Reports

In MONAHRQ, reports are interpreted data that has been written to `.js` file(s) by a Report Generator. Installed reports may be browsed using the "Reports" library, and reports available for a website are listed on the "Select Reports" tab for that website.

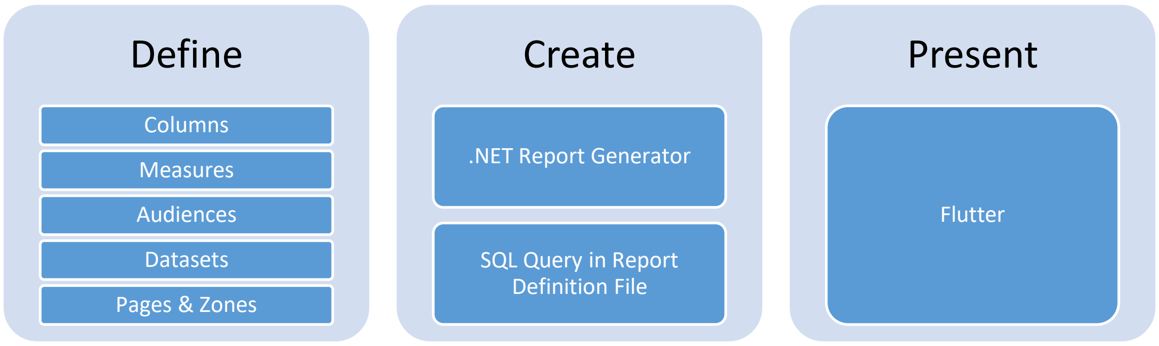

Figure 1 Logical groupings of report components

Creating a New Report

Creating new `.js` files to be consumed by the MONAHRQ-generated website requires two components to be defined:

1. Report Definition File
2. Report Generator[1]

This document does not describe how to render reports on the generated website. For information about rendering reports, please refer to the documentation for creating a new Flutter using the MONAHRQ Open Source Framework at https://github.com/AHRQ/MONAHRQ-Framework.

Report Definition File

The report definition file is an XML serialized `ReportManifest` instance[2]. This file describes the data contained in the report, information necessary to render the report, and includes a unique identifier for the report used to relate the report definition to the report generator.

Creating a Report Definition

The following describes the information that must be contained within a report definition file; for additional information about the capabilities of report definitions, please refer to the source code for the report definition type.

[1] When developing an XML Wing, the report generator is provided in the report definition as a SQL statement

[2] See `Monahrq.Infrastructure.Entities.Domain.Reports.ReportManifest` and `Monahrq.Infrastructure\Domain\ReportManifest.xsd`

XPath	Description	Example or Permitted Values
@RptId	Report GUID, which should match the GUID specified in the report definition[3]. This is not used as a unique identifier.	`"{E5C24EC4-6583-41D8-87A1-9F45B143819B}"`
@Name	The name of the report. This doubles as the unique identifier of the report.	`"My Sample Report"`
@Category	The report's category	`"Quality"` or `"Utilization"`
@PreviewImage	Optional path to an image of a rendered sample report	`"MySampleReportPreview.png"`
@IsTrending	Indicates whether this report displays trend data	`true` or `false`
@InterpretationText, @ShowInterpretationText	End User explanation for how the report should be interpreted	
.\Audiences	Specifies the intended audience of the report	`<Audience AudienceType="Professionals"/>`
.\Datasets	A listing of the datasets this report consumes. At least one dataset must be listed for the report to be available for use. This text is displayed in the footer of each report on the website.	`<Dataset Name="Hospital Spending by Claim Type"/>`

.\Filters	Options available for filtering the report output, displayed in the Report Filters tab; may also describe UI behaviors (e.g.: radio buttons)	`<Filter Type="Hospital">`
		`  <Values>`
		`    <FilterValue Name="Hospital Name" />`
		`    <FilterValue Name="Hospital Type" />`
		`    <FilterValue Name="Region" />`
		`    <FilterValue Name="County" />`
		`  </Values>`
		`</Filter>`

.\ReportAttributes	Selects elements to be displayed in the MONAHRQ UI for this report	HospitalFilters, DRGsDischargesFilters, ConditionsAndDiagnosisFilters, KeysForRatings, IncludedHospitals, ReportColumns, Display, CountyFilters
.\Columns	Lists measures to be displayed in the "Customize Site" > "Website Pages" tab of the Websites library	`<Column IsMeasure="true" MeasureCode="HS-01" Name="Average spending per episode - Hospital"/>`
.\WebsitePages	Places the report on pages in the generated MONAHRQ website	See below

For working examples, refer to the definitions for existing reports in the `Domain\Reports\Data` subdirectory of the MONAHRQ program directory.

Columns

Report columns are either static or dynamic; static columns are listed in the Columns collection while dynamic columns are determined by the report generator at runtime. A column may refer to a measure.

[3] Flutters made using the open source framework do not need to define the RptId attribute because they do not have generators

The list of columns included in a report is accessible in the MONAHRQ UI in the Websites library, under “Customize Site” > “Website Pages”.

A report implementing a dynamic column list, such as `HospitalComparison-icons.xml`, may list all possible columns in the Columns collection.

For an example of a report with static columns, refer to one of the Inpatient Utilization reports, such as `IPUtilizationDetail.xml`.

WebsitePages and WebsitePageZones

The WebsitePages collection lists all the pages where the report is displayed and controls whether the Host User is permitted to customize the report’s header or footer. A single report may define many WebsitePages (e.g.: one for consumers and another for professionals, or multiple pages for each audience). This information is used by the built-in content management system.

```
<WebsitePage Name="Compare Hospitals"
             Audience="Consumers"
             Path="app/products/consumer/hospitals/views/compare.html"
             Url="/hospitals/compare?ids=1,2"
             IsEditable="true">
  <WebsitePageZones>
    <WebsitePageZone Name="Header" CodePath=""/>
    <WebsitePageZone Name="Footer" CodePath=""/>
  </WebsitePageZones>
</WebsitePage>
```

Figure 2 Example WebsitePage declaration

The `Path` attribute describes the path to the page template in the local filesystem. `Url` describes the URL used in the generated website to refer to the page.

Deploying a Report Definition

To install a report definition, copy it to the `Domain\Reports\Data` subdirectory of the MONAHRQ program directory and restart MONAHRQ. The report will be installed on startup. The report’s `@Name` is used to uniquely identify it.

For an automatic deployment as part of the MONAHRQ solution build process, refer to the post-build actions of the `Monahrq.Wing.HospitalSpendingSample` project.

Once deployed, reports are added to the `Reports` database table (and other tables prefixed with `Reports_`), along with the complete manifest. Note that the `IsCustom` column of the `Reports` table is a misnomer; it actually indicates whether the report was defined in CLR code (`0`) or by an XML wing (`1`).

On startup, MONAHRQ compares the last modified timestamp of all report definition files to the `LastReportManifestUpdate` timestamp in the `Reports` table of the database; if the file is newer, the manifest in the database is updated. Otherwise, the manifest from the database is used.

Refer to `Monahrq.Reports.ReportsModule` for additional information regarding the loading and updating of individual reports.

Report Generator

The report generator is responsible for building reports from datasets and outputting `.js` files that may be consumed by the generated website. All report generators inherit from the `BaseReportGenerator` class and are decorated with `ReportGeneratorAttribute`[4] and `ExportAttribute`[5].

```
[Export(typeof(IReportGenerator))]
[ReportGenerator(
    // must match GUID from report XML definition
    reportIds: new[] {"E5C24EC4-6583-41D8-87A1-9F45B143819B"},
    moduleDependencies: new string[] { },
    datasetTargetDependencies: new[] { typeof(MyTargetType) }
    )]
public class MyReportGenerator : BaseReportGenerator
{
```

Figure 3 Sample report generator class header

Lifecycle Overview

Report generators are utilized at two times when MONAHRQ is running: at startup and during website generation.

At startup, while the splash screen is visible, the report generator is given a chance to initialize itself using the `InitGenerator()` method.

When the Host User starts the process of generating the website, `BaseReportGenerator`'s `GenerateReport(Website, PublishTask)` method is called, which calls `ValidateDependencies(Website, IList<ValidationResult>)` , followed by `RefreshRptDataObjects()`, `LoadReportData()`, and finally, `OutputDataFiles()`, as follows:

```
if (ValidateDependencies(website, validationResults))
{
    RefreshRptDataObjects();
    if (LoadReportData())
        OutputDataFiles();
}
```

Figure 4 Excerpt from BaseReportGenerator.GenerateReport(Website, PublishTask)

For more information about how unsuccessful results are handled, please refer to the `BaseReportGenerator` source code.

Initialization

Although overriding the `InitGenerator()` method is required, most report generators should not need to do anything during initialization. Some of MONAHRQ's built-in generators broadcast their initialization through a `MessageUpdateEvent` while others have an empty method body.

[4] Defined in the `Monahrq.Sdk.Generators` namespace
[5] Provided by MEF

Dependency Validation

`BaseReportGenerator`'s implementation of the `ValidateDependencies(…)` confirms that the output directory specified by the `Website.OutPutDirectory` property has been provided and that any modules listed as dependencies are initialized. This method does not validate that required datasets are available, so any override of the `ValidateDependencies()` method should perform its own check for datasets; refer to `HospitalSpendingReportGenerator` for an example of such a check.

Loading Report Data

Most of the information needed to load the report data is available in the base class's `CurrentWebsite` property, including Datasets and Measures selected by the Host User.

There are two approaches to loading data: either doing it all at once and committing to disk or memory in the `LoadReportData()` method, or doing nothing in the `LoadReportData()` method and doing everything in the `OutputDataFiles()` method. The approach taken does not matter to MONAHRQ as nothing happens between these two method calls.

Writing Data Files

The `OutputDataFiles()` method is responsible for writing the report data as `.js` files that will be read by the generated website. If your data is in a format that can be serialized to JSON[6] and small enough to fit in memory, `BaseReportGenerator` offers a helper method, `GenerateJsonFile(…)` to write your output file(s). Otherwise, you may need to manually write your output files. Refer to existing wings as well as the four generator applications for examples.

For information about where to place and how to consume your output files, refer to the open source Flutter documentation.

Execution Order

Report generators each advertise a priority through the `ExecutionOrder` property of the `ReportGeneratorAttribute`. Property values generally range from 1 to 10. Report generators with the same `ExecutionOrder` are executed in parallel with each other while reports with different `ExecutionOrder`s are executed starting with the lowest values.

User Interface

Reports are accessible in the UI from both the ***Error! Reference source not found.*** library and the "Select Reports" tab of the ***Error! Reference source not found.*** library.

Flutters: Rendering Reports in Generated Website

A "flutter" defines the layout, style, and flow of a report. Flutters are written using the AngularJS framework.

For more information about writing a flutter, please refer to the MONAHRQ Open Source Framework**Error! Bookmark not defined.** documentation at https://github.com/AHRQ/MONAHRQ-Framework.

For more information about how flutters are consumed, please refer to the `DynamicReportGenerator` class in the `Monahrq.Wing.Dynamic` source code.

[6] Using Newtonsoft's `JsonConvert.SerializeObject(object)`