Data Sets

A data set consists of four key elements: a Target, a Target Map, a Module, and a Wing. All four elements must be defined for the data set to function.

The *Target* is the actual definition of the data type that is being imported. The map *Target Map* relates the type definition to the SQL database and ensures easy access using NHibernate, while the *Module* and *Bulk* Import Mapper

MONAHRQ uses implementations of `IBulkMapper` to determine how data of a given Target type should be loaded into the database. For Target types, a subclass `DatasetRecordBulkInsertMapper<T>` is used; this type provides the logic necessary to map the `DatasetRecord.Dataset` CLR property to the `Dataset_Id` SQL column.

If additional custom mapping logic is desired, create a new implementation of `DatasetRecordBulkInsertMapper<T>` and override the `DatasetRecord.CreateBulkInsertMapper<T>(…)` method. Refer to the following MONAHRQ types for examples:

- `Monahrq.Wing.Discharge.DischargeTargetBulkInsertMapper<T>`
  Maps diagnosis and procedure codes for various Target types defined by `Monahrq.Wing.Discharge`.
- `Monahrq.Infrastructure.Entities.Domain.Categories.CategoryBulkInsertMapper<T>`
  Maps the `CategoryType` SQL column to the CLR type name of the entity

Bulk Import User Interface

If desired, the default Host User interface for importing a dataset may be overridden. This may be accomplished by defining a new UI context type that extends `DatasetContextBase` and a new wizard step collection that extends `StepCollection<YourDatasetContext>`.

The `StepCollection` implementation is invoked by an event listener in the Module definition. See `Monahrq.Wing.HospitalCompare.HospitalCompareModule` for a working example. Note that the `RECORD_KEY` defined in the HospitalCompare Wing's `WizardContext` class is the same GUID used in `HospitalCompareTarget`.

Dataset Win provide additional information about the data set and expose the data set to the rest of MONAHRQ.

Creating a new Data Set

Creation of a new Data Set requires the following components:

1. *Target Definition*: to define your data type
2. *Target Map*: to explain how MONAHRQ should store your data type in SQL
3. *Module Definition*: to provide additional installation and upgrade logic, and to provide column name hints during the bulk import process
4. *Dataset Wing Definition*: to advertise the data set Target type to MONAHRQ

Optionally, the following components may also be developed:

1. *Bulk Import Mapper*: to change the way specific columns are handled during the import process
2. *Bulk Import User Interface*: to present a custom wizard UI to the Host User in place of the standard data import interface

Target Definition

The foundation of a data set is the Target – a CLR representation of a single data set row. The Target contains all columns that are imported from the source data along with some metadata to help MONAHRQ interpret and validate data during the import process.

The bare minimum requirements for a Target are as follows:

- The type must inherit the `DatasetRecord`[1] class
- The type must be decorated with the `WingTarget` attribute
- The type must declare one or more properties decorated with the `WingTargetElement` attribute

Inheriting from the DatasetRecord Class

The `DatasetRecord` class inherits from `Entity<T>` which implements `IEntity<T>`[2]. These types provide functionality common to all entity types in MONAHRQ including support for bulk loading, field validation, and property change notifications. The `DatasetRecord` class includes an `Id` property, which is assumed to be a primary key and identity column in SQL.

The WingTarget Attribute

Each Target must be decorated with a `WingTarget`[3] attribute which identifies the name and description of the data set, a unique identifying GUID, and additional metadata.

```
[WingTarget(name: "My Custom Data Set Name",
    targetGuid: "71a308bb-05f8-4e18-84e6-8db6a7ecc7dc",
    description: "My Custom Data Set Description",
    isReferenceTarget: false,
    isTrendingEnabled: false,
    displayOrder: 0)]
public class HospitalSpendingByClaimTypeTarget : DatasetRecord
{
```

Figure 1 Example of the WingTarget attribute decorating a Target

The GUID selected for this attribute will be used to reference this data set type in other area of MONAHRQ, such as during report generation.

[1] Refer to the `Monahrq.Infrastructure.Data.Extensibility.ContentManagement.Records` namespace
[2] Both types are in the `Monahrq.Infrastructure.Entities.Domain` namespace
[3] Most attributes used in this document are defined in `Monahrq.Infrastructure.Core.Attributes`

Declaring Properties

Properties should be defined as standard CLR properties and decorated with the `WingTargetElement`[3] attribute.

```
[WingTargetElement("AverageSpendingPerEpisodeHospital",
    description: "Average Spending per Episode - Hospital",
    isRequired: true,
    ordinal: 4,
    longDecription: "The average amount of money spent per admission at
                     this hospital")]
public double AverageSpendingPerEpisodeHospital { get; set; }
```

Figure 2 Sample Property Declaration

The `Name` property of `WingTargetElement` must correspond exactly with the name of the property to prevent runtime errors. The `Description` property should be used for the full descriptive name of the property.

Data Validation

Validation is possible at the class level and at the property level using attributes derived from `ValidationAttribute`[4]. Several MONAHRQ-specific validation attributes are provided:

Attribute	Purpose	Applies To
`Monahrq.Infrastructure.ICDValidation`	Validates ICD9 and ICD10 codes found in commonly named properties	Class
`Monahrq.Infrastructure.NonEmptyList`	`IList` contains at least one element	`IList` properties
`Monahrq.Infrastructure.Numeric`	Value can be parsed as a double	Properties
`Monahrq.Infrastructure.RegexWarning`	Raises a warning if the regex pattern doesn't match	Properties
`Monahrq.Infrastructure.RejectIfAnyPropertyHasValue`	Suppresses rows that have the given value for any property	Class
`Monahrq.Infrastructure.RequiredWarning`	Raises a warning if the value wasn't provided	Properties
`Monahrq.Infrastructure.UniqueConstraintCheck, Monahrq.Sdk.ViewModels.UniqueAttribute`	Prevent duplicate values	Properties
`Monahrq.Sdk.RangeToCurrentYear`	Given value is between the given year and the current year	Properties

[4] `System.ComponentModel.DataAnnotations.ValidationAttribute`

Handling Enum Type Members

Enum type properties do not require any special handling in the Target definition; however, the enum types themselves must be decorated with the `WingScope`[3] attribute and enum members must be decorated with the `WingScopeValue`[3] attribute.

```
[WingScope("Claim Type", typeof(ClaimType))]
public enum ClaimType
{
    [WingScopeValue("Exclude", -1)]
    Exclude = -1,

    [WingScopeValue("Unknown", 0)]
    Unknown = 0,

    [WingScopeValue("HomeHealthAgency", 1, "Home Health Agency")]
    HomeHealthAgency = 1,
```

Figure 3 A sample enum type with WingScope and WingScopeValue attributes

Note: only integer values are currently supported

Using a Custom Bulk Import Mapper

`IBulkMapper` instances are used to transform source data before it is stored in the database. They are utilized in MONAHRQ to populate the ICD version column for discharge-related targets in the `Monahrq.Wing.Discharge` module, for XML Wing targets in `Monahrq.Wing.Dynamic`, and for some base data types.

If you require the functionality provided by `IBulkMapper`, you should override the `CreateBulkInsertMapper<T>(…)` method in your target type to return a new instance of your custom `IBulkMapper`.

For additional information about creating a bulk import mapper, see *Bulk Import Mapper*.

Performing In-Place Schema Upgrades

Since the schema for a target is defined in the Fluent NHibernate configuration of the *Target Map* and MONAHRQ's use of Fluent NHibernate does not support in-place upgrades. All logic for in-place upgrades must be implemented separately from the schema definition. There are two options for integrating such an upgrade into MONAHRQ:

1. As a core database schema upgrade. This is most appropriate for targets that are built-in to MONAHRQ. See ***Error! Reference source not found.*** for more information.
2. As a base data importer. This is most appropriate for .NET Wings that define their own targets. See ***Error! Reference source not found.***.

Target Map

The Fluent NHibernate mapping for your Target relates the CLR type definition to your desired SQL schema and is defined in a class that derives from `DatasetRecordMap<T>`[1], where `T : DatasetRecord`.

The specifics of how to use Fluent NHibernate are not within the scope of MONAHRQ documentation; for information, refer to the Fluent NHibernate documentation[5].

```
public class HospitalSpendingByClaimTypeTargetMap
        : DatasetRecordMap<HospitalSpendingByClaimTypeTarget>
{
    public HospitalSpendingByClaimTypeTargetMap()
    {
        this.Map(m => m.CmsProviderId).Not.Nullable().Length(12);
        this.Map(m => m.ClaimType).CustomType<ClaimType>();
```

Figure 4 Excerpt from sample DatasetRecordMap<T> implementation

Module Definition

A Wing module in MONAHRQ serves two key purposes:

- Advertise database installation and refresh logic to MONAHRQ
- Provide default column name hints for a target definition to be used when importing new data

Module Lifecycle

A module for a Data Set has the same startup sequence and lifecycle as other modules in MONAHRQ. See ***Error! Reference source not found.*** for more information.

Choosing a Base Class

All modules in MONAHRQ derive from the abstract `WingModule` class, but modules that specifically describe a Target should derive from the more specific `TargetedModuleBase<T>`; this class provides convenient references to the `WingTargetAttribute`[3] decorating your Target as well as `IDomainSessionFactoryProvider`.

If your module also provides measures and/or topics, you may wish to derive your module class from `TargetedModuleWithMeasuresAndTopics<T>`, which includes additional overrides for the installation of measures and measure topics. This is covered in greater depth in *The WingModule Attribute*

[5] https://github.com/jagregory/fluent-nhibernate/wiki/Getting-started

The WingModule Attribute

The module must be decorated with a `WingModuleAttribute`[3] which identifies the name and description of the module as well as a unique identifying GUID. This GUID differs from the GUID specified in the `WingTargetAttribute` and is used to uniquely identify the `WingModule` to MONAHRQ.

```
[WingModule(typeof(SampleModule),
    "{7D39A125-315D-47B0-BE69-E7D6DFB64BB3}",
    "Hospital Spending",
    "Hospital Spending Sample Module")]
public class SampleModule
    :  TargetedModuleWithMeasuresAndTopics<SampleTarget>
{
```

Figure 5 Sample WingModule Attribute

Refer to the `Subscribe()` method of the `Monahrq.Wing.Ahrq.AhrqModuleBase` module implementation for an example of how the `WingModule` GUID can be used.

Overriding Default Database Installation and Update Procedure

The default behavior for a module is to run the `Install()`, `InstallDb()`, `Update()`, and `UpdateDb()` functions on MONAHRQ startup only if the module has not been previously installed into the database. If this behavior is not appropriate for your implementation, it may be changed by overriding the protected `Reconcile()` method[6].

If you are implementing from `TargetedModuleBase<T>`, the four install and update methods mentioned above have an empty default implementation and may be overridden in any way.

If you are implementing from `TargetedModuleWithMeasuresAndTopics<T>`, the `InstallDb()` method calls the virtual `ImportMeasures()` and `ImportMeasureTopics()` methods.

Providing Default Column Name Hints

If the dataset being imported contains column names that differ from those of the `WingTargetElement` properties defined in your Target, you may override the method `OnApplyDatasetHints()` to tell MONAHRQ about their other name(s); call the `Target<T>(Expression<Func<T,object>>)` method to obtain a reference to the metadata object for a particular property, and call the `ApplyMappingHints` on that object to specify alternate column names.

```
protected override void OnApplyDatasetHints()
{
    Target<SampleTarget>(t => t.ClaimType)
        .ApplyMappingHints("Claim_Type");
    Target<SampleTarget>(t => t.CmsProviderId)
        .ApplyMappingHints("Provider_ID", "ID");
```

Figure 6 Sample OnApplyDatasetHints() override

[6] Refer to the base implementation in `WingModule` to determine what functionality should be replicated in your implementation

In this example, the ClaimType property may be imported using the column names "ClaimType" or "Claim_Type", and the CmsProviderId property may be using the column names "CmsProviderId", "Provider_ID", or "ID".

Column name hints are stored in the Hints column of the Wings_Elements database table. When a hostuser specifies a column mapping manually in MONAHRQ, that mapping is added to the value of the Hints column.

Adding Measures and Measure Topics to a Target

Modules that derive from TargetedModuleWithMeasuresAndTopics<T> may override the ImportMeasures() and ImportMeasureTopics() methods to define Measure Categories, Measure Topics, and individual Measures. For more information, refer to **Error! Reference source not found.**.

Bulk Import Mapper

MONAHRQ uses implementations of IBulkMapper to determine how data of a given Target type should be loaded into the database. For Target types, a subclass DatasetRecordBulkInsertMapper<T>[7] is used; this type provides the logic necessary to map the DatasetRecord.Dataset CLR property to the Dataset_Id SQL column.

If additional custom mapping logic is desired, create a new implementation of DatasetRecordBulkInsertMapper<T> and override the DatasetRecord.CreateBulkInsertMapper<T>(…) method. Refer to the following MONAHRQ types for examples:

- Monahrq.Wing.Discharge.DischargeTargetBulkInsertMapper<T>
  Maps diagnosis and procedure codes for various Target types defined by Monahrq.Wing.Discharge.
- Monahrq.Infrastructure.Entities.Domain.Categories.CategoryBulkInsertMapper<T>
  Maps the CategoryType SQL column to the CLR type name of the entity

Bulk Import User Interface

If desired, the default Host User interface for importing a dataset may be overridden. This may be accomplished by defining a new UI context type that extends DatasetContextBase[8] and a new wizard step collection that extends StepCollection<YourDatasetContext>[9].

The StepCollection implementation is invoked by an event listener in the Module definition. See Monahrq.Wing.HospitalCompare.HospitalCompareModule for a working example. Note that the RECORD_KEY defined in the HospitalCompare Wing's WizardContext class is the same GUID used in HospitalCompareTarget.

[7] See the Monahrq.Infrastructure.Data.Extensibility.ContentManagement.Records namespace
[8] See Monahrq.Datasets.Model.DatasetContextBase
[9] See Monahrq.Theme.Controls.Wizard.Models.StepCollection<T>

```
protected override void OnInitialize()
{
    base.OnInitialize();
    this.Subscribe();
}

private void Subscribe()
{
    Events.GetEvent<WizardStepsRequestEvent<DataTypeModel, Guid, int?>>()
        .Subscribe(this.OnWizardStepRequestEvent);
}

private void OnWizardStepRequestEvent(
    WizardStepsRequestEventArgs<DataTypeModel, Guid, int?> args)
{
    if (args.WingId == WingGUID)
        args.WizardSteps = FactoryWizardSteps(args.Data, args.ExistingDatasetId);
}
```

Figure 7 Overriding the WizardSteps collection

Dataset Wing Definition

The purpose of the Dataset Wing, an implementation of `IDatasetWing`, is to advertise the Target type and its metadata to MONAHRQ. If your Target type is decorated with `WingTargetAttribute`, the only requirements for creating a Wing are to inherit from `DatasetWing<T>`, where T is your Target type, and decorate the Wing with `DatasetWingExportAttribute`[3].

```
[DatasetWingExport]
public class HospitalSpendingByClaimTypeDatasetWing
    : DatasetWing<HospitalSpendingByClaimTypeTarget>
{
}
```

Figure 8 Entire IDatasetWing implementation

The Dataset Wing does not expose any other functionality or serve any other purpose.