Библиотека ExcelReportGenerator позволяет выводить данные в Excel, размечая листы в нём определённым образом . В основе библиотеки лежат шаблоны и панели. Есть несколько типов шаблонов, которые позволяют выводить свойства, элементы данных, вызывать методы и т.п.

**Шаблоны**

Шаблон представляет из себя выражение, заключённое в некоторые границы. По умолчанию границами шаблона выступают фигурные скобки (возможна настройка), например, {p:Name} – вывод свойства или {m:GetData()} – вызов метода и т.д. Шаблон может занимать как ячейку целиком:



так и появляться в ячейке внутри текста:



Если шаблон находится внутри текста, то значение, которое он возвращает, всегда приводится к строке, то есть на нём вызывается метод ToString(). Если же шаблон занимает всю ячейку, то метод ToString() не вызывается, а значит в ячейку попадает значение того типа, которое вернул шаблон. Это особенно актуально для числовых типов, а также DateTime, Boolean и некоторых других.

Рассмотрим подробнее виды шаблонов.

1. **Шаблон свойства**. Шаблон свойства в простейшем варианте имеет следующий вид:

{p:PropName}

, где p (property) – обозначает, что нужно вывести свойство, PropName – имя свойства (с учётом регистра). Свойство может быть как статическим, так и экземплярным. Обязательное условие – оно должно иметь модификатор доступа public. Следует учесть, что вместо имени свойства может быть также указано имя публичного поля.

Этот шаблон может иметь более сложный вид. В частности можно склеивать обращения к свойствам через «.», то есть шаблон {p:Prop1.Prop2…..PropN} будет означать, что нужно получить значение свойства Prop1, далее вызвать на полученном значении свойство (или поле) Prop2 и т.д. При этом, если какое-то из свойств в цепочке окажется равным null, то будет выброшено исключение InvalidOperationException. Свойств или полей в цепочке может быть сколько угодно.

По умолчанию свойство или поле ищется в типе отчёта, экземпляр которого передаётся в конструктор класса DefaultReportGenerator. Но это поведение можно переопределить, указав перед именем свойства тип, например:

{p:Company:Name}

В данном случай поле «Name» будет искаться в типе «Company» (тип отделяется от имени свойства символом «:»). При этом сам тип по умолчанию ищется в сборке исполняемого файла. Если нужный вам тип находится в другой сборке, то необходимо настроить соответствующим образом класс DefaultReportGenerator. В данном случае тип «Company» будет искаться во всех сборках, указанных в настройках класса DefaultReportGenerator. Если будет найден только один тип с таким именем, то в нём будет произведён поиск свойства «Name». Если свойство будет найдено, то шаблон вернёт его значение и поместит в соответствующую ячейку. Если же свойство в классе отсутствует, либо не соответствует модификаторам доступа (public, instance, static), то будет выброшено исключение MemberNotFoundException. Если свойство не является статическим, то будет предпринята попытка создания экземпляра данного типа. По умолчанию в таком случае тип должен иметь конструктор по умолчанию. Это поведение можно переопределить в классе DefaultReportGenerator, предоставив свою реализацию интерфейса IInstanceProvider, которая, например, будет получать экземпляры объектов с помощью контейнера IoC. Стоит заметить, что созданный экземпляр будет являться Singleton-объектом, то есть при множественных обращениях к экземплярным членам данного типа, они будут вызываться на объекте, созданном при первом обращении.

Если найдется более одного типа с данным именем, то будет выброшено исключение InvalidTemplateException. В данном случае можно скорректировать шаблон, указав в нём перед типом явно пространство имён, например:

{p:Reports.Common:Company:Name}

где Reports.Common – пространство имён, которое отделяется от имени самого типа с помощью «:». Отдельные пространства имён разделяются с помощью «.».

Также может быть ситуация, когда тип не будет найден. В этом случае будет выброшено исключение TypeNotFoundException.

Данный шаблон можно выводить как во всех типах панелей (кроме динамической), так и просто на листе. Также данный шаблон можно передавать в качестве параметров методов.

1. **Шаблон вызова метода.** Шаблон вызова метода в простейшем случае имеет вид:

{m:GetData()}

где m (method) - обозначает вызов метода, GetData() – непосредственно вызов метода (с учётом регистра). Метод может быть как статическим, так и экземплярным. Обязательное условие – он должен иметь модификатор доступа public. Методу можно передавать сколько угодно параметров, причём в качестве параметров могут выступать как статические данные (строки, числа и т.п., зашитые внутри шаблона), так и практически любые типы шаблонов. Ниже представлен вполне корректный вызов метода:

{m:GetData(p:Code, EN, m:Namespace:GetCompany(m:GetName(sv:Lang)), 56)}

Отсюда видно следующее:

1. Границы шаблона обрамляют только вызов самого внешнего метода, шаблоны внутри метода уже пишутся без границ.
2. Метод в качестве параметров может принимать как статические данные, так и любые другие шаблоны, в том числе вызовы других методов (вложенность вызова методов не ограничена)
3. Как и в случае с шаблонами свойств, перед именем метода можно указать имя типа, включая пространство имён (поиск типа и создание экземпляра в данном случае такое же, как и для шаблонов свойств)
4. Статические данные, даже если это строки, необязательно заключать в кавычки, при вызове метода будет предпринята попытка преобразовать переданные параметры к типам соответствующих параметров метода, если преобразование провалится, то будет выброшено исключение.

Поиск метода осуществляется по имени, а также по количеству и типам параметров. Методы с переменным числом параметров (params) не поддерживаются. Могут возникнуть ситуации, особенно при перегрузке методов, когда необходимо будет указать в шаблоне тип статического параметра явно, это возможно сделать следующим способом:

{m:GetData([string]10, [decimal]56.6)}

то есть в данном случае перед параметром явно указывается его тип. Это актуально для всех примитивов, а также для типов string, char, decimal и DateTime. Следует отметить, что для строк вместо указания типа, можно просто обрамить параметр в кавычки, то есть написать «10» вместо [string]10. Также ещё может возникнуть ситуация, когда внутри статического параметра присутствует запятая. По умолчанию запятые расцениваются как разделители параметров. Чтобы пометить запятую как простой символ, следует экранировать её ещё одной запятой, то есть написать «,,» например:

{m:GetData(p:Code, Hello,, World!)}

Данный шаблон можно выводить как во всех типах панелей (кроме динамической), так и просто на листе. Также данный шаблон можно передавать в качестве параметров методов.

1. **Шаблон элемента данных.** Шаблон элемента данных в простейшем случае имеет вид:

{di:Amount}

где di (DataItem) - обозначает обращение к элементу данных, Amount – в зависимости от того, чем является элемент данных, может быть именем публичного поля или свойства (элемент данных –экземпляр класса), именем колонки (элемент данных DataRow), ключ словаря (элемент данных IDictionary). В случае, если элементом данных является экземпляр класса, то возможно склеивание свойств или полей через «.».Так как панели могут вкладываться друг в друга, то в данном шаблоне можно получить доступ к текущему элементу данных родительской панели следующим образом:

{di:parent:Code}

что будет означать, что нужно получить поле «Code» из элемента данных родительской панели. Также, если элемент данных имеет простейший тип, например «int» или «string», и вы хотите вывести сам этот элемент, то в таком случае обращение должно быть следующим:

{di:di}

Данный шаблон может появляться только внутри панелей, которые имеют свойство DataSource (кроме динамической). В панели с итогами – либо в составе агрегатной функции, либо при обращении к элементу родительской панели. Также данный шаблон можно передавать в качестве параметров методов.

1. **Шаблон агрегатной функции.** Шаблон агрегатной функции в простейшем случае имеет вид:

{Sum(di:Amount)}

где Sum – имя агрегатной функции, di:Amount – элемент данных, по которому следует произвести агрегацию. Данный вид шаблона может появляться только в панелях итогов. Если элемент данных представляет из себя экземпляр класса, то обращение к свойствам можно склеивать через «.». Поддерживаются следующие встроенные агрегатные функции: Sum, Count, Avg, Min, Max. Также есть возможность создать свою агрегатную функцию, в таком случае в шаблоне нужно указать функцию «Custom» и вторым параметром наименование этой функции. При этом сама функция должна размещаться в экземпляре класса отчёта, который передаётся в конструкторе DefaultReportGenerator. То есть шаблон в данном случае будет иметь вид:

{Custom(di:Amount, CustomAggregationFunc)}

При этом CustomAggregationFunc должна иметь следующую сигнатуру:

public TResult CustomAggregationFunc(TAccumulation accumulation, TItem currentItem, int itemNumber);

где TResult- тип возвращаемого результата, TAccumulation – тип накапливаемого результата, TItem - тип каждого элемента данных, itemNumber – номер текущего элемента данных. То есть на каждой итерации в функцию передаётся накопленный результат, текущий элемент данных и номер текущего элемента данных (первый элемент будет иметь значение «1»). Например, для произведения это может выглядеть так:

public decimal CustomAggregation(decimal? accumulation, decimal currentItem, int itemNumber)

{

return (accumulation ?? 1) \* currentItem;

}

Также есть возможность указать третьим параметром функцию, которая отработает после агрегации:

{Custom(di:Amount, CustomAggregationFunc, PostAggregation)}

где PostAggregation – имя функции, которая вызовется после агрегации. Данная функция должна размещаться в экземпляре класса отчёта, который передаётся в конструкторе DefaultReportGenerator Сигнатура функции должна выглядеть следующим образом:

public TResult PostAggregation(TAggregationResult aggregationResult, int itemsCount);

где TResult- тип возвращаемого результата, TAggregationResult – тип результата агрегации, aggregationResult- результат агрегации, itemsCount – количество элементов данных. Например:

public string PostAggregation(decimal aggregationResult, int itemsCount)

{

return (aggregationResult / itemsCount).ToString("P");

}

Если необходимо применить функцию постагрегации для встроенной функции, то можно просто пропустить второй параметр, в таком случае шаблон будет выглядеть следующим образом:

{Custom(di:Amount, , PostAggregation)}

Данный шаблон может появляться только внутри панелей с итогами. Также данный шаблон можно передавать в качестве параметров методов.

1. **Шаблон системной переменной.** Шаблон вывода системной переменной имеет вид:

{sv:SheetName}

где sv (SystemVariable) - обозначает обращение к системной переменной, SheetName – имя переменной (с учётом регистра). В настоящий момент доступны следующие предопределённые системные переменные:

RenderDate – дата рендеринга

SheetName – имя листа

SheetNumber – номер листа.

Системные переменные можно расширять. Данный функционал будет описан в отдельной главе.

Данный шаблон можно выводить как во всех типах панелей (кроме динамической), так и просто на листе. Также данный шаблон можно передавать в шаблон вызова метода в качестве параметра.

1. **Шаблон вызова системной функции.** Шаблон вызова системной функции имеет вид:

{sf:Format(p:value, “C”)}

где sf (SystemFunction) - обозначает вызов системной функции, Format(p:value, “C”) – непосредственно вызов самой функции. В настоящий момент доступны следующие предопределённые системные функции:

object GetDictVal(object dictionary, object key)

Данная функция позволяет получить значение из словаря по ключу. Параметр dictionary должен реализовывать интерфейс IDictionary. Если какой-либо из параметров окажется равным null, или параметр dictionary не реализует интерфейс IDictionary или данный ключ отсутсвует в словаре, то будет выброшено соответствующее исключение

object TryGetDictVal(object dictionary, object key)

Данная функция полностью аналогична вышеописанной за исключением того, что она никогда не выбрасывает исключение. В случае любой ошибки функция возвратит null.

object GetByIndex(object list, int index)

Позволяет получить элемент списка по индексу. Параметр list должен реализовывать IList. Если параметр list равен null или он не реализует IList или выходит за границы диапазона, то будет выброшено соответствующее исключени.

object TryGetByIndex(object list, int index)

Данная функция полностью аналогична вышеописанной за исключением того, что она никогда не выбрасывает исключение. В случае любой ошибки функция возвратит null.

string Format(object input, string format, IFormatProvider formatProvider = null)

Функция позволяет преобразовать входное значение в строку с заданным форматированием. Параметр input должен реализовывать интерфейс IFormattable, в противном случае будет выброшено исключение. Также существует третий параметр для возможности задания культуры.

Системные функции можно расширять. Данный функционал будет описан в отдельной главе.

Данный шаблон можно выводить как во всех типах панелей (кроме динамической), так и просто на листе. Также данный шаблон можно передавать в шаблон вызова метода в качестве параметра.