**Dynamic Mapping System Documentation**

Table of Contents

[**1. System Architecture** 2](#_Toc200925443)

[**Overview** 2](#_Toc200925444)

[**Key Components** 3](#_Toc200925445)

[**2. Repository Structure** 3](#_Toc200925446)

[**3. Key Classes and Methods** 3](#_Toc200925447)

[**Core Components** 3](#_Toc200925448)

[**Mapping Profile Interface** 4](#_Toc200925449)

[**Example Profile (Google)** 4](#_Toc200925450)

[**4. Extending the System** 5](#_Toc200925451)

[**Adding New Mappings (Step-by-Step)** 5](#_Toc200925452)

[a. Create new models: 5](#_Toc200925453)

[b. Create mapping profile: 5](#_Toc200925454)

[c. Register profile at startup: 6](#_Toc200925455)

[**Extension Points** 6](#_Toc200925456)

[**5. Error Handling** 6](#_Toc200925457)

[**Error Scenarios** 6](#_Toc200925458)

[**6. Usage Examples** 7](#_Toc200925459)

[**Basic Mapping** 7](#_Toc200925460)

[**Error Handling** 7](#_Toc200925461)

[**7. Assumptions** 8](#_Toc200925462)

[**8. Limitations** 8](#_Toc200925463)

[**9. Best Practices** 8](#_Toc200925464)

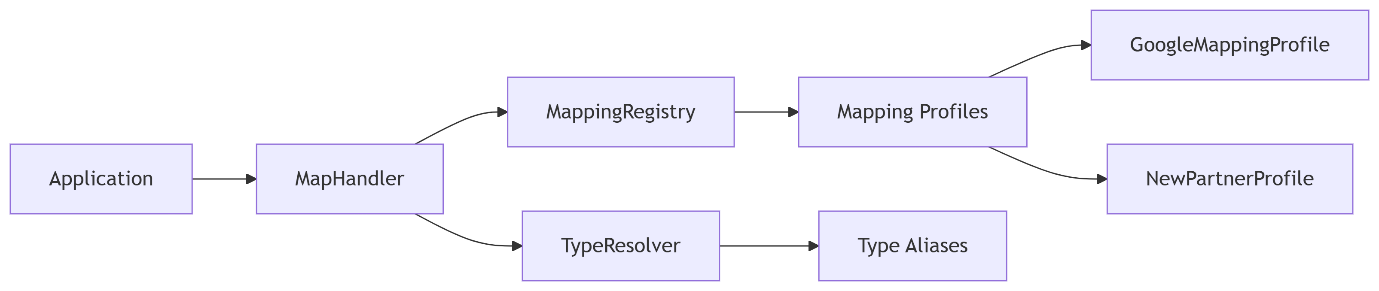
[**10. Future Enhancements** 8](#_Toc200925465)

# **1. System Architecture**

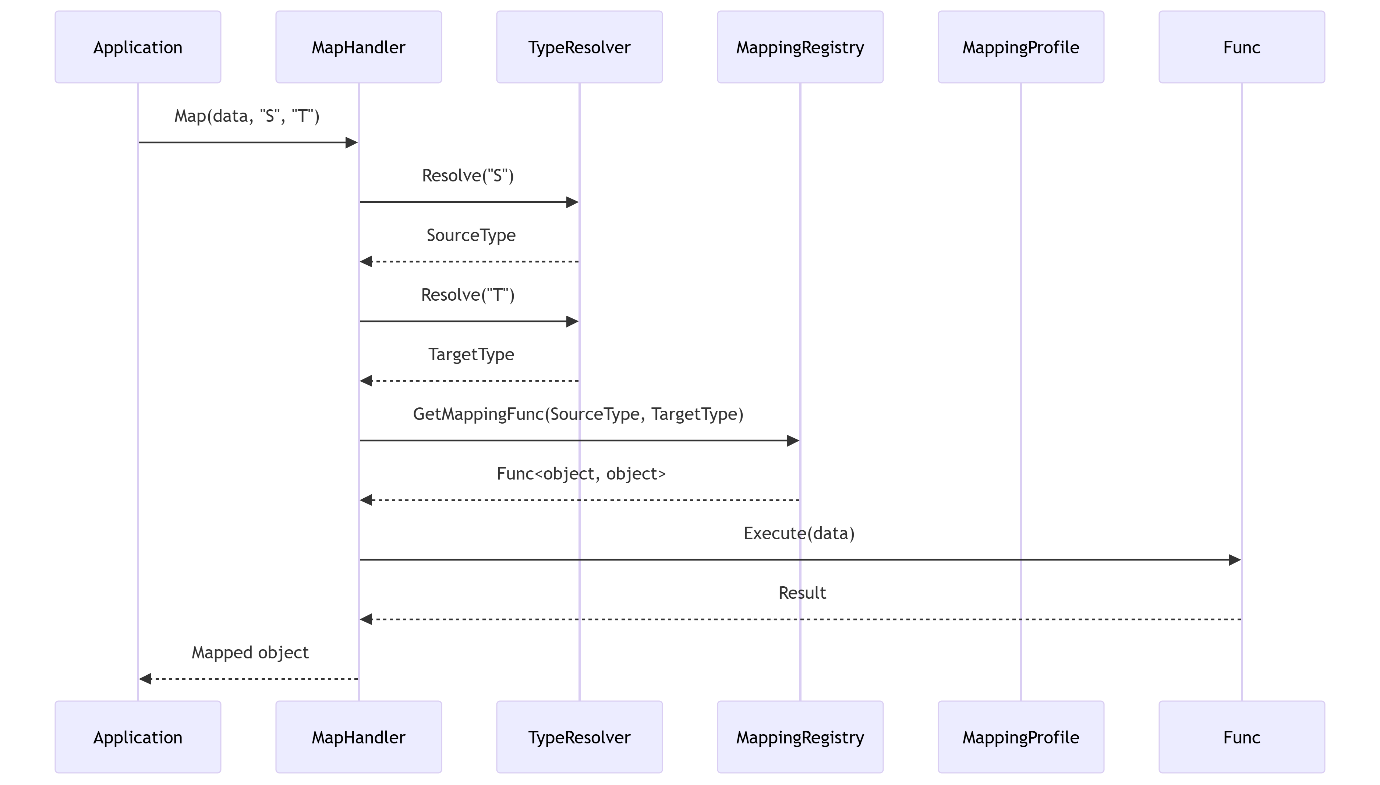
## **Overview**

The Dynamic Mapping System provides an extensible framework for converting data between internal DIRS21 models and external partner models. The system follows a modular design with clear separation of concerns, enabling easy maintenance and extension.

**Architecture Diagram**



**Data Flow**



## **Key Components**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| MapHandler | Orchestrates mapping process and error handling |
| TypeResolver | Manages type aliases and resolves them to actual types |
| MappingRegistry | Stores and retrieves mapping functions between types |
| MappingProfile | Defines mappings and type aliases for specific partners (extensible) |
| Models | Contains internal DIRS21 models and external partner models |

# **2. Repository Structure**

DynamicMappingSystem/

|── src/

| |── DynamicMappingSystem.Core/ # Core mapping logic

| |── DynamicMappingSystem.Models/ # Data models

│ └── DynamicMappingSystem.ConsoleDemo # Demo application

|── tests/

│ └── DynamicMappingSystem.Tests/ # Unit tests

|── docs/

| └── Architecture Documentation # Design documentation

|── README.md # Getting started guide

└── DynamicMappingSystem.sln # Solution file

# **3. Key Classes and Methods**

## **Core Components**

1. MapHandler**(Core Service)**

* Map(object data, string sourceType, string targetType): Main entry point for mapping operations
  + Validates input parameters
  + Resolves types using TypeResolver
  + Retrieves mapping function from MappingRegistry
  + Executes mapping with error handling

1. TypeResolver**(Core Service)**

* RegisterAlias(string alias, Type type): Associates type alias with .NET type
* Resolve(string alias): Retrieves Type object for given alias

1. MappingRegistry**(Core Service)**

* Register<TSource, TTarget>(Func<TSource, TTarget> mappingFunc): Stores mapping function
* GetMappingFunc(Type sourceType, Type targetType): Retrieves mapping function

## **Mapping Profile Interface**

IMappingProfile

public interface IMappingProfile

{

void ConfigureMappings(IMappingRegistry registry);

void ConfigureTypeAliases(ITypeResolver typeResolver);

}

## **Example Profile (Google)**

GoogleMappingProfile

public class GoogleMappingProfile : IMappingProfile

{

public void ConfigureMappings(IMappingRegistry registry)

{

registry.Register<Reservation, GoogleReservation>(MapToGoogle);

registry.Register<GoogleReservation, Reservation>(MapFromGoogle);

}

public void ConfigureTypeAliases(ITypeResolver typeResolver)

{

typeResolver.RegisterAlias("Model.Reservation", typeof(Reservation));

typeResolver.RegisterAlias("Google.Reservation", typeof(GoogleReservation));

}

// Mapping methods implementation...

}

# **4. Extending the System**

## **Adding New Mappings (Step-by-Step)**

### a. Create new models:

// In Models/NewPartner/NewPartnerReservation.cs

namespace DIRS21.API.Models.NewPartner

{

public class NewPartnerReservation

{

public string PartnerId { get; set; }

public DateTime Arrival { get; set; }

// Additional properties

}

}

### b. Create mapping profile:

// In Core/MappingProfiles/NewPartnerMappingProfile.cs

public class NewPartnerMappingProfile : IMappingProfile

{

public void ConfigureMappings(IMappingRegistry registry)

{

registry.Register<Reservation, NewPartnerReservation>(MapToNewPartner);

registry.Register<NewPartnerReservation, Reservation>(MapFromNewPartner);

}

public void ConfigureTypeAliases(ITypeResolver typeResolver)

{

typeResolver.RegisterAlias("NewPartner.Reservation", typeof(NewPartnerReservation));

}

private NewPartnerReservation MapToNewPartner(Reservation source)

{

// Custom mapping logic

}

private Reservation MapFromNewPartner(NewPartnerReservation source)

{

// Custom mapping logic

}

}

### c. Register profile at startup:

// In application initialization

var newProfile = new NewPartnerMappingProfile();

newProfile.ConfigureTypeAliases(typeResolver);

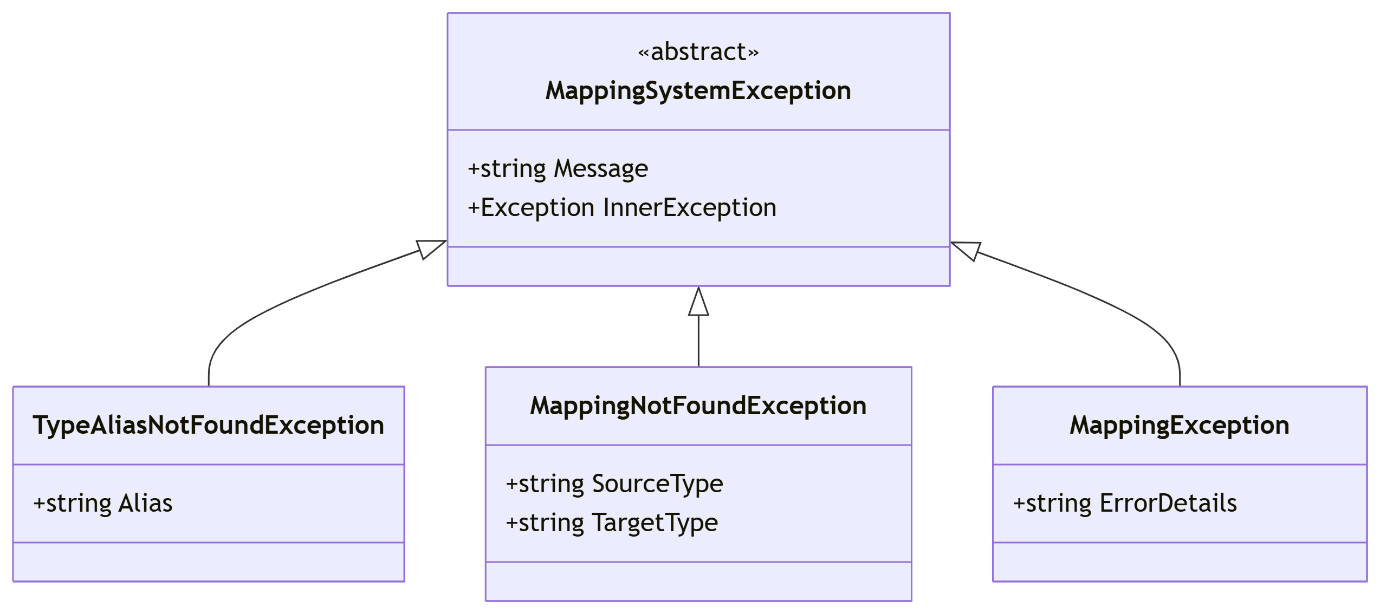
newProfile.ConfigureMappings(mappingRegistry);

## **Extension Points**

1. **Add new mapping profiles** for additional partners
2. **Extend existing profiles** with new mapping methods
3. **Implement custom type resolution** by extending ITypeResolver
4. **Add pre/post processing hooks** by decorating mapping functions

# **5. Error Handling**

**Exception Hierarchy**



## **Error Scenarios**

|  |  |  |
| --- | --- | --- |
| **Error Case** | **Exception Type** | **Handling Recommendation** |
| Unregistered type alias | TypeAliasNotFoundException | Register alias in mapping profile |
| Missing mapping function | MappingNotFoundException | Add mapping to profile |
| Invalid data type | ArgumentException | Verify input data |
| Mapping execution error | MappingException | Check mapping logic |
| Null input data | ArgumentNullException | Validate before mapping |

# **6. Usage Examples**

## **Basic Mapping**

// Initialize system

var typeResolver = new TypeResolver();

var mappingRegistry = new MappingRegistry();

var profile = new GoogleMappingProfile();

profile.ConfigureTypeAliases(typeResolver);

profile.ConfigureMappings(mappingRegistry);

var mapHandler = new MapHandler(mappingRegistry, typeResolver);

// Execute mapping

var googleRes = mapHandler.Map(

dirsReservation,

"Model.Reservation",

"Google.Reservation") as GoogleReservation;

## **Error Handling**

try

{

return mapHandler.Map(data, sourceType, targetType);

}

catch (TypeAliasNotFoundException ex)

{

logger.LogError($"Missing type alias: {ex.Alias}");

// Recovery logic

}

catch (MappingException ex)

{

logger.LogError($"Mapping failed: {ex.InnerException?.Message}");

// Fallback strategy

}

# **7. Assumptions**

1. **Type Registration**: All types must be registered via profiles before mapping
2. **Immutable Configuration**: Mapping configuration is static after initialization
3. **Simple Object Graphs**: Handles moderate complexity objects (1-2 levels nesting)
4. **Synchronous Processing**: Designed for synchronous operations only
5. **Trusted Input**: Input data is assumed to be structurally valid
6. **Single Direction**: Mappings are explicitly defined in both directions

# **8. Limitations**

1. **Collection Mapping**: Does not automatically map collections (requires explicit handling)
2. **Circular References**: Cannot handle objects with circular references
3. **Performance Scaling**: Not optimized for high-volume scenarios
4. **Complex Type Conversion**: Requires manual implementation of custom conversions
5. **Versioning**: No built-in support for model versioning
6. **Thread Safety**: Not thread-safe during configuration changes
7. **Partial Updates**: No support for partial/delta mapping

# **9. Best Practices**

1. **Profile Organization**: Create separate profiles for each partner
2. **Idempotent Mappings**: Ensure mappings produce consistent results
3. **Error Logging**: Log inner exceptions for mapping failures
4. **Validation**: Validate input before mapping operations
5. **Test Coverage**: Maintain 1:1 mapping test for all profiles
6. **Alias Consistency**: Use consistent naming for type aliases (Partner.Model)

# **10. Future Enhancements**

1. **Automatic Collection Mapping**: Add support for list-to-list conversions
2. **Async Support**: Implement async mapping pipeline
3. **Caching**: Add mapping result caching
4. **Validation Hooks**: Integrate with FluentValidation
5. **Dynamic Profiles**: Load profiles from configuration
6. **Metrics**: Add performance monitoring hooks