage.Data;  
using AZTableStorage.Models;  
using System;  
using System.Collections.Generic;  
  
// Initialize data access  
var customerAccess = new DataAccess<Customer>("your-account", "your-key");  
  
// Create a new customer  
var customer = new Customer  
{  
 CompanyId = "company001",  
 CustomerId = Guid.NewGuid().ToString(),  
 Name = "New Customer Inc.",  
 Email = "[info@newcustomer.com](mailto:info@newcustomer.com)",  
 PhoneNumber = "555-123-4567"  
};  
  
// Save to database  
customerAccess.ManageData(customer);  
Console.WriteLine($"Customer saved with ID: {customer.CustomerId}");  
  
// Update the customer  
customer.PhoneNumber = "555-987-6543";  
customerAccess.ManageData(customer);  
Console.WriteLine("Customer updated");  
  
// Retrieve the customer  
var retrievedCustomer = customerAccess.GetRowObject(customer.CustomerId);  
Console.WriteLine($"Retrieved: {retrievedCustomer.Name}");  
  
// Delete the customer  
customerAccess.ManageData(retrievedCustomer, TableOperationType.Delete);  
Console.WriteLine("Customer deleted");

### **Session Management Example**

using AZTableStorage.Data;  
using System;  
  
// Initialize session  
var userSessionId = Guid.NewGuid().ToString();  
var session = new Session("your-account", "your-key", userSessionId);  
  
// Store shopping cart data  
session["CartItems"] = "Product1,Product2,Product3";  
session["CartTotal"] = "149.99";  
session["CartLastUpdated"] = DateTime.Now.ToString();  
  
// Later in the code...  
Console.WriteLine($"Cart contains: {session["CartItems"]}");  
Console.WriteLine($"Cart total: ${session["CartTotal"]}");  
  
// Commit changes before ending session  
session.CommitData();  
  
// In a new request, reload the same session  
var reloadedSession = new Session("your-account", "your-key", userSessionId);  
Console.WriteLine($"Reloaded cart total: ${reloadedSession["CartTotal"]}");  
  
// When the user completes checkout or leaves  
reloadedSession.RestartSession();

## **Best Practices**

1. **Create Separate Instances**: Always create a new DataAccess instance for each table you're working with.
2. **Leverage Batch Operations**: When updating multiple entities, use BatchUpdateList for better performance.
3. **Handle Large Data**: Let TableEntityBase handle chunking of large text fields automatically - no need for custom code.
4. **Commit Session Data**: Always ensure session data is committed before ending user sessions.
5. **Clean Up Regularly**: Implement a periodic job to call CleanSessionData() to remove stale sessions.
6. **Use Appropriate Comparisons**: When querying, use the most specific ComparisonTypes to improve performance.
7. **Partition Key Design**: Design your PartitionKey strategy carefully to ensure good performance with Azure Table Storage.
8. **Type Safety**: Leverage the generic nature of DataAccess<T> for type safety in your application.

This library simplifies Azure Table Storage operations while adding powerful features like automatic field chunking for large data, complex querying, and built-in session management capabilities.