ООО «КВАНТУМ АРТ»

115184, Москва, Озерковский переулок, д. 12

тел. (495) 783-65-74

Программные продукты «QP8.CMS» и «QP8.CMS с поддержкой PostgreSQL»

Руководство разработчика

Версия 0.12.0
02.12.2020

Назначение документа

Настоящий документ содержит руководство разработчика по программным продуктам «QP8.CMS» и «QP8.CMS с поддержкой PostgreSQL» . Также в документе содержится описание дополнительного продукта «QP8.Widgets». Цель документа – предоставить разработчику сведения, достаточные для разработки и сопровождения Информационных Систем, основанных на продуктах.

Целевая аудитория

Документ предназначен для разработчиков, обладающих следующими компетенциями:

- знание технологий программной платформы .NET для разработки веб-приложений (ASP.NET MVC, ASP.NET Core);
- знание языков C#, SQL, XAML;
- знание технологий LINQ to SQL, Entity Framework, EF.Core;
- знание веб-технологий (HTTP, DNS).

История изменений

Версия	Дата	Автор	Описание
0.12.0	02.12.2020	Селю П.Н.	Разделение на «QP8.CMS» и «QP8.CMS с поддержкой PostgreSQL»
0.11.0	01.12.2020	Филинова А.С.	Обновлено: • Раздел 5.8 JS-интеграция для веб-приложений пользовательских действий
0.10.0	10.07.2020	Духанина К.В.	Добавлено: • Раздел 5.3 Использование Виджетной платформы ASP.NET Core
0.9.9	06.07.2020	Мальцева Ю.А.	Добавлено: • Требование изменить пароль при следующем входе и требования к паролю (см. Параметры входа (Login Parameters)) • Описание настройки «Максимальное количество элементов» (The maximum number of items) в поле типа «Связь», позволяющая управлять типом контрола (см. Связь (Relation)) • Описание команды «Создать по образцу» (Create Like) для пользовательских действий (см. Пользовательское действие (Custom action))

			• Описание параметра свойства сайта «Внешняя разработка» (External Development) (см. Свойства сайта) • Подраздел «Интерфейс для просмотра существующих обратных ссылок в свойстве поля M2M» Обновлено: • Раздел «Механизм автозамены URL» • Требования к операционной системе (см. Операционная система). Обновленное требование - серверная ОС Microsoft Windows Server 2012 или выше • Требования к СУБД (см. СУБД). Использование Microsoft SQL Server 2012 или выше • Требования к веб-серверу (см. Веб-сервер) Использование Microsoft IIS 8.5 или выше • Требования к используемому веб-браузеру Internet Explorer не ниже 11.0 (см. Программное обеспечение для работы с ГПИ продукта) • Требования к используемому ПО для разработчика (см. ПО для Разработчика). Использование Microsoft Visual Studio 2017 или выше • Требования к используемому ПО для разработчика (см. ПО для Разработчика). Использование .NET Framework не ниже 4.7.1
0.9.8	15.03.2018	Советкали Б.С.	Обновлено: • описание метода `MassUpdate`. Добавлено уточнение о том, что обновляемые поля передаются в виде ключа коллекции. Также добавлено описание исполнения метода при расщеплении статьи (см. Метод MassUpdate); • описание метода `ImportToContent`. Добавлено уточнение параметра `attrIds` и исполнения метода при передачи статьей с разными коллекциями полей (см. Метод ImportToContent)
0.9.7	02.01.2018	Советкали Б.С.	• Добавлено уточнение свойства поля «Уникальное (Unique)» (см. Свойства поля) • Удалено упоминание о том, что свойство «Обязательное (Required)» недоступно полям типа M2M (см. Свойства поля) • Добавлено примечание о использовании команды контекстного меню поля «Apply Default Value» (см. Параметры, зависящие от типа (Type-specific parameters))

0.9.6	31.01.2018	Советкали Б.С.	Добавлено уточнение свойства «Общее владение (Shared Ownership)» (см. Свойства группы пользователей)
0.9.5	25.1.2018	Советкали Б.С.	**Добавлено:** • Описание параметра `Disable list auto wrapping (ul, ol, dl)` (см. Параметры визуального редактора (Visual Editor Parameters)) **Удалено:** • United режим работы сайта
0.9.4	25.1.2018	Советкали Б.С.	Добавлена ссылка на описание свойства поля (см. Фильтрация по умолчанию (Default Filters))
0.9.3	19.01.2018	Советкали Б.С.	**Добавлено:** • Пример использования функционала «фильтрация по умолчанию» (см. Фильтрация по умолчанию (Default Filters)); • Описание команд контекстного меню. Например, создание статьи (см. Статья (Article)). **Обновлено:** • Структура QP (Логическая структура QP). **Удалено:** • Информация о том, что учетная запись создаваемого пользователя имеет статус «Заблокирован» (Свойства пользователя)
0.9.2	17.01.2018	Советкали Б.С.	Обновлено: • Описание подразделов: Основные принципы, Описание синтаксиса, Remote-валидация, Поддержка ссылок {x:Reference}, Ресурсный словарь, Добавлено: • Подраздел Условие «AreEqual»; • Пример использования динамического ресурсного словаря (см. Динамический ресурсный словарь). Удалено: • Подраздел «Unit-тестирование»
0.9.1	17.01.2018	Советкали Б.С.	Обновлено: • Ссылки на описания NuGet-пакетов (см. «Установка NuGet-пакетов»)
0.9	16.01.2018	Советкали Б.С.	Обновлено: • Подраздел «Установка NuGet-пакетов»
0.8	18.12.2017	Советкали Б.С.	Добавлено: • Установка пакетов NuGet для QP

Содержание

1. Термины и определения ... 10
1.1. Общие термины и определения ... 10
1.2. Термины и определения для QP ... 10
1.3. Определения для ролей пользователей в Системе ... 11
1.4. Аббревиатуры ... 11
2. Обозначения ... 12
3. Общие сведения о продукте ... 13
3.1. Системные требования ... 13
3.1.1. Аппаратное обеспечение ... 13
3.1.2. Программное обеспечение ... 13
3.2. Документация по продукту ... 14
4. Справочник Разработчика ... 15
4.1. Логическая структура QP ... 15
4.2. Пользователь (User) ... 15
4.2.1. Свойства пользователя ... 16
4.3. Группа пользователей (User Group) ... 18
4.3.1. Свойства группы пользователей ... 19
4.4. Сайт (Site) ... 19
4.4.1. Создание нового сайта ... 20
4.4.2. Копирование сайта ... 25
4.4.3. Изменение свойств сайта ... 25
4.4.4. Удаление сайта ... 25
4.4.5. Сборка сайта ... 25
4.5. Контент (Content) ... 27
4.5.1. Создание нового контента ... 27
4.5.2. Копирование контента ... 29
4.5.3. Изменение свойств контента ... 29
4.5.4. Очистка контента ... 29
4.5.5. Удаление контента ... 30
4.6. Виртуальный контент (Virtual Content) ... 30
4.6.1. Виртуальный контент типа «JOIN» ... 30
4.6.2. Виртуальный контент типа «UNION» ... 30
4.6.3. Виртуальный контент типа «USER QUERY» ... 30
4.6.4. Создание виртуального контента ... 31
4.6.5. Изменение свойств виртуального контента ... 31

4.6.6. Удаление виртуального контента ... 31
4.7. Поле (Field) ... 31
4.7.1. Создание нового поля ... 31
4.7.2. Копирование поля ... 43
4.7.3. Изменение свойств поля ... 43
4.7.4. Удаление поля ... 43
4.8. Статья (Article) ... 43
4.8.1. Служебные свойства статьи ... 44
4.9. Архивная статья (Archive Article) ... 44
4.10. Уведомление (Notification) ... 45
4.10.1. Архитектура уведомлений ... 45
4.10.2. Обобщённый алгоритм работы уведомлений ... 45
4.10.3. Свойства уведомления ... 45
4.10.4. Сборка уведомления ... 48
4.10.5. Внешнее уведомление ... 48
4.11. Workflow ... 49
4.11.1. Статус ... 50
4.11.2. Создание и изменение Workflow ... 50
4.11.3. Использование Workflow ... 51
4.11.4. Параллельный Workflow ... 51
4.11.5. Отмена расщепления ... 52
4.11.1. Комментарий при смене статуса Workflow ... 52
4.11.2. История изменений статуса статьи ... 53
4.11.3. Удаление Workflow ... 53
4.12. Пользовательское действие (Custom action) ... 53
4.12.1. Свойства пользовательского действия ... 53
4.12.2. Передача контекста ... 56
4.12.3. Неинтерфейсное пользовательское действие ... 57
4.12.4. Подтверждение на выполнение пользовательского действия ... 57
4.13. Стиль визуального редактора (Visual Editor Style) ... 57
4.13.1. Свойства стиля визуального редактора ... 58
4.14. Плагин визуального редактора (Visual Editor Plugin) ... 58
4.14.1. Свойства плагина визуального редактора ... 59
4.15. Пользовательская валидация ... 59
4.15.1. Основные принципы ... 59

4.15.2. Описание синтаксиса ... 59
4.15.1. Remote-валидация ... 65
4.15.2. Поддержка ссылок {x:Reference} ... 66
4.15.3. Ресурсный словарь ... 67
4.15.4. Использование в QP ... 70
4.15.5. Порядок разработки валидатора ... 70
4.16. Шаблон, страница, объект, формат ... 71
4.16.1. Шаблон ... 72
4.16.2. Страница ... 75
4.16.3. Объект ... 76
4.16.4. Формат ... 84
4.16.5. Поиск по коду для объектов ... 84
4.17. Прочее ... 85
4.17.1. Библиотека сайта ... 85
4.17.2. Библиотека контента ... 85
4.17.3. Кэширование ... 85
4.17.4. Блокировка сущностей ... 86
4.17.5. Механизм автозамены URL ... 86
4.17.6. Запись и воспроизведение действий ... 87
5. Создание Систем с использованием продукта ... 92
5.1. Установка NuGet-пакетов ... 92
5.2. Использование Виджетной платформы для ASP.NET MVC ... 93
5.2.1. Архитектура платформы ... 93
5.2.2. Создание экземпляра страницы ... 94
5.2.3. Процедура создания и добавления нового типа страницы в Систему ... 95
5.2.4. Создание экземпляра виджета ... 95
5.2.5. Процедура создания и добавления нового типа виджета в Систему ... 97
5.3. Использование Виджетной платформы для ASP.NET Core ... 99
5.3.1. Список библиотек и общее описание ... 99
5.3.2. Подключение структуры сайта из QP ... 100
5.3.3. Использование структуры сайта из QP как сервис ... 108
5.3.4. Подключение таргетирования к сайту ... 109
5.3.5. Подключение движка AB-тестирования к сайту ... 114
5.3.6. Подключение функциональности кэштегов ... 118
5.3.7. Подключение и использование режима onscreen ... 122
5.3.8. Подключение структуры сайта из XML ... 125

5.4. QP8 API ... 127
5.4.1. Класс DBConnector ... 127
5.4.2. Класс QScreen ... 143
5.4.3. Класс Permissions ... 144
5.4.4. Класс ContentItem ... 151
5.4.5. Классы из пространства имён «Quantumart.QP8.BLL.Services.API» ... 153
5.4.6. Классы QPage и QUserControl ... 157
5.4.7. Класс QPublishControl ... 167
5.5. Классы LINQ to SQL ... 169
5.5.1. Сборка контентов в классы LINQ to SQL ... 169
5.5.2. Использование контекстного класса ... 172
5.5.3. Использование поля «Связь» типа M2M в классах LINQ to SQL ... 175
5.5.4. Поддержка стандартного поведения объекта «Publishing Container» ... 175
5.5.5. Поддержка служебных полей контента ... 176
5.5.6. Дополнительные свойства полей «Изображение» и «Динамическое изображение» 177
5.5.7. Классы LINQ to SQL и кэширование ... 177
5.5.8. Использование классов LINQ to SQL при создании компонентов ... 177
5.5.9. Основные ошибки использования классов LINQ to SQL ... 178
5.5.10. Пример использования ... 178
5.5.11. Настройка LINQ to SQL ... 178
5.6. Использование Entity Framework ... 179
5.6.1. Особенности генерации EF ... 179
5.6.2. Принципы работы ... 180
5.6.3. Выборочная загрузка полей ... 181
5.6.4. Подходы к локализации ... 181
5.6.5. Обратные свойства для поля «Связь» типа M2M ... 182
5.6.6. Порядок использования ... 184
5.7. Использование Entity Framework Core (отличия от Entity Framework) ... 186
5.7.1. Особенности генерации EF Core ... 186
5.7.2. Принципы работы ... 186
5.8. JS-интеграция для веб-приложений пользовательских действий ... 188
5.8.1. Метод checkHost ... 188
5.8.2. Метод executeBackendAction ... 188
5.8.3. Метод closeBackendHost ... 190
5.8.4. Метод openSelectWindow ... 191

5.8.5. Метод BackendEventObserver 191
5.8.6. Метод openFileLibrary 192
5.8.7. Метод sendNotification 193
5.8.8. Метод previewImage 193
5.8.9. Метод downloadFile 194
5.9. Кастомизация форм QP 194
5.9.1. Базовые возможности 194
5.9.2. Translit.js 197
5.9.3. Dynamic.js 197
5.10. Использование объектов QP 200
6. Пример создания Системы 201
6.1. Постановка задачи. Определение требований 201
6.2. Проектирование 201
6.2.1. Определение групп пользователей 201
6.2.2. Проектирование структуры данных 201
6.2.3. Проектирование ГПИ 201
6.3. Реализация 201
6.3.1. Подготовка структуры данных 201
6.3.2. Разработка кода веб-приложения 202
6.3.3. Создание ГПИ 203
6.3.4. Размещение виджета 205

1. Термины и определения

1.1. Общие термины и определения

Информационная Система (далее «Система») — автоматизированный программно-аппаратный комплекс, предназначенный для хранения, обработки и выдачи данных.

QP8.CMS (далее «QP») – программный продукт, обладающий широким спектром возможностей для разработки программной части Систем различной сложности.

Модульное приложение (также «Приложение», «Виджет») – обладающий ГПИ инструмент, содержащий набор функциональных возможностей для взаимодействия пользователей с какой-либо Системой (текущей или сторонней).

QP8.Widgets (также «Виджетная платформа») – продукт, расширяющий возможности QP. Позволяет через бекэнд наполнять веб-страницы Системы самостоятельно разработанными Модульными приложениями. Виджетная платформа и виджеты основаны на шаблоне архитектуры MVC (от англ. «Model-View-Controller», «Модель-Представление-Контроллер»).

Development-окружение – среда, в которой осуществляется разработка и отладка Систем.

Stage-окружение – среда, максимально приближенная к production-окружению, в которой персоналом организации-разработчика осуществляется тестирование Систем.

Production-окружение – среда, используемая для размещения Систем, готовых к эксплуатации неограниченным кругом пользователей.

Графический пользовательский интерфейс (далее «ГПИ») – метод взаимодействия пользователя с Системой, при котором все ключевые способы управления Системой выполнены с использованием различных графических элементов.

Инструмент – часть Системы, обладающая определёнными функциональными возможностями.

Обработчик – программное средство, используемое на серверной части Системы для обработки запросов пользователей к веб-сайту Системы.

Active Directory (далее «AD») – служба каталогов для операционных систем Microsoft Windows Server. Базируется на протоколе LDAP.

Entity Framework (далее «EF») – технология для доступа к данным с использованием объектно-реляционного сопоставления (ORM, от англ. «Object-Relational Mapping»).

NuGet – средство для управления пакетами, используемое при разработке программных продуктов на платформе Microsoft.

1.2. Термины и определения для QP

DNS – доменное имя, используемое в Системе для работы с веб-сайтом.

Бекэнд – копия QP. Бекэнд обладает ГПИ для работы с содержимым БД Системы.

Виртуальный путь – URI до объекта.

Код клиента (Customer code) – уникальный параметр, определяющий БД Системы, с которой взаимодействует бекэнд QP.

Контент – раздел сайта.

Поле – атрибут контента. С использованием полей формируется структура данных для контента.

Пользовательское действие – дополнительная функциональная возможность для бекэнда, добавленная Разработчиком в Систему.

Реплейс – уникальное кодовое имя для статьи, с использованием которого можно вызвать содержимое этой статьи в других статьях текущего сайта.

Сайт – набор данных в бекэнде. Допускается создание нескольких сайтов. Содержимое каждого сайта определяется созданными в нём контентами.

Статья – элемент контента. Статья содержит данные, заданные в поля контента.

Физический путь – путь до объекта в файловой системе.

1.3. Определения для ролей пользователей в Системе

Пользователь – персона, осуществляющая взаимодействие с Системой.

Администратор – пользователь с правами на внесение любых изменений в Систему, которые можно выполнить с использованием бекэнда QP.

Контент-менеджер – пользователь с ограниченными правами на изменение содержимого Системы с использованием бекэнда QP.

Разработчик – пользователь с правами на внесение любых изменений в Систему (в том числе в содержимое скриптов, структуру БД).

1.4. Аббревиатуры

БД – База данных.

СУБД – система управления базами данных.

HTML (от англ. «HyperText Markup Language») – язык разметки документов.

JSON (от англ. «JavaScript Object Notation») – текстовый формат обмена данными.

XML (от англ. «eXtensible Markup Language») – расширяемый язык разметки документов.

API (от англ. «Application programming interface», интерфейс программирования приложений) – набор правил по использованию функциональных возможностей Системы, предоставляемый разработчикам для организации взаимодействия сторонних программных продуктов с Системой.

LINQ (от англ. «Language-Integrated Query») – компонент .NET Framework для работы с данными из БД, как с объектами. Запросы к СУБД формируются с использованием языков программирования .NET.

LINQ to SQL – решение для LINQ, позволяющее в качестве источника данных использовать СУБД Microsoft SQL Server.

DPC (Digital Product Catalog) – программный продукт, разработанный на базе QP. Ориентирован на удобство работы со структурой данных для различных продуктов Заказчика.

POCO (от англ. «Plain old CLR object») – подход к разработке программного обеспечения, предполагающий использование максимально простых классов для работы с объектами.

2. Обозначения

Обозначение	Описание	Пример использования
`Технические данные`	Используется для выделения различных технических данных в тексте: URL, названия свойств и методов, имена файлов и т.п.	ГПИ Системы доступен по URL `https://www.domainname.zone/`.
Код	Пример кода.	`public DataTable Data { get; set; }`
Переменная	Используется для указания переменного значения.	Формат URL: *Базовый URI/Псевдоним объекта*
Примечание:	Дополнительные данные справочного характера.	**Примечание:** используется при генерации классов LINQ to SQL.
Внимание:	Важные данные, которые требуется обязательно учитывать.	**Внимание:** опция поддерживается только ASP-сборкой в целях совместимости.

3. Общие сведения о продукте

3.1. Системные требования

3.1.1. Аппаратное обеспечение

	Минимальная конфигурация	**Рекомендуемая конфигурация**
Процессор	Intel Pentium IV 1.8 ГГц	Intel Xeon 2.4 ГГц x2
Память	2 ГБ	8 ГБ
Дисковое пространство	2 ГБ	100 ГБ и больше (в зависимости от применения)

3.1.2. Программное обеспечение

Операционная система

Окружение	**Описание**
Production	Требуется серверная ОС: • Microsoft Windows Server 2012 или выше
Stage	Достаточно ОС для настольных ПК: • Microsoft Windows 8.1, • Microsoft Windows 10. Рекомендуется ОС из списка для production-окружения.
Development	

Поддерживаются 64-битные версии ОС.

Для рабочего места Разработчика достаточно любой из перечисленных ОС для настольных ПК.

СУБД

Полностью поддерживаемые СУБД:

- Microsoft SQL Server 2012 или выше

Примечание: для выполнения обновления продукта с использованием мастера установки требуется модуль sqlps (SQL Server PowerShell).

Частично поддерживаемые СУБД:

Внимание: в Microsoft SQL Server Express без компонента «Advanced Services» не поддерживается полнотекстовый поиск.

- Microsoft SQL Server 2012 Express (с «Advanced Services») или выше.

Веб-сервер

- Microsoft IIS 8.5 или выше

Программное обеспечение для работы с ГПИ продукта

Работа с ГПИ ведётся с использованием веб-браузера. Поддерживаемые веб-браузеры:

- Google Chrome (или веб-браузер на основе Chromium),
- Microsoft Internet Explorer (не ниже 11.0),
- Microsoft Edge,
- Mozilla Firefox.

Примечание: рекомендуется использовать актуальную версию веб-браузера.

ПО для Разработчика

- .NET Framework версии не ниже 4.7.1.
- Microsoft Visual Studio 2017 или выше.

3.2. Документация по продукту

Пакет документов по продукту содержит:

Название	Описание
Руководство администратора	Установка, обновление, удаление продукта и эксплуатация Системы на основе продукта
Руководство разработчика	Разработка Системы на основе продукта
Руководство редактора	Работа с данными в Системе, созданной на основе продукта, в роли Контент-менеджера

4. Справочник Разработчика

4.1. Логическая структура QP

Внимание: объекты ветви «Шаблон» являются устаревшими (не рекомендуется использовать для разработки новых Систем; используются для сопровождения Систем, основанных на предыдущих версиях продукта).

Структура QP представляет собой иерархию сущностей:

- БД
  - Пользователь
  - Группа пользователей
  - Сайт
    - Контент
      - Поле
      - Статья
      - Архивная статья
      - Уведомление
    - Виртуальный контент
    - Workflow
    - Статус
    - Шаблон
      - Объект шаблона
        - Формат
      - Страница
        - Объект страницы
          - Формат
  - Пользовательское действие
  - Стиль визуального редактора
  - Плагин визуального редактора

Для обновления структуры иерархии, если были внесены изменения, необходимо вызвать контекстное меню для сайта и нажать на пункт «Обновить».

Примечание: обновление доступно на всех уровнях иерархии. Вызов команды «Обновить» обновляет сущности выбранного уровня.

4.2. Пользователь (User)

Доступ к бекэнду предоставляется только пользователям, авторизованным в Системе. Управление пользователями осуществляется с использованием:

1) ГПИ бекэнда (раздел «Пользователи»),
2) Команд контекстного меню в структуре QP:
   a. Если контекстное меню вызывается для сущности «Пользователь», то доступна команда создания новой учетной записи пользователя - «Новый пользователь»;
   b. Если контекстное меню вызывается для конкретного пользователя сущности «Пользователь», то доступны команды:
      i. «Создать по образцу». Команда создает копию учетной записи пользователя. Копия имеет отличный от оригинала логин, к логину добавляется символ «1»;
      ii. «Удалить». Команда удаляет учетную запись после ее подтверждения;

iii. «Свойства». Команда открывает вкладку в ГПИ с настройками учетной записи пользователя.

3) QP8 API (класс `Permissions`).

4.2.1. Свойства пользователя

Параметры входа (Login Parameters)

Название	Описание
Логин (Login)	Уникальный (в пределах БД) идентификатор пользователя. Используется для доступа в бекэнд.
Пароль (Password)	Набор символов, используемый для авторизации пользователя в бекэнде.
NT-логин (NT-login)	Идентификатор пользователя. Используется режимом единой авторизации для привязки пользователя QP к пользователю Windows из AD.
Пользователь должен изменить пароль при следующей авторизации (User must change password at next logon)	Указатель, что при следующем входе, пользователь должен будет изменить пароль. Требования к паролю следующие: • Длина пароля от 7 до 20 символов Пароль должен содержать три из четырех категорий: цифры, латинские прописные буквы, латинские строчные буквы, не алфавитно-цифровой символ
Автоматический вход (Auto Login)	Указатель, что для пользователя включен режим единой авторизации с AD.
Заблокирован (Disabled)	Указатель, что пользователь деактивирован (нет прав на работу в бекэнде)

Данные профиля (Profile data)

Название	Описание
Имя (First Name)	Персональные данные пользователя. **Примечание:** используются в ГПИ бекэнда при выводе данных, связанных с пользователем (например, данные о последнем изменении сущности пользователем).
Фамилия (Last Name)	
E-mail	Адрес электронной почты пользователя.
Язык (Language)	Язык для ГПИ бекэнда. **Внимание:** ОС на веб-сервере и на компьютере пользователя должны поддерживать выбранный язык.

Членство (Membership)

Название	Описание
Группы (Groups)	Группы пользователей, в которые входит пользователь.

Фильтрация по умолчанию (Default Filters)

Функциональная возможность применяется тогда, когда нет необходимости полностью ограничивать доступ пользователю только определенными категориям статей (права доступа на связи). В тоже время у пользователя могут быть предпочтительные категории статей, с которыми он работает в контенте чаще всего. В этом случае можно настроить фильтрацию по умолчанию (см. пример).

Первичная настройка выполняется в ГПИ бэкэнда на уровне пользователя: задается сайт, контент и статьи, задающие требуемые категории.

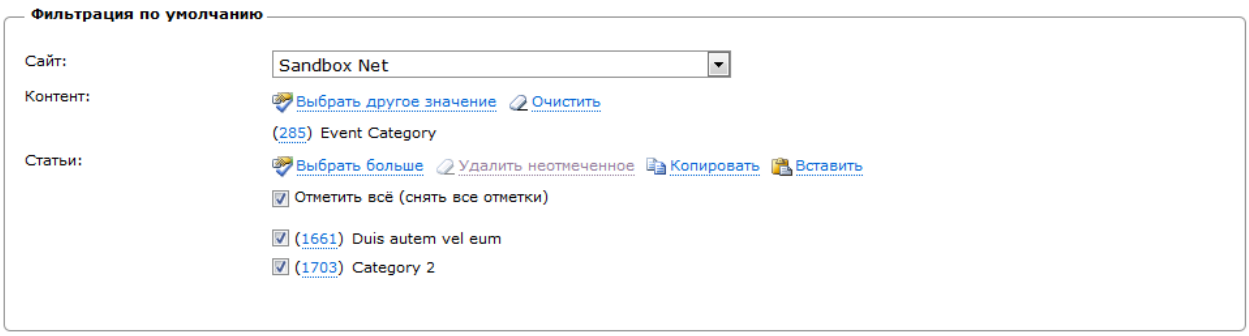

Рисунок 4.2.1-1 ГПИ для возможности «Фильтрация по умолчанию»

После этого фильтрацию нужно включить на уровне конкретного поля, так как в контенте может быть несколько подходящих полей. Это делается с помощью свойства «Использовать для фильтрации по умолчанию» (см. Связь типа O2M, Связь типа M2M).

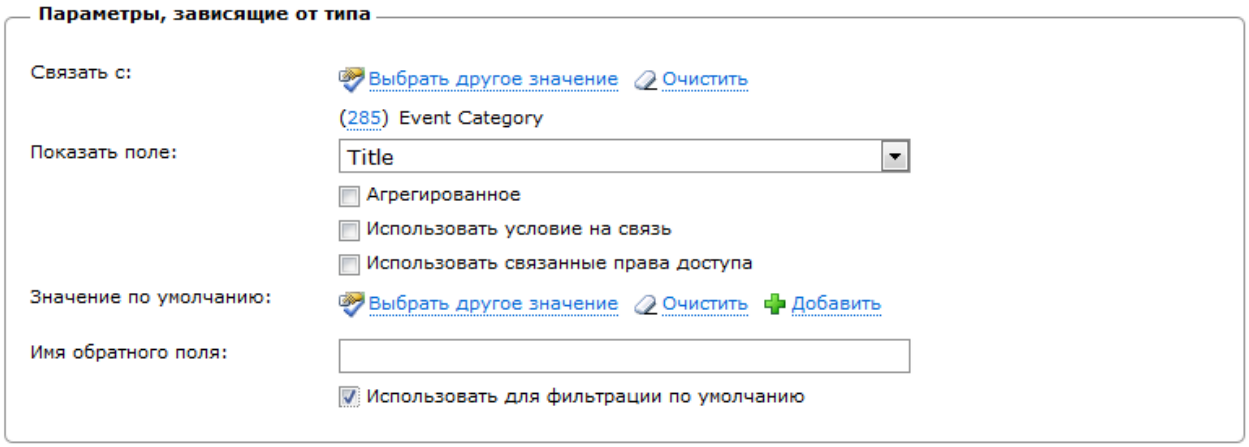

Рисунок 4.2.1-2. Свойство «Использовать для фильтрации по умолчанию» для поля

Пример настройки:

1. В настройках учетной записи пользователя «А» задаем сайт `SandBox Net`, контент `Event Category` и статьи, с которыми связано поле контента `Event`;
2. Для поля `Field` контента `Event` задаем связь с контентом `Event Category` и устанавливаем флаг `Использовать для фильтрации по умолчанию`;

Внимание!

Поле контента, которое настраивается на связь с контентом, указанном в настройках сайта, не должно ссылаться на иерархический контент или контент-расширение.

Примечание: на момент написания документа (18.01.2018) данная функциональность применялась к роли Контент-менеджер.

3. При открытии списка статей контента `Event`, для которого настроена фильтрация по умолчанию, отобразятся те статьи, что имеют связь со статьями (контент `Event Category`), указанными в настройках учетной записи пользователя «А». В настройках фильтрации, которые вызываются нажатием пиктограммы , по умолчанию будет установлена фильтрация по умолчанию. Для того чтобы сбросить заданную фильтрация, необходимо нажать на «X», установлено фильтра. Клик по кнопке «По умолчанию» вызывает фильтрацию по умолчанию.

Фильтрация по умолчанию применяется при открытии списка статей. Параметры фильтрации можно изменить, если развернуть панель поиска/фильтрации, затем удалить/изменить фильтр по умолчанию или добавить другой фильтр. Чтобы вернуть фильтр по умолчанию, следует нажать кнопку «По умолчанию».

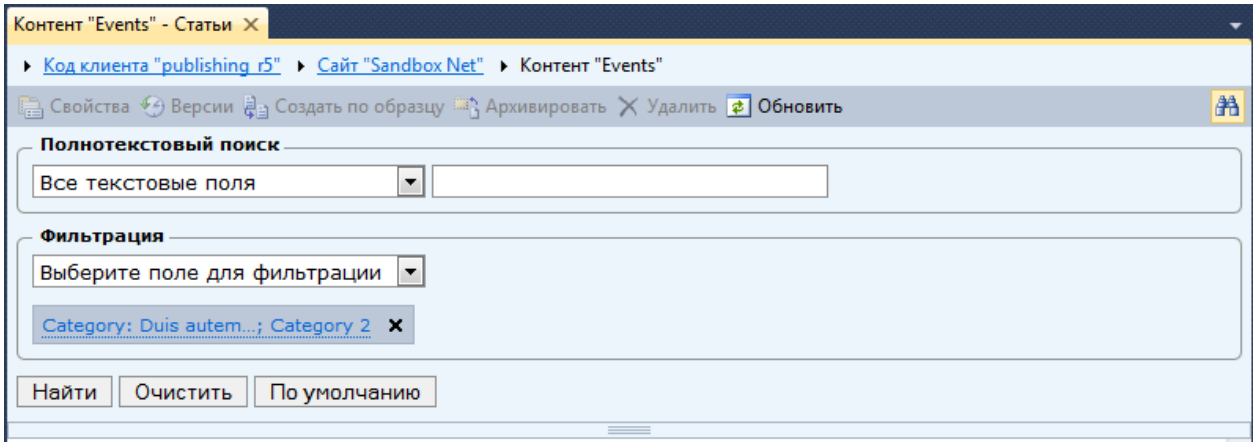

Рисунок 4.2.1-3. Результат использования фильтрации по умолчанию

Пример

Название	Описание
Сайт (Site)	Выбор сайта, в котором требуется применять правила фильтрации по умолчанию.
Контент (Content)	Выбор контента, в котором требуется применять правила фильтрации по умолчанию.
Статьи (Articles)	Выбор статей, в которых требуется применять правила фильтрации по умолчанию.

Настройки отображения (View settings)

Название	Описание
Разрешить группировку контентов в дереве бекэнда (Enable content grouping in backend tree)	Указатель, требуется ли использовать группы контентов в дереве сущностей.

Параметры OnScreen (OnScreen Parameters)

Внимание: режим OnScreen существует только в предыдущих версиях продукта.

Название	Описание
Разрешить для объектов (Allow for objects)	**Внимание:** опция поддерживается только ASP-сборкой в целях совместимости. Указатель, имеет ли пользователь право работать в режиме OnScreen с объектами QP.
Разрешить для полей (Allow for fields)	Указатель, имеет ли пользователь право работать в режиме OnScreen с содержимым полей.

4.3. Группа пользователей (User Group)

Группы пользователей позволяют присваивать нескольким пользователям единый набор полномочий на использование продукта. Если один пользователь добавлен в несколько групп, то ему предоставляются все полномочия, определённые для данных групп.

Управление группами пользователей осуществляется с использованием:

1. ГПИ бекэнда (раздел «Группа пользователей»),

2. Команд контекстного меню в структуре QP:
   a. Если контекстное меню вызывается для сущности «Группа пользователей», то доступна команда создания новой группы пользователей - «Новая группа»;
   b. Если контекстное меню вызывается для конкретного пользователя сущности «Пользователь», то доступны команды:
      i. «Создать по образцу». Команда создает копию группы пользователей. Копия имеет отличное от оригинала имя, к имени добавляется символ «1»;
      ii. «Удалить». Команда удаляет группу пользователей, после подтверждения вызова команды;
      iii. «Свойства». Команда открывает вкладку свойств группы пользователей в ГПИ.
3. QP8 API (класс `Permissions`).

4.3.1. Свойства группы пользователей

Название	Описание
Имя (Name)	Имя группы пользователей
Описание (Description)	Описание группы пользователей
Общее владение (Shared Ownership)	Указатель, требуется ли предоставлять пользователям из группы полные права доступа к статьям контента, созданным любым из входящих в группу пользователем. При этом на уровне контента должны быть включены права доступа статей (см. Основные параметры (Basic Parameters)). По умолчанию выключено (полные права доступа на сущность имеет только пользователь, её создавший).
Параллельный Workflow (Parallel Workflow)	Указатель, требуется ли пользователям группы использовать параллельный Workflow
Члены группы могут разблокировать сущности (Group members can unlock entities)	Указатель, что пользователи имеют право разблокировать сущности, заблокированные другими пользователями
Пользователи (Users)	Список пользователей, включённых в группу пользователей
Родительская группа (Parent Group)	Группа, от которой требуется наследовать права пользователей

4.4. Сайт (Site)

Сайт в QP является контейнером верхнего уровня для следующих сущностей:

1) «Контент»,
2) «Виртуальный контент»,
3) «Workflow»,
4) «Статус»,
5) «Шаблон».

БД QP может содержать один или более сайтов. На уровне сайта хранятся общие настройки, такие как физические пути в файловой системе и доменные имена (DNS). На уровне веб-сервера сайту QP может соответствовать:

- веб-сайт,

- виртуальный каталог/приложение,
- обычная директория (не рекомендуется).

В ГПИ бэкенда доступны следующие функции управления сайтами:

1) создание нового сайта,
2) копирование сайта,
3) сборка сайта,
4) изменение свойств сайта,
5) удаление сайта.

4.4.1. Создание нового сайта

Создание сайта выполняется одним из следующих способов:

1) выбрать раздел «Сайты» (Sites), нажать «Добавить новый сайт» (New Site);
2) в контекстном меню раздела «Сайты» выбрать пункт «Новый сайт» (New Site).

Свойства сайта

Основные параметры (Basic parameters)

Название	Описание
Имя (Name)	Имя сайта
Описание (Description)	Описание сайта
Внешняя разработка (External Development)	Указатель, что разработка кода для веб-сайта осуществляется во внешней относительно QP среде. При использовании внешней разработки скрываются и исключаются из валидации поля из групп полей: • Параметры сборки • Расположение страниц в Основном режиме • Расположение страниц в Тестовом режиме • Расположение файлов сборок • Параметры сборки OnScreen Также исключено создание лишних папок и копирование файлов в эти папки из скрываемых контролов с путями. Генерируемые из бэкенда данные (классы, сопоставления) помещаются не в директорию сайта, а архивируются и предлагаются пользователю на загрузку в веб-браузере сразу же после их генерации.
Заменять URLS на заполнители (Replace urls to placeholders)	Указатель, требуется ли использовать механизм автозамены URLS на заполнители (placeholders). Подробнее в разделе «4.17.5 Механизм автозамены URL»
Режим работы сайта (Site Mode)	Доступные варианты: 1) «Основной» (Live), «Тестовый» (Stage).

Режим работы сайта

Для сайта QP доступны следующие режимы работы:

1) основной (Live),
2) тестовый (Stage).

Примечание: по умолчанию используется режим «Live».

Режим задаётся в свойствах сайта.

От выбора режима зависит, какие данные сайта используются для вывода на веб-сайт.

В Live-режиме не используются следующие данные:

1) неопубликованные статьи,
2) архивные статьи,
3) новые версии расщеплённых статей.

Режим влияет на:

1) генерируемые классы LINQ to SQL,
2) используемую при сборке сайта директорию.

Параметры сборки (Assembling Parameters)

Название	Описание
Тип сборки (Assembling Type)	**Внимание:** тип «ASP» является устаревшим и не рекомендуется к использованию при разработке новых Систем. Доступные варианты: 1) ASP.NET, 2) ASP.
Включить пользовательские сессии (Enable User Sessions)	Включает поддержку пользовательских сессий для веб-сайта, что позволяет использовать настройку и персонализацию страниц для отдельных пользователей. По умолчанию включено. **Внимание:** режим OnScreen существует только в предыдущих версиях продукта. **Примечание:** в Stage-режиме пользовательские сессии включены всегда, так как это необходимо для режима OnScreen. Отключение этого режима позволяет снизить нагрузку на сервер в том случае, если персонализация не требуется.
Собирать страницы для предварительного просмотра и уведомлений в Основном режиме (Assemble Preview and Notifications In Live Mode Only)	**Внимание:** свойство устарело. Используется только в Системах, разработанных с использованием объектов QP из ветви «Шаблон». Указатель, требуется ли выполнять сборку следующих страниц всегда в Live-режиме: • страницы для форматов уведомлений, • страницы для предварительного просмотра, • страница для глобального CSS. По умолчанию выключено. Полезно в случаях, когда доступ к веб-сайту в Stage-режиме ограничен (например, по паролю или IP-адресу).
Принудительная сборка (Force assembling without taking the date of the object last update into account)	Включает режим сборки всех элементов управления вне зависимости от того, когда они были модифицированы. Так как данная опция может оказать негативное влияние на производительность, то после любой сборки сайта, шаблона или страницы происходит её автоматическое отключение.

Примечание: раздел неактуален при включенной галочке «Внешняя разработка» (External Development)

Адрес сайта (Site Location)

Название	Описание

DNS	Доменное имя для веб-сайта в режиме «Live»
Использовать отдельный DNS для Тестового режима (Use Separate DNS for Stage Mode)	Указатель, что для веб-сайта в Stage-режиме следует использовать отдельное доменное имя
DNS для тестового режима (Stage DNS)	Доменное имя для веб-сайта в режиме «Stage»

Расположение загруженных файлов (Uploaded Files Location)

Название	Описание
Виртуальный путь (Virtual Path)	URI для Библиотеки сайта
Использовать абсолютный виртуальный путь (Use absolute virtual path)	Указатель, что требуется использовать URL для Библиотеки сайта
Префикс виртуального пути (Virtual Path Prefix)	Протокол и доменное имя для обращения к Библиотеке сайта
Физический путь (Physical Path)	Путь до Библиотеки сайта в файловой системе

Расположение страниц в Основном режиме (Live Pages Location)

Название	Описание
Виртуальный путь (Virtual Path)	URI для сайта в режиме Live
Физический путь (Physical Path)	Абсолютный путь до директории сайта в файловой системе в режиме Live
Использовать тестовую папку (Force Test Directory)	Указатель, что сборка должна быть выполнена в отдельную тестовую директорию
Тестовая папка (Test Directory)	Абсолютный путь до тестовой директории сайта в файловой системе

Примечание: раздел неактуален при включенной галочке «Внешняя разработка» (External Development)

Расположение страниц в Тестовом режиме (Stage Pages Location)

Название	Описание
Виртуальный путь (Virtual Path)	URI для сайта в режиме Stage
Физический путь (Physical Path)	Путь до директории сайта в файловой системе в режиме Stage

Примечание: раздел неактуален при включенной галочке «Внешняя разработка» (External Development)

Расположение файлов сборок (Assembly Files Location)

Название	Описание
Физический путь для Основного режима (Live Physical Path)	Путь в файловой системе до директории для размещения файлов сборок сайта в режиме Live
Физический путь для Тестового режима (Stage Physical Path)	Путь в файловой системе до директории для размещения файлов сборок сайта в режиме Stage

Примечание: раздел неактуален при включенной галочке «Внешняя разработка» (External Development)

Параметры сборки OnScreen (OnScreen Assembling Parameters)

Внимание: режим OnScreen существует только в предыдущих версиях продукта.

Название	Описание
Разрешить OnScreen (Enable OnScreen)	Указатель, что разрешена работа в режиме OnScreen
Рамка поля (Field Border)	Выбор условия показа рамки для полей. Доступные варианты:

	• «Всегда», • «При наведении курсором», • «Никогда».

Примечание: раздел неактуален при включенной галочке «Внешняя разработка» (External Development)

Параметры визуального редактора (Visual Editor Parameters)

Примечание: свойства используются для поля типа «Визуальный редактор».

Название	Описание
Вставлять тег P при нажатии Enter (Insert P tag while pressing Enter)	Глобальное определение режима. • Свойство активно – при переводе строки добавляется тэг `<p>`. • Свойство неактивно – при переводе строки добавляется тэг ` `.
Использовать правила английской типографики (Use English typographic rules)	Глобальное определение, требуется ли использовать правила английской типографики. **Примечание:** по умолчанию выключено (используются правила русской типографики). **Примечание:** реализовано с использованием плагина визуального редактора.
Disable list auto wrapping (ul, ol, dl)	Отключает автоматическое закрытие парных тегов: • ul • ol • dl
Команды (Commands)	Глобальное определение доступной пользователям функциональности для редактора. **Примечание:** в список включены команды объектов «Плагин визуального редактора» (выводятся после списка базовых команд). **Примечание:** для пользователей из группы «Администраторы» свойство не используется (пользователям доступны все имеющиеся в редакторе возможности).
Класс корневого элемента (Root Element Class)	Имя CSS-класса для корневого элемента (в тэге `<body>`). **Примечание:** используется при подключении классов с использованием свойства «Внешние CSS».
Внешние CSS (External CSS)	URL для CSS-файлов, которые требуется использовать в редакторе
Стили (Styles)	Глобальное задание объектов «Стиль визуального редактора», доступных пользователю в редакторе. **Примечание:** используются объекты с неактивным свойством «Формат».
Форматы (Formats)	Глобальное задание объектов «Стиль визуального редактора», доступных пользователю в редакторе. **Примечание:** используются объекты с активным свойством «Формат».

Параметры сборки в LINQ-классы (LINQ Assembling Parameters)

Название	Описание
Импортировать файл отображения в базу данных (Import mapping file to database)	Указатель, что требуется выполнить импорт в БД существующего пользовательского файла отображения
Использовать прямое отображение из базы	Выбор способа генерации классов LINQ to SQL

данных (Use direct mapping from database)	
Выполнять генерацию, независимую от БД (Proceed DB-independent generation)	Включение опции позволяет генерировать код, переносимый между различными БД. В этом случае сопоставление между сущностными классами и таблицами БД осуществляется с помощью специального MAP-файла, который также генерируется в этом режиме. Имя MAP-файла совпадает с именем контекстного класса.
Генерировать только .map-файл (Generate only .map file)	Модификация предыдущей опции. Нужна в случае выполнения на данной БД переносимого кода, который был сгенерирован на другой БД. Таким образом, от данной БД требуется только сопоставление между сущностными классами и таблицами БД.
Скачивать только источник данных для EF (Download EF data source only)	**Внимание:** свойство доступно только при использовании свойства «Внешняя разработка». При генерации данных пользователю предлагаются для загрузки только данные для использования с Entity Framework (см. Использование Entity Framework).
Имя строки подключения (Connection string name)	Название строки соединения в разделе `connectionStrings` конфигурационного файла веб-сайта (`web.config`). **Примечание:** значение по умолчанию – `qp_database`.
Заменять URL (Replace URLs)	Указатель, требуется ли использовать механизм автозамены URL. **Примечание:** включение опции позволяет классам поддерживать стандартное поведение объекта «Publishing Container».
Использовать длинные URL (Use long URLs)	Модифицирует предыдущую опцию так, что замена будет проводиться на URL, а не на URI. **Примечание:** аналогичная опция есть для объекта «Publishing Container».
Пространство имён для генерируемых классов (Namespace for generated classes)	Пространство имён, в котором будут сгенерированы классы. **Примечание:** по умолчанию не задано.
Имя контекстного класса (Context Class Name)	Имя контекстного класса. **Примечание:** значение по умолчанию – `QPDataContext`.
Посылать уведомления (Send notifications)	Указатель, требуется ли поддержка уведомлений

Свойства уведомлений (Notification Properties)

Название	Описание
Внешний URL (External URL)	URL, к которому следует формировать запросы на создание уведомлений

Пользовательская валидация (Custom Validation)

Название	Описание
Создать словарь для валидации XAML по умолчанию (Create Default XAML Dictionary)	Указатель, что при сохранении сайта требуется автоматически сгенерировать шаблон для создания словаря для валидации и сохранить результат в качестве значения свойства «Словари для валидации XAML». **Примечание:** значение опции не хранится в БД.
Словари для валидации XAML (Dictionaries for XAML Validation)	**Внимание:** в случае использования свойства «Создать словарь для валидации XAML по умолчанию» следует обязательно наполнить полученный шаблон собственным кодом. XAML-код словаря для валидации.

Скрипты (Scripts)

Название	Описание

Скрипт для формы контента (Content form script)	Вызов скриптов, которые требуется подключить к форме создания и изменения статьи в ГПИ бекэнда. **Примечание:** используется механизмом изменения форм в ГПИ бекэнда.

4.4.2. Копирование сайта

Копирование сайта выполняется одним из следующих способов:

1) в контекстном меню сайта выбрать пункт «Создать по образцу» (Create Like),
2) в списке сайтов выбрать требуемый сайт и нажать кнопку «Создать по образцу» (Create Like).

Перед созданием копии запрашивается указание свойств нового сайта.

Если в свойствах сайта-источника установлен указатель External Development, то реализуется аналогичная логика скрытия контролов/исключения создания папок.

4.4.3. Изменение свойств сайта

Изменение свойств сайта выполняется одним из следующих способов:

1) в контекстном меню сайта выбрать пункт «Свойства» (Properties),
2) в списке сайтов выбрать требуемый сайт и нажать кнопку «Свойства» (Properties).

4.4.4. Удаление сайта

Внимание: сайт может быть удалён только при выполнении следующих условий:

1) при наличии прав пользователя на выполнение данного действия;
2) сайт не заблокирован другим пользователем;
3) сайт не содержит контентов, используемых другими сайтами.

Удаление сайта выполняется одним из следующих способов:

1) в контекстном меню сайта выбрать пункт «Удалить» (Remove),
2) в списке сайтов выбрать требуемый сайт и нажать кнопку «Удалить» (Remove).

4.4.5. Сборка сайта

Внимание: данные предназначены для использования с предыдущими версиями продукта, в которых рекомендовалось применение объектов ветви «Шаблон».

При использовании объектов ветви «Шаблон» код веб-сайта хранится в БД. Под сборкой понимается процесс генерации на веб-сервере файлов с кодом (страниц и элементов управления) из пользовательского кода и настроек, хранящихся в БД QP. Процесс сборки заключается в том, что этот код извлекается и к нему добавляется служебный код, формируемый на основании пользовательских настроек и прочих метаданных из БД. Весь полученный код записывается в файлы в соответствии с выбранным типом сборки и режимом работы сайта.

Примечание: процесс сборки может быть вызван для следующих объектов QP:

- «Сайт»,
- «Шаблон»,
- «Страница»,
- «Объект».

ASP-сборка

Осуществляется постранично. Сборка сайта и шаблона представляет собой сборку последовательности страниц. Сборка отдельных объектов не поддерживается. Для каждой страницы создается один ASP-файл, куда записываются все необходимые для ее работы данные:

1) код шаблона,
2) все вызываемые на странице объекты (во всех возможных форматах вызовов).

ASP.NET-сборка

Осуществляется как постранично, так и пообъектно. Связано с тем, что код записывается во множество файлов. По флагам в БД осуществляется проверка, нужно ли собирать элемент кода. Если значение флага равно `1`, то формат объекта (страница или формат шаблона) должен быть собран в нужном режиме.

Список флагов:

- `Live`,
- `Stage`,
- `Live Notification`,
- `Stage Notification`.

Алгоритмы сборки

Название технологии	**Название алгоритма**	**Описание**
ASP.NET	Сборка сайта	Осуществляется из свойств сайта и из списка сайтов. • Сборка уведомлений сайта, • Сборка пустых шаблонов сайта (в которых нет страниц), ○ Сборка объектов шаблонов, • Сборка страниц сайта (по алгоритму сборки нескольких страниц).
	Сборка шаблона	Осуществляется из свойств шаблона и его объектов, а также из списка шаблонов. • Сборка уведомлений шаблона; • Если шаблон пустой (нет страниц), то собираются все объекты шаблона; • Если в шаблоне есть страницы, то проводится сборка страниц шаблона (по алгоритму сборки нескольких страниц).
	Сборка нескольких страниц	Происходит при сборке шаблона или при выборе нескольких страниц для сборки из списка страниц. • Сборка объектов шаблона с учётом флагов, • Цикл по страницам, ○ Сборка объектов страницы с учётом флагов, • Обновление зависимостей кэша.
	Сборка одной страницы	Происходит при вызове сборки в свойствах страницы или объектов этой страницы.

		• Сборка объектов шаблона с учётом флагов, • Сборка объектов страницы с учётом флагов, • Обновление зависимостей кэша.
ASP	Сборка страницы	• Анализ кода шаблона, поиск вызовов объектов, • Найденные объекты записываются в код страницы, • Рекурсивный анализ кода объектов. Собранный ASP-файл содержит в себе всё необходимое для работы, включая объекты других шаблонов. В этом состоит принципиальное отличие от сборки ASP.NET.
	Сборка сайта	• Цикл по страницам сайта, • Сборка каждой страницы.

4.5. Контент (Content)

Работа с контентами осуществляется на уровне сайта. Контенты представляют собой пользовательские таблицы БД, управляемые с помощью QP. Данные из контентов можно выводить на веб-сайт с помощью:

1) объекта типа «Publishing Container»,
2) QP8 API,
3) генерируемых классов LINQ to SQL,
4) прямого доступа к БД (не рекомендуется).

Дополнительно для контента и его статей могут быть настроены:

- права доступа,
- уведомления,
- Workflow.

В ГПИ бэкенда доступны следующие функции управления контентами:

1) создание нового контента,
2) копирование контента,
3) изменение свойств контента,
4) очистка контента,
5) удаление контента.

4.5.1. Создание нового контента

Создание контента выполняется одним из следующих способов:

1) выбрать раздел «Контенты» (Contents), нажать «Добавить новый контент» (New Content);
2) в контекстном меню раздела «Контенты» выбрать пункт «Новый контент» (New Content).

Свойства контента

Основные параметры (Basic Parameters)

Название	Описание
Имя контента (Content name)	Имя контента
Описание (Description)	Описание контента
Группа (Group)	Группа контентов.

	Группа влияет на отображение контента в ГПИ в дереве сущностей. **Примечание:** по умолчанию контенты добавляются в группу «Группа по умолчанию». Заданная группа может быть изменена в дальнейшем. **Примечание:** если в группе не остается ни одного контента, то она автоматически удаляется.
Число статей на странице (Number of Articles Per Page)	Количество статей, которое требуется выводить на странице со списком статей
Включить права доступа для статей (Enable articles permissions)	Указатель, должна ли быть возможность устанавливать права доступа для статей контента
Архивировать при удалении (Archive on Removal)	Указатель, что после удаления статья должна переводиться в архивные статьи
Разрешить доступ из других сайтов (Enable Site Sharing)	**Внимание:** свойство устарело. Используется только в Системах, разработанных с использованием объектов QP из ветви «Шаблон». Указатель, что контент должен быть доступен из других сайтов бекэнда.
Запретить операции редактирования (Disable Changing Actions)	Указатель, что в ГПИ бекэнда не должно быть возможности изменять содержимое статей контента

Workflow

Название	Описание
Workflow	Выбор Workflow. Если в одном из Workflow установлен флаг По умолчанию для новых контентов, то именно он будет выбран по умолчанию в данном поле
Расщеплять статьи (Split Articles)	Указатель, требуется ли использовать режим расщепления статей

Контроль версий (Version Control)

Название	Описание
Создавать версии (Create Versions)	Указатель, требуется ли создавать версии статьи
Максимальное число хранимых версий (Maximum Number of Stored Versions)	Количество хранимых версий статьи

Параметры отображения в LINQ-классы (LINQ Mapping Parameters)

Название	Описание
Отображать как класс (Map as class)	Указатель, что для контента требуется генерация класса.
Имя (единственное) (Name (singular))	Допустимое в C# имя контента в единственном числе. Используется в качестве названия класса.
Имя (множественное) (Name (plural))	Допустимое в C# имя контента во множественном числе. Используется в качестве названия свойства контекстного класса, с которого начинается построение запроса LINQ to SQL.
Использовать фильтрацию по умолчанию (Use default filtration)	Указатель, что в классе LINQ to SQL должна быть поддержка: • расписания публикации, • Workflow, • архива статей. **Примечание:** включение опции требуется для реализации поведения, идентичного стандартному поведению объекта «Publishing Container».
Имя дополнительного контекстного класса	**Внимание:** используется только при включённой опции «Выполнять генерацию, независимую от БД» на уровне сайта.

(Additional context class name)	Кроме стандартной генерации, позволяет создать дополнительные небольшие контекстные классы для контента или группы контентов (у которых совпадают значения опции). Это может быть полезно для создания небольших переносимых частей функциональности. В качестве значения нужно указывать полное имя класса с учетом пространств имён. Пространства имён для всех генерируемых контекстных классов (как основного, задаваемого на уровне сайта, так и дополнительных) должны различаться для предотвращения конфликта имён.

Клиентские скрипты (Client scripts)

Название	Описание
Скрипт для формы (Script for form)	Вызов скриптов, которые требуется подключить к форме создания и изменения статьи в ГПИ бекэнда. **Примечание:** используется механизмом изменения форм в ГПИ бекэнда.

Пользовательская валидация (Custom Validation)

Название	Описание
Создать валидацию XAML по умолчанию (Create Default XAML Validation)	Указатель, что при сохранении контента требуется автоматически сгенерировать правила валидации по умолчанию и сохранить результат в качестве значения свойства «Валидация XAML». **Примечание:** значение опции не хранится в БД.
Валидация XAML (XAML Validation)	XAML-код с правилами валидации данных для статей контента
Запретить валидацию XAML (Disable XAML Validation)	Указатель, что требуется отключить правила валидации

4.5.2. Копирование контента

Копирование контента выполняется одним из следующих способов:

1) в контекстном меню контента выбрать пункт «Создать по образцу» (Create Like),
2) в списке контентов выбрать требуемый контент и нажать кнопку «Создать по образцу» (Create Like).

В результате QP создаёт новый контент с именем *Имя исходного контента* *Номер копии*.

Примечание: копирование статей контента не осуществляется. В случае необходимости копирование статей может быть осуществлено путём их импорта из исходного контента.

4.5.3. Изменение свойств контента

Изменение свойств контента выполняется одним из следующих способов:

1) в контекстном меню контента выбрать пункт «Свойства» (Properties),
2) в списке контентов выбрать требуемый контент и нажать кнопку «Свойства» (Properties).

4.5.4. Очистка контента

Очистка контента позволяет удалить все существующие в контенте статьи.

Примечание: очистка контента может использоваться, например, для устранения результатов неудачного импорта статей.

Действие является многошаговым, вызывается из контекстного меню контента (пункт «Очистить» (Clear)).

4.5.5. Удаление контента

Удаление контента выполняется одним из следующих способов:

1) в контекстном меню контента выбрать пункт «Удалить» (Remove),
2) в списке контентов выбрать требуемый контент и нажать кнопку «Удалить» (Remove).

4.6. Виртуальный контент (Virtual Content)

Виртуальный контент не содержит собственных статей, а получает данные из других контентов или таблиц БД. На уровне БД виртуальный контент является представлением.

Доступны виртуальные контенты следующих типов:

- JOIN,
- UNION,
- USER QUERY.

4.6.1. Виртуальный контент типа «JOIN»

Виртуальный контент типа «JOIN» позволяет сформировать виртуальный контент на базе существующего контента-источника с самостоятельно заданным набором полей контента. При наличии в контенте-источнике полей «Связь» типов O2M и M2M допускается выбрать поля из связанных контентов.

Примечание: от использования JOIN-контентов можно отказаться в пользу генерируемых классов LINQ to SQL.

4.6.2. Виртуальный контент типа «UNION»

Виртуальный контент типа «UNION» позволяет объединить несколько контентов одного сайта в один виртуальный контент.

Примечание: для объединения контентов из разных сайтов нужно использовать тип «USER QUERY»).

По умолчанию в результирующий контент попадают все поля всех составляющих контентов-источников. Сопоставление полей производится по имени, но при этом производится проверка типов. Если в объединяемых контентах есть два поля с одним именем и несовместимым типом, то UNION-контент не может быть создан. Если в объединяемом контенте поле отсутствует, то в виртуальном на этом месте будет `NULL`.

4.6.3. Виртуальный контент типа «USER QUERY»

Для виртуального контента типа «USER QUERY» пользователь самостоятельно задаёт SQL-запрос на получение требуемых данных. В качестве источника данных допускается использовать не только контенты, но и произвольные таблицы БД.

Внимание: для корректного функционирования SQL-запрос должен возвращать служебные поля QP.

Имя поля	Тип данных SQL	Описание	Рекомендуемое значение
CONTENT_ITEM_ID	Numeric	Уникальный идентификатор статьи	Auto increment
CREATED	Datetime	Дата создания статьи	-
MODIFIED	Datetime	Дата последней модификации статьи	-
LAST_MODIFIED_BY	Numeric	Пользователь, последним модифицировавший статью	1 (Administrator)

STATUS_TYPE_ID	Numeric	ID текущего статуса статьи	ID статуса `Published`
VISIBLE	Numeric	Флаг видимости	1
ARCHIVE	Numeric	Флаг нахождения в архиве	0

4.6.4. Создание виртуального контента

Для создания виртуального контента требуется в контекстном меню раздела «Виртуальные контенты» (Virtual Contents) выбрать пункт «Новый виртуальный контент» (New Virtual Content).

4.6.5. Изменение свойств виртуального контента

Изменение свойств виртуального контента выполняется одним из следующих способов:

1) в контекстном меню виртуального контента выбрать пункт «Свойства» (Properties),
2) в списке виртуальных контентов выбрать требуемый контент и нажать кнопку «Свойства» (Properties).

4.6.6. Удаление виртуального контента

Удаление виртуального контента выполняется одним из следующих способов:

1) в контекстном меню виртуального контента выбрать пункт «Удалитть» (Remove),
2) в списке виртуальных контентов выбрать требуемый контент и нажать кнопку «Удалить» (Remove).

4.7. Поле (Field)

После создания контента необходимо задать его структуру. Для этого применяются поля.

Примечание: при создании контента автоматически создаётся необязательное поле `Title` типа Строка. Допускается его изменение или удаление.

В ГПИ бэкенда и контекстного меню доступны следующие функции управления сайтами:

1) создание нового поля,
2) копирование поля,
3) изменение свойств поля,
4) удаление поля.

4.7.1. Создание нового поля

Создание поля выполняется одним из следующих способов:

1) выбрать раздел «Поля» (Fields), нажать «Добавить новое поле» (New Field);
2) в контекстном меню раздела «Поля» выбрать пункт «Новое поле» (New Field).

Существует возможность автоматически расположить новое поле в списке полей после другого существующего поля.

Примечание: в форме создания поля будет автоматически задано значение свойства «Разместить» (Place).

Для создания смежного поля в контекстном меню существующего поля необходимо выбрать пункт «Новое смежное поле» (New Adjacent Field).

Свойства поля

Общие свойства (Basic parameters)

Название	Описание
Имя (Name)	Имя поля контента. **Примечание:** значение должно быть уникально в пределах контента.
Дружественное имя (Friendly Name)	Используется для задания понятного пользователю имени поля. Используется только в ГПИ бекэнда, выводится вместо значения «Имя поля».
Описание (Description)	Описание поля.
Тип поля (Field Type)	Выбор типа поля из списка поддерживаемых типов. При изменении существующего поля в списке возможных типов отображаются только совместимые типы.
Разместить (Place)	Расположение поля в списке полей в ГПИ бекэнда. Задаётся относительно существующих полей.
Использовать в фильтре дочернего контента (Use in child content filter)	Указатель, что поле должно быть доступно в качестве параметра фильтрации для фильтра данных в списке статей дочернего контента.
Является полем локализации (Is localization field)	Указатель, что поле требуется использовать в мультиязычном окружении. **Примечание:** используется при генерации классов LINQ to SQL.
Обязательное (Required)	Указатель, что пользователю перед сохранением статьи обязательно будет нужно заполнить поле. Поддерживается для всех типов полей, кроме: • «Булевый», • «Динамическое изображение». Опция поддерживается QP8 API.
Только чтение (Read-only)	Включает режим «Только чтение». Доступно только в ГПИ бекэнда.
Уникальное (Unique)	После установки данного флага значения поля в статьях будут проверяться на уникальность в пределах контента. **Примечание:** • Уникальность поля проверяется в действующих контентах. Т.е. в архивных контентах уникальность не проверяется. • Пустое поле также считается уникальным
В комбинации с (In combination with)	Позволяет задать правило уникальности на несколько полей.
Индексированное (Indexed)	Создает индекс на поле. Работает для всех типов полей, кроме: • «Булевый», • «Текстовое окно», • «Визуальный редактор», • «Связь» типа M2M. **Примечание:** для поля «Связь» типа O2M индекс создается всегда. **Примечание:** в SQL Server существует ограничение на максимальный размер индекса (900 байт, 450 символов в строке NVARCHAR).
Просмотреть в списке (View in list)	Позволяет выводить значение поля на странице со списком статей контента. **Примечание:** для полей «Текстовое окно» и «Визуальный редактор» выводятся первые 500 символов значения. **Примечание:** для полей «Связь» выводятся первые 4 значения (сортировка по идентификатору).

Разрешить OnScreen (Enable OnScreen)	**Внимание:** режим OnScreen существует только в предыдущих версиях продукта. Указатель, должна ли быть возможность работы с полем в режиме OnScreen. Включена по умолчанию для полей типа: • «Строка», • «Текстовое окно», • «Визуальный редактор». ГПИ выводится при каждом вызове метода `Field`. Если это нежелательно, то следует использовать метод `FieldNS`. Режим доступен всех типов полей, кроме: • «Связь» типа M2M, • «Динамическое изображение».
Пересобрать виртуальные контенты (Rebuild virtual contents	**Внимание:** отключение автоматической пересборки виртуальных контентов требует оперативной пересборки содержащего поле контента в ручном режиме (из контекстного меню для контента). Указатель, требуется ли выполнять автоматическую пересборку виртуальных контентов, для которых поле является базовым.

Параметры отображения в LINQ-классы (LINQ Mapping Parameters)

Название	Описание
Отображать как LINQ-свойство (Map as LINQ Property)	Указатель, что требуется генерация свойства для поля.
Имя LINQ-свойства (LINQ Property Name)	Допустимое в C# имя поля. Используется в качестве названия свойства.

Для поля «Связь» типа O2M доступны следующие дополнительные свойства:

Название	Описание
Имя обратного LINQ-свойства (LINQ Back Property Name)	Название свойства в генерируемой модели, указывающее на текущую сущность. Доступно только в том случае, если для «Связи» типа O2M не существует обратной. При создании обратной «Связи» типа M2O через функциональность Имя обратного поля (Backward field name) текущее значение для «Связи» типа O2M будет скопировано в Имя LINQ-свойства (LINQ Property Name) «Связи» типа M2O

Для поля «Связь» типа M2M доступны следующие дополнительные свойства:

Название	Описание
Отображать узловую таблицу как класс (Map Junction Table as Class)	При включённой опции производится генерация класса для узловой таблицы.
Имя узлового LINQ-класса (единственное) (LINQ Junction Class Name (singular))	Допустимое в C# имя узловой таблицы, хранящей данные M2M-связи (в единственном числе). Используется в качестве названия класса.
Имя узлового LINQ-класса (множественное) (LINQ Junction Class Name (plural))	Допустимое в C# имя узловой таблицы, хранящей данные M2M-связи (во множественном числе). Используется в качестве названия свойства контекстного класса, с которого начинается построение запроса LINQ to SQL.

Параметры, зависящие от типа (Type-specific parameters)

Название	Описание
Значение по умолчанию (Default Value)	Значение, которое должно автоматически задаваться в поле при создании новой статьи. Используется для всех типов полей, кроме: • «Связь» типа M2M, • «Динамическое изображение». Опция поддерживается QP8 API. **Примечание:** для того чтобы присвоить значение по умолчанию полю для всех статей контента, необходимо вызвать контекстное меню поля, у которого задано в свойствах значение по умолчанию, и выбрать пункт «Apply Default Value»

Строка (String)

Короткое текстовое поле. Внутренний SQL-тип – NVARCHAR.

Примечание: следует помнить об ограничении в SQL Server на размер индексированного поля (900 байт или 450 символов NVARCHAR). Для больших объёмов текстовых данных рекомендуется использовать тип поля «Текстовое окно» или «Визуальный редактор».

Название	Описание
Размер поля (Size)	Максимальное количество символов для значения поля. Значение по умолчанию – 255 символов.
Использовать маску ввода (Use Input Mask)	Указатель, что для поля требуется применять маску ввода.
Тип маски ввода (Input Mask Type)	Выбор типа маски ввода: • «Базовый» (Basic), • «Настраиваемый» (Custom).
Маска ввода (Input Mask)	Для типа маски «Базовый» предлагается сделать выбор из готовых масок: 1) «E-mail», 2) «No white space», 3) «Numbers only with spaces», 4) «Numbers only without spaces», 5) «Letters only with spaces», 6) «Letters only without spaces». Для типа маски «Настраиваемый» предлагается задать собственное регулярное выражение для маски. Поддерживается QP8 API.

Число (Numeric)

Текстовое поле для ввода целых и вещественных чисел. Внутренний SQL-тип – NUMERIC.

Название	Описание
Целое (Integer)	Указатель, что число является целым.
Десятичные знаки (Decimal Places)	Количество знаков после запятой. **Примечание:** значение по умолчанию – 2.

Булевый (Boolean)

Поле с возможными значениями «Да» и «Нет». Внутренний SQL-тип – NUMERIC.

Дата (Date)

Поле допускает только календарную дату. Внутренний SQL-тип – DATETIME.

Время (Time)

Поле допускает только время. Внутренний SQL-тип – DATETIME.

Дата & время (Date & Time)

Поле, объединяющее возможности двух предыдущих типов. Внутренний SQL-тип – DATETIME.

Файл (File)

Позволяет привязать к статье файл. Внутренний SQL-тип – NVARCHAR(255).

Файл хранится в Библиотеке контента или Библиотеке сайта, в БД хранится только имя файла.

Название	Описание
Использовать Библиотеку сайта (Use Site Library)	Указатель, что следует использовать Библиотеку сайта вместо Библиотеки контента. По умолчанию отключён. Поддерживается QP8 API.
Переименовывать совпадающие файлы (Rename Matched Files)	Указатель, что бекэнд должен выполнять автоматическое переименование нового загружаемого в Библиотеку файла в случае, если его имя совпадает с именем файла, уже существующего в Библиотеке. При переименовании к исходному имени файла добавляется число в скобках до тех пор, пока в Библиотеке не будет найдено файла с полученным именем. Если свойство неактивно, то у пользователя запрашивается подтверждение на замену существующего файла.
Отключить контроль версий (Disable Version Control)	Рекомендуется использовать для полей, которые могут содержать большие файлы. Отключение позволяет снизить: 1) нагрузку на дисковую систему в случае использования репликации и резервного копирования, 2) потребление дискового пространства. Отключение контроля версий относится только к самому файлу, значения в БД (название файла) по-прежнему сохраняются. **Примечание:** контроль версий организован через файловую систему (для хранения файлов для различных версий статей используются отдельные директории).
Подпапка для файлов (File Subfolder)	Предназначено для разделения файловых хранилищ для полей одного контента. Также появляется возможность прозрачного переноса больших файлов на отдельный сервер через механизм символических ссылок (реализовано на уровне ОС в Microsoft Windows Vista и Server 2008). В качестве значения следует указать относительный путь до директории в файловой системе. Поддерживается QP8 API.

Изображение (Image)

Позволяет привязать изображение к статье. Внутренний SQL-тип – NVARCHAR(255).

Свойства идентичны свойствам поля типа «Файл».

Отличия от поля типа «Файл»:

1) наличие функции просмотра изображения в ГПИ бекэнда,
2) возможность создать поле типа «Динамическое изображение» на основе текущего поля.

Текстовое окно (Textbox)

Расширенное поле для ввода текста. Внутренний SQL-тип – NTEXT.

Название	Описание
Строки (Rows)	Число строк в поле ввода. Значение по умолчанию – `5`.

Визуальный редактор (VisualEdit)

Поле, обеспечивающее возможность изменения значения с использованием WYSIWYG-редактора.

Примечание: в качестве редактора используется сторонний программный продукт «CKEditor».

Внутренний SQL-тип – NTEXT.

Отличается от поля типа «Текстовое окно» только на уровне ГПИ бекэнда.

Название	Описание
Высота визуального редактора (Height)	Высота визуального редактора в пикселях. Значение по умолчанию – `450`.
Автоматическое развёртывание (Auto Expand)	Указатель, требуется ли автоматически выводить блок с WYSIWYG-редактором и значением поля при открытии страниц для создания или изменения статьи. Значение по умолчанию – «Отключено» (блок с редактором и значением поля свёрнут).
Как простой текстовый редактор (As Text Editor)	Указатель, что при открытии страницы с полем по умолчанию требуется выводить текстовую область с содержимым поля без его обработки с использованием WYSIWYG-редактора. **Примечание:** в ГПИ пользователю предоставляется возможность подключить WYSIWYG-редактор.
Разрешить целые HTML-страницы (Enable full HTML pages)	Указатель, требуется ли в редакторе возможность изменять содержимое HTML-страницы целиком. Значение по умолчанию – «Отключено» (содержимое вне тега `<body>` не выводится в редакторе).
DocType	Значение элемента `DocType`. **Примечание:** по умолчанию визуальный редактор в качестве значения `DocType` использует `XHTML 1.0 Transitional`.
Вставлять тег P при нажатии Enter (Insert P tag while pressing Enter)	Локальное определение режима работы редактора. • Свойство активно – при переводе строки добавляется тэг `<p>`. • Свойство неактивно – при переводе строки добавляется тэг ` `.
Использовать правила английской типографики (Use English typographic rules)	Локальное определение, требуется ли использовать правила английской типографики. По умолчанию выключено (используются правила русской типографики). **Примечание:** реализовано с использованием плагина визуального редактора.
Команды (Commands)	Локальное определение доступной пользователям функциональности для редактора. **Примечание:** в список включены команды объектов “Плагин визуального редактора” (выводятся после списка базовых команд). **Примечание:** для пользователей из группы “Администраторы” свойство не используется (доступны все имеющиеся в редакторе возможности).
Класс корневого элемента (Root Element Class)	Имя CSS-класса для корневого элемента (в тэге `<body>`). **Примечание:** используется при подключении классов с использованием свойства «Внешние CSS».
Внешние CSS (External CSS)	URL для CSS-файлов, которые требуется использовать в редакторе.

Стили (Styles)	Локальное задание объектов «Стиль визуального редактора», доступных пользователю в редакторе. Используются объекты с неактивным свойством «Формат».
Форматы (Formats)	Локальное задание объектов «Стиль визуального редактора», доступных пользователю в редакторе. Используются объекты с активным свойством «Формат».

Связь (Relation)

Поле используются для задания связей между контентами. Доступные следующие типы связей:

Название	Дополнительные обозначения	Описание
Один-ко-многим	• One-To-Many • O2M	У одного родительского элемента может быть множество дочерних элементов. **Примечание:** допускается выбор поля O2M в качестве поля отображения для другого поля O2M. Используется в случае, когда существует ряд контентов: *Контент3* -> *Контент2* -> *Контент1* При этом: • контенты ссылаются друг на друга с помощью связи O2M; • в контенте *Контент2* нет подходящего поля для заголовка, а в контенте *Контент1* – есть. В качестве выводимого поля для связи *Контент3* -> *Контент2* можно задать поле связи *Контент2* -> *Контент1*. После этого в списках и формах для поля связи *Контент3* -> *Контент2* будет выводиться заголовок из контента *Контент1*.
Многие-ко-многим	• Many-To-Many • M2M	У множества родительских элементов может быть множество дочерних элементов.
Многие-к-одному	• Many-To-One • M2O	Противоположность типу O2M. Во многих случаях такой подход удобнее, так как позволяет задать одного родителя сразу для многих статей. Можно создать только для уже существующего O2M-поля

Связь типа O2M

Внутренний SQL-тип – NUMERIC.

Примечание: связь осуществляется по значению служебного свойства `CONTENT_ITEM_ID`. В БД хранятся только значения служебного свойства `CONTENT_ITEM_ID` статей связанного контента.

Название	Описание
Связать с (Relate To)	Контент, с которым требуется установить связь.
Показать поле (Display Field)	Выбор поля из связанного контента, название которого которое должно выводиться в ГПИ бекэнда.
Использовать условие на связь (Use relation condition)	Указатель, что требуется применять дополнительный фильтр (условие на связь) для статей контента перед тем, как они будут выведены пользователю.
Условие на связь (Relation Condition)	Значение для условия. **Примечание:** условие задаётся с использованием синтаксиса SQL. Пример записи: `c.[Title] is not null`

Использовать связанные права доступа (Use related permissions)	Указатель, что требуется использовать права доступа на связи.
Максимальное количество элементов (The maximum number of items)	Настройка, влияющая на вид контрола, в зависимости от количества элементов. Если значение больше указанного, то отображается SingleItemPicker, иначе – DropDownList
Имя обратного поля (Backward field name)	Доступно только в том случае, если не существует обратного поля связи типа M2O. Позволяет создать обратное поле из интерфейса текущего.
Обратное поле (Backward field)	Доступно только в том случае, если существует обратное поле связи типа M2O. Ссылка на обратное поле.
Использовать для фильтрации по умолчанию (Use for Default Filtration)	Указатель, что поле следует использовать для функциональности «Фильтрация по умолчанию».

В случае, когда в качестве контента для связи задан содержащий поле контент, доступны следующие свойства:

Название	Описание
Использовать для дерева (Use for tree)	**Внимание:** свойство может быть активно только у одного поля контента. Указатель, что поле требуется использовать для построения иерархической структуры статей в контенте для ГПИ бекэнда. **Примечание:** по умолчанию для построения иерархической структуры используется первое найденное в контенте поле «Связь» типа O2M, ссылающееся на текущий контент. **Примечание:** значение учитывается в виртуальных контентах, что позволяет использовать в них иерархическое представление.
Автоматически отмечать дочерние статьи (Auto check child articles)	**Внимание:** режим относится к случаю, когда в ГПИ открыто всплывающее окно выбора статей и в нём выбран режим дерева. Указатель, что операции выбора и отмены выбора элементов должны осуществляться синхронно для родительского и дочерних элементов.
Копировать права доступа при создании дочерних элементов (Copy permissions while creating children)	Указатель, что при создании статьи в дочернем контенте для неё должны быть заданы права доступа для родительского контента.
Поле для сортировки в дереве (Tree Sorting Field)	Поле, по значению которого требуется осуществлять сортировку статей в иерархической структуре статей.
Сортировка по умолчанию по заголовку (Default sorting by title)	Указатель, что сортировка статей в иерархической структуре статей должна выполняться по значению сформированного заголовка для элемента. **Примечание:** если отключено, то используется сортировка по значению идентификатора.
Число полей, выводимых в заголовке элемента дерева (Number of fields included in tree item title)	Количество полей, значения из которых требуется выводить в названии каждого из элементов в иерархической структуре статей.
Включать поля связи в заголовок элемента дерева (Include relation fields in tree item title)	Указатель, что название элемента в иерархической структуре статей должно формироваться с учётом полей «Связь».

В случае, когда в качестве контента для связи задан другой контент, доступны следующие свойства:

Название	Описание
Агрегированное (Aggregated)	Указатель, что контент является агрегированным
Классификатор (Classifier)	Поле типа «Классификатор» из родительского контента
Число полей, выводимых в заголовке элемента списка (Number of fields included in list item title)	Количество полей, значения из которых требуется выводить в названии каждого из элементов в списке статей
Включать поля связи в заголовок элемента списка (Include relation fields in list item title)	Указатель, что название элемента в списке статей должно формироваться с учётом полей «Связь»

Связь типа M2M

Внутренний SQL-тип – NUMERIC.

Примечание: в таблице контента хранится только идентификатор связи. Данные поля хранятся в отдельной узловой таблице.

Название	Описание
Связать с (Relate To)	Контент, с которым требуется установить связь.
Использовать условие на связь (Use relation condition)	Указатель, что требуется применять дополнительный фильтр (условие на связь) для статей контента перед тем, как они будут выведены пользователю.
Условие на связь (Relation Condition)	Значение для условия. **Примечание:** условие задаётся с использованием синтаксиса SQL. Пример записи: `c.[Title] is not null`
Использовать связанные права доступа (Use related permissions)	Указатель, что требуется использовать права доступа на связи.
Максимальное количество элементов (The maximum number of items)	Настройка, влияющая на вид контрола, в зависимости от количества элементов. Если значение больше указанного, то отображается MultipleItemPicker, иначе – CheckboxList
Имя обратного поля (Backward field name)	Доступно только в том случае, если не существует обратного поля связи типа M2M. Позволяет создать обратное поле из интерфейса текущего.
Обратное поле (Backward field)	Доступно только в том случае, если существует обратное поле связи типа M2M. Ссылка на обратное поле.
Использовать для фильтрации по умолчанию (Use for Default Filtration)	Указатель, что поле следует использовать для функциональности «Фильтрация по умолчанию».
Симметричное (Symmetric)	Указатель, требуется ли использовать двустороннюю связь. Значение по умолчанию – «Включено». Функциональность поддерживается в QP8 API и классах LINQ to SQL.
Оптимизировать для иерархии (Optimize for hierarchy)	Указатель, требуется ли использовать дополнительные правила для оптимизации хранения данных о иерархической структуре в БД. Используемые правила: • при выборе родительского элемента иерархии хранится только запись о нём (если в ГПИ

	пользователем выбран родительский элемент иерархии, то на всех уровнях вложенности отменяется выбор его дочерних элементов); • если пользователь в ГПИ выбирает все дочерние элементы и не выбирает родительский, то сохраняется запись о родительском элементе (в ГПИ отменяется выбор дочерних элементов, выбирается родительский).
Поле для сортировки в списке (List Sorting Field)	Поле, по значению которого требуется осуществлять сортировку статей в списке статей.
Сортировка по умолчанию по заголовку (Default sorting by title)	Указатель, что сортировка статей в списке статей должна выполняться по значению сформированного заголовка для элемента. **Примечание:** если отключено, то используется сортировка по значению идентификатора.
Число полей, выводимых в заголовке элемента списка (Number of fields included in list item title)	Количество полей, значения из которых требуется выводить в названии каждого из элементов в списке статей.
Включать поля связи в заголовок элемента списка (Include relation fields in list item title)	Указатель, что название элемента в списке статей должно формироваться с учётом полей «Связь».

Односторонняя связь для поля «Связь» типа M2M

С помощью свойства «Симметричное» существует возможность ввести направление связи.

Если для одной связи между контентами используется два поля (по одному полю в контенте), то изменение значения свойства «Симметричное» в одном поле приведёт к его автоматическому изменению в другом поле.

Если свойство активно, то реализуется поведение, которое было раньше. Если же его выключить, то можно реализовать поведение, чтобы статья из первого контента ссылалась на статью из второго контента, но статья из второго контента не ссылалась на статью из первого. Это особенно актуально для связи M2M, обе стороны которой указывают на один и тот же контент (например, контент «Новости» и поле «Смотри также»).

Связь типа M2O

Поле «Связь» типа M2O является обратным к типу O2M. Тип M2O не предполагает хранение каких-либо новых данных. По сути, осуществляется изменение значения поля «Связь» типа O2M со стороны родителя. Во многих случаях такой способ более удобен, так как позволяет задать одного родителя сразу для многих дочерних статей.

С точки зрения ГПИ форма для работы с полем выглядит аналогично форме для поля типа M2M. Отличие в том, что для поля M2O среди доступных к привязке статей выводятся статьи связанного контента, которые по данным обратного O2M-поля либо уже привязаны к текущей статье, либо еще не привязаны ни к одной.

Поле может быть создано следующими способами:

Название	Описание
Задать базовое поле связи	В качестве значения свойства «Базовое поле связи» требуется указать O2M-поле, ссылающееся на текущий контент

Создать M2O-поле, как обратное, в форме создания изменения O2M-поля	Имя поля следует указать в свойстве «Имя обратного поля». **Примечание:** параметр «Имя LINQ-свойства» M2O-поля будет скопирован из параметра «Имя обратного LINQ-свойства» O2M-поля.

Свойства:

Название	Описание
Базовое поле связи (Base relation field)	Поле «Связь» типа O2M, которое требуется использовать для создания связи
Поле для сортировки в списке (List Sorting Field)	Поле, по значению которого требуется осуществлять сортировку статей в списке статей
Сортировка по умолчанию по заголовку (Default sorting by title)	Указатель, что сортировка статей в списке статей должна выполняться по значению сформированного заголовка для элемента. **Примечание:** если отключено, то используется сортировка по значению идентификатора.
Число полей, выводимых в заголовке элемента списка (Number of fields included in list item title)	Количество полей, значения из которых требуется выводить в названии каждого из элементов в списке статей
Включать поля связи в заголовок элемента списка (Include relation fields in list item title)	Указатель, что название элемента в списке статей должно формироваться с учётом полей «Связь»

Преобразование типов

Существует возможность изменить тип с O2M на M2M и обратно (если позволяют данные) с сохранением данных. Если для связи между контентами существует пара полей M2M с каждой стороны связи, то они будут преобразованы в пару полей O2M и M2O.

Динамическое изображение (Dynamic Image)

Поле предоставляет возможность автоматического создания миниатюры (thumbnail) изображения при создании или изменении изображения из поля «Изображение». Внутренний SQL-тип – NVARCHAR(255).

На основе одного поля «Изображение» допускается создание множества полей «Динамическое изображение».

Создаваемый файл хранится в Библиотеке контента.

Название	Описание
Основано на (Based on)	Поле «Изображение» из текущего контента, на основе которого требуется создавать миниатюру
Режим (Mode)	Выбор режима для создания изображения. Доступные режимы: 1) «Ограничить размеры» (Limit Size), 2) «Ограничить по высоте» (Limit Height), 3) «Ограничить по ширине» (Limit Width).
Высота (Height)	Высота изображения в пикселях. Используется в режимах: 1) «Ограничить размеры», 2) «Ограничить по высоте». Значение по умолчанию – `100`.
Ширина (Width)	Ширина изображения в пикселях. Используется в режимах:

	1) «Ограничить размеры», 2) «Ограничить по ширине». Значение по умолчанию – `100`.
Тип файла на выходе (Output File Type)	Тип создаваемого файла. Доступные типы: • JPG, • PNG, • GIF.
Качество (%) (Quality (%))	Значение для качества изображения. Используется с типом файла «JPG».

Классификатор (Classifier)

Поле «Классификатор» позволяет создать агрегированную связь между контентами, при которой статьи дочернего контента существуют только вместе со статьями родительского контента и не могут быть созданы и изменены сами по себе. Все неинтерфейсные операции с такими статьями также нужно выполнять через родительскую сущность.

Отношение агрегации реализовано в варианте «Один-к-одному». В типовом сценарии использования на один родительский контент ссылается несколько дочерних контентов, каждая из этих связей является отношением агрегации. В таком сценарии дочерние контенты называются контентами-расширениями (фактически реализуется шаблон Multiple table inheritance).

Поле «Классификатор» создаётся в родительском контенте. Значение поля для конкретной статьи указывает на то, какой контент-расширение используется. Таких полей (а значит и наборов контентов-расширений для родительского контента) может быть несколько. При этом контент-расширение может иметь только одну агрегированную связь, то есть быть расширением только к одному родительскому контенту.

Чтобы настроить агрегирование, необходимо выполнить следующие действия:

1) в родительском контенте создать поле «Классификатор»,
2) создать нужное количество контентов-расширений. В каждом из этих контентов нужно создать поле «Связь» типа O2M, сослаться на созданное поле классификатора и включить опцию «Агрегированное».

В ГПИ бекэнда при создании/изменении статьи значение для поля «Классификатор» задаётся с помощью выпадающего списка, в качестве элементов которого используются названия подготовленных контентов-расширений. После выбора значения в форму создания/изменения статьи загружаются все поля выбранного контента-расширения (кроме поля «Связь» типа O2M, которое реализует отношение агрегации).

Свойства поля:

Название	Описание
Изменяемое поле (Changeable)	Указатель, разрешено ли изменение значения поля после создания статьи
Использовать права доступа по типам (Use type security)	Указатель, требуется ли ограничить права доступа по типу элемента (страница, виджет)

Строковое перечисление (String Enum)

Тип поля реализован на базе типа «Строка». При создании поля следует задать набор строковых значений. Используется в бекэнде при создании и изменении статьи. Пользователю предоставляется возможность выбрать одно значение из набора.

В БД сохраняются строковые значения.

Свойства поля:

Название	Описание
Использовать группу радиокнопок (Show as Radio Button Group)	Указатель, что для вывода набора в ГПИ требуется использовать группу переключателей. По умолчанию отключено. В этом случае в ГПИ используется выпадающий список.
Значения (Values)	Данные для значения. Для значения допускается задание следующих параметров: 1) значение, сохраняемое в БД (Value); 2) значение, выводимое в ГПИ (Alias); 3) указатель, что значение следует выводить в ГПИ по умолчанию (Default).

4.7.2. Копирование поля

Копирование поля выполняется одним из следующих способов:

1) в контекстном меню поля выбрать пункт «Создать по образцу» (Create Like),
2) в списке полей выбрать требуемый сайт и нажать кнопку «Создать по образцу» (Create Like).

В результате QP создаёт новый контент с именем `Copy of` *Имя исходного контента*.

4.7.3. Изменение свойств поля

Изменение свойств поля выполняется одним из следующих способов:

1) в контекстном меню поля выбрать пункт «Свойства» (Properties),
2) в списке полей выбрать требуемое поле и нажать кнопку «Свойства» (Properties).

4.7.4. Удаление поля

Удаление поля выполняется одним из следующих способов:

1) в контекстном меню поля выбрать пункт «Удалить» (Remove),
2) в списке полей выбрать требуемое поле и нажать кнопку «Удалить» (Remove).

4.8. Статья (Article)

Статьи содержат данные контентов в соответствии со структурой, заданной с помощью полей. В БД статья является записью в таблице, используемой для контента.

Управлять статьями можно с помощью:

1. ГПИ бэкэнда (вкладка со списком статей и со свойствами статьи);
2. Контекстного меню, вызванного для статьи. Команды:
   a. «Создать по образцу». Команда создает копию статьи;
   b. «Архивировать». Команда архивирует статью;
   c. «Удалить». Команда удаляет статью после ее подтверждения;
   d. «Свойства». Команда открывает вкладку со свойствами статьи;
   e. «История изменений». Команда выводит вкладку с историей изменений;
   f. «Версии». Команда выводит вкладку с версиями статей;
   g. «Права доступа». Команда открывает вкладку, задающую права доступа к текущей статье.

Взаимодействие со статьями доступно так же с помощью QP API (см. QP8 API).

С помощью расписания видимости существует возможность задать условия, при которых статья должна выводиться на веб-сайте.

Примечание: дополнительные данные по работе с расписаниями приведены в руководстве администратора.

Расписание видимости автоматически работает при доступе к контентам через объект типа «Publishing Container» (если не отключено в его настройках). Если нужна работа расписания при прямом доступе к БД, то необходимо фильтровать статьи по условию `visible = 1`.

4.8.1. Служебные свойства статьи

Название	Описание
CONTENT_ITEM_ID	Уникальный идентификатор статьи. **Примечание:** используется сквозная нумерация по всем контентам в пределах БД. **Примечание:** на это поле ссылаются поля типа «Связь».
CREATED	Дата создания статьи
MODIFIED	Дата последнего изменения статьи
ARCHIVE	Указатель, находится ли статья в Архиве статьей
VISIBLE	Указатель, доступна ли статья на веб-сайте. Используется функцией расписания.
LAST_MODIFIED_BY	Идентификатор пользователя, который последним изменил содержимое статьи
STATUS_TYPE_ID	Идентификатор статуса Workflow статьи

4.9. Архивная статья (Archive Article)

Архив статей предназначен для статей, содержимое которых уже не требуется выводить на веб-сайте, но ещё может оказаться полезным в будущем, поэтому удалять их нельзя.

Работа с архивными статьями осуществляется в разделе «Архивные статьи» в контенте.

Поместить статью в архив можно с использованием ГПИ:

- «Архивировать» (Move to Archive) в списке статей,
- «Архивировать» (Archive) на странице свойств статьи.

Примечание: статья может попасть в архив и при её удалении в случае, если в свойствах контента активировано свойство «Архивировать при удалении» (Archive on Removal).

Статья в архиве:

- может быть удалена,
- не может быть отредактирована (предварительно её следует восстановить).

Восстановить статью из архива можно двумя способами:

- «Восстановить» (Restore Selected) в списке архивных статей,
- «Восстановить» (Restore) на странице свойств архивированной статьи.

Примечание: команда «Восстановить» доступна в контекстном меню.
Также контекстное меню, для статьи, содержит команды:

- «Удалить». Команда удаляет статью после ее подтверждения;
- «Свойства». Команда открывает вкладку со свойствами статьи.

При доступе к контенту через объект «Publishing Container» архивные статьи по умолчанию не показываются (может быть изменено в свойствах объекта). При прямом доступе к БД, если необходимо исключить архивные статьи, следует использовать условие `archive = 0`.

Взаимодействие с архивными статьями доступно так же с помощью QP API (см. QP8 API).

4.10. Уведомление (Notification)

QP позволяет настроить автоматическое формирование уведомлений о различных событиях, связанных с изменением содержимого контента.

ГПИ инструмента доступен в разделе «Уведомления» (Notifications) контента.

Внимание: для создания электронных писем с уведомлениями в конфигурационном файле QP должны быть указаны данные для работы с SMTP-сервером, ответственным за отправку электронных писем.

Примечание: с точки зрения производительности целесообразно создавать по одному уведомлению на каждое событие, а всех получателей объединять в группу или передавать адреса электронной почты через точку с запятой.

4.10.1. Архитектура уведомлений

- Метод `SendNotification` (реализует алгоритм отправки уведомлений),
- Компонент сборки (используется для создания динамической страницы уведомления),
- Компонент HTTP-загрузки (используется для получения тела сообщения из динамической страницы),
- Почтовый компонент (отправка почтового сообщения),
- Глобальные настройки (хранятся в конфигурационном файле),
- Локальные настройки (хранятся в таблице `NOTIFICATIONS`).

4.10.2. Обобщённый алгоритм работы уведомлений

1. Явно (из кода формата) или неявно (в результате действий пользователя) вызывается метод `SendNotification`.
2. Данным методом определяются уведомления, которые подходят по типу события.
3. Для каждого уведомления производится:
    a. сборка формата в динамическую страницу;
    b. загрузка динамической страницы с нужными параметрами (идентификатор статьи) и инициализация тела сообщения;
    c. инициализация всех необходимых полей сообщения;
    d. отправка электронного письма.

4.10.3. Свойства уведомления

Основные свойства (Basic Parameters)

Название	Описание
Имя (Name)	Уникальный идентификатор уведомления (в пределах контента). Кроме этого, данное значение подставляется в поле «Тема отправляемого письма».
Внешнее (External)	Указатель, что уведомление должно быть обработано сторонней Системой.
Внешний URL (External URL)	URL для запроса к сторонней Системе, ответственной за обработку уведомления.

Использовать сервис для отправки (Use service for sending)	Указатель, требуется ли использовать службу формирования внешних уведомлений.
Создать формат по умолчанию (Create Default Format)	Доступно только при создании уведомления. После создания уведомления: 1) В шаблоне «Default Notification Template» будет создан объект типа «Publishing Container»; 2) В созданном объекте будет сгенерирован формат, связанный с уведомлением.
Формат (Format)	Выбор формата. Предлагаются все связанные с контентом форматы, принадлежащие объектам типа «Publishing Container».

События (Events)

Поддерживаются следующие типы событий:

Название	Описание события

Создание (Create)	Статья создана.
	Срабатывает при:
	• создании новой статьи через ГПИ бекэнда;
	• создании копии статьи с использованием функции «Создать по образцу» (Create like):
	  ○ через ГПИ бекэнда,
	  ○ в режиме OnScreen бекэнда;
	• вызове метода `AddFormToContent`;
	• вызове метода `SendNotification` с параметром `for_create`.

Изменение (Modify)	Существующая статья изменена. Срабатывает при: • обновлении статьи через ГПИ бекэнда, • вызове метода `UpdateContentItem`, • вызове метода `UpdateContentItemField`, • вызове метода `SendNotification` с параметром `for_modify`.
Удаление (Remove)	Статья удалена. Срабатывает при: • удалении статьи через ГПИ бекэнда, • вызове метода `RemoveContentItem`, • вызове метода `SendNotification` с параметром `for_remove`.
Изменение статуса (Status Changed)	В статье достигнут указанный пользователем статус Workflow. Срабатывает при: • изменении статуса статьи через ГПИ бекэнда, • изменении статуса статьи с помощью метода `UpdateContentItem`, • вызове метода SendNotification с параметром `for_status_changed`. **Примечание:** при изменении статьи через ГПИ бекэнда с одновременным изменением статуса срабатывают два события – «Изменение» и «Изменение статуса».
Частичное изменение статуса (Status Partially Changed)	В статье достигнут указанный пользователем статус Workflow при параллельном Workflow.

Запрос по требованию (Request On Demand)	Произошло событие из собственного кода Разработчика. Срабатывает при вызове метода `SendNotification` с параметром `for_frontend`.
Отложенная публикация (Delayed Publication)	Статья автоматически опубликована по расписанию публикации. Используется с Workflow в режиме расщепления статей.

Отправитель (Sender)

Название	Описание
Использовать имя отправителя по умолчанию (Use Default Sender Name)	Указатель, что требуется использовать имя отправителя по умолчанию. В качестве имени отправителя используется значение `Q-Publishing Backend`. **Примечание:** значение можно изменить в конфигурационном файле QP.
Имя отправителя (Sender Name)	Самостоятельно указанное имя отправителя.
Использовать e-mail пользователя QP8 (Use QP8 User Email)	Указатель, что в качестве значения поля `From:` требуется использовать адрес электронной почты пользователя QP.
E-mail	Самостоятельно указанное значение поля `From:`.
Пользователь QP8 (QP8 User)	Имя пользователя QP, чей адрес электронной почты требуется использовать в качестве значения поля `From:`.

Получатель (Receiver)

Название	Описание
Тип получателя (Receiver Type)	Выбор получателя уведомления. Доступные типы получателей: 1) «Пользователь» (User), 2) «Группа» (User Group), 3) «Все пользователи из истории изменений» (Everyone in History), 4) «Уведомление на поле из статьи» (E-Mail from Article Field), 5) «Не уведомлять» (None). При использовании типа «Все пользователи из истории изменений» уведомление формируется на адреса электронной почты всех пользователей QP, указанных в истории изменений для статьи. Тип «Не уведомлять» используется в случае, если не требуется формировать и отправлять уведомление по электронной почте.
Пользователь (User)	Пользователь QP, на чей адрес электронной почту должно формироваться уведомление.
Группа (User Group)	Группа пользователей QP, все пользователи из которой должны получить уведомление на указанный ими адрес электронной почты.
Уведомление на поле из статьи (E-Mail from Article Field)	Поле контента, содержащее адрес электронной почты, который требуется использовать для формирования уведомления.

Примечание: адрес электронной почты получателя может быть определён в качестве параметра метода `SendNotification`. Данное значение имеет больший приоритет, чем значение, заданное в ГПИ бекэнда.

Дополнительные опции (Options)

Название	Описание

Посылать файлы как вложения (Send files as attachments)	Указатель, что содержимое полей типа «Файл» и «Изображение» следует добавлять в электронное письмо с уведомлением в виде вложений.

4.10.4. Сборка уведомления

Сборка требуемого набора уведомлений осуществляется кнопкой «Собрать» (Assemble).

4.10.5. Внешнее уведомление

Внешнее уведомление позволяет Разработчику самостоятельно обрабатывать события. При создании внешнего уведомления потребуется указать URL, на который должен быть выполнен запрос при наступлении события.

Примечание: при использовании внешнего уведомления QP не осуществляет отправку электронных писем с уведомлениями.

Приёмник внешнего уведомления может быть реализован, например, как служба, которая сбрасывает кэш веб-приложения.

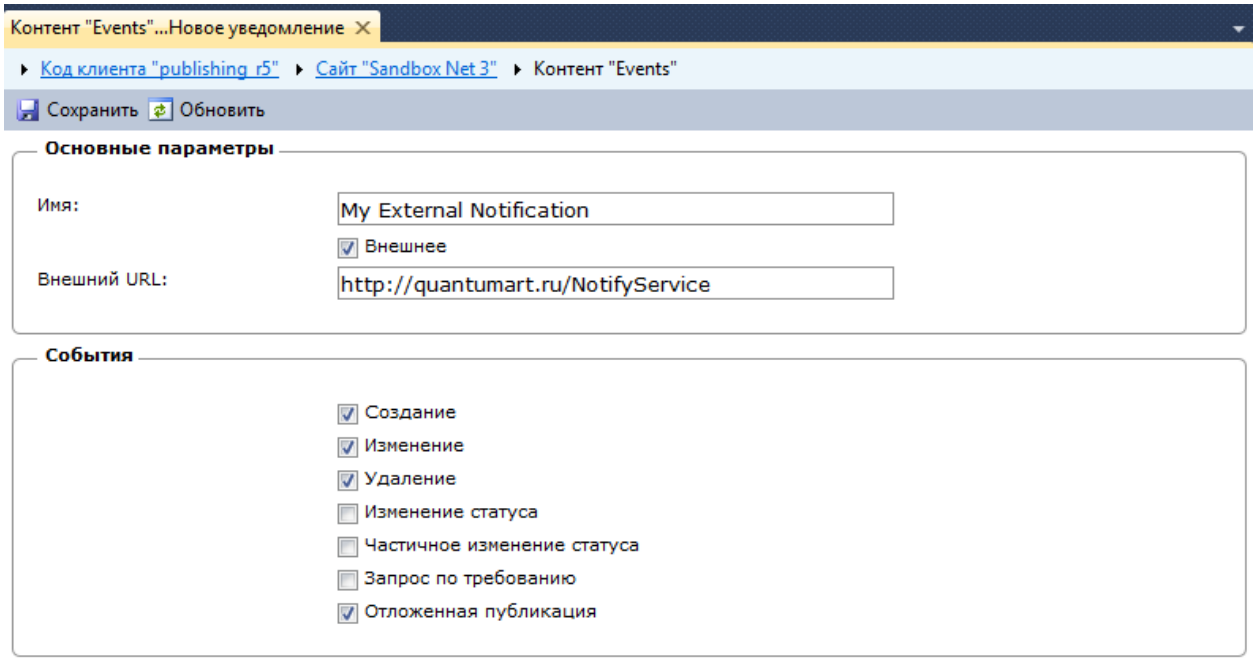

Рисунок 4.10.5-1. Создание внешнего уведомления.

При запросе к URL передаются следующие данные:

- `eventName` (одна из строковых констант, определённых в классе `Quantumart.QPublishing.NotificationEvent`),
- `id`,
- `contentId`,
- `siteId`,
- `visible`,
- `archive`,
- `splitted`,
- `statusName`.

Если произошло событие «Изменение», «Изменение статуса» или «Частичное изменение статуса», то также передаются следующие данные:

- `oldVisible`,

- `oldArchive`,
- `oldStatusName`.

Служба формирования внешних уведомлений

В QP имеется служба, ответственная за отправку запросов на формирование внешних уведомлений.

Если служба не используется, то при наступлении события для уведомления формируется GET-запрос к URL для внешних уведомлений, указанному пользователем. В запросе передаются только служебные данные. В случае, если принимающая запрос сторона не смогла его обработать, то повторный запрос не формируется, что означает потерю уведомления.

Если служба используется, то все задачи на отправку запросов помещаются в очередь и собираются в пакеты. Служба регулярно обрабатывает очередь и осуществляет обращения в внешним URL. Обращения выполняются методом POST, в запросе передаются как служебные данные, так и значения всех полей статьи, с которой связано уведомление. В случае, если принимающая запрос сторона не смогла его обработать, то позднее служба отправляет запрос повторно. Результат работы службы не оказывает влияния на успешность выполнения операции в QP, в результате которой возникает событие, требующее формирования уведомления (например, невозможность сформировать уведомление о создании статьи не помешает операции по созданию этой статьи; в предыдущих версиях продукта успешность операции зависела от формирования уведомления).

4.11. Workflow

Инструмент «Workflow» позволяет организовать автоматизированный документооборот для статей. Каждый Workflow представляет собой набор правил, которые должны быть в определённом порядке применены к статье, чтобы она была опубликована на веб-сайте. Фактически набор правил представляет собой упорядоченный список статусов (этапов Workflow), через которые последовательно должна пройти статья. При достижении максимального статуса Workflow статья считается опубликованной. На каждый этап Workflow назначается пользователь либо группа пользователей, которые получают возможность переводить статью на данный этап.

Создание и изменение Workflow осуществляется в бекэнде на уровне сайта (раздел «Workflows» в дереве сущностей).

Создание Workflow возможно с помощью:

- ГПИ бэкэнда (сущность «Сайт» → сущность «Workflow» → список Workflow → команда «Добавить новый Workflow»);
- Исполнения команды «Создать новый Workflow» в контекстном меню для сущности «Workflow».

Изменение Workflow возможно с помощью:

- ГПИ бэкэнда;
- Контекстного меню для выбранного Workflow. Контекстное меню содержит команды:
  - «Удалить». Команда удаляет Workflow после подтверждения ее вызова;
  - «Свойства». Команда открывает вкладку со свойствами Workflow;
  - «Права доступа». Команда открывает вкладку, задающую права доступа к текущему Workflow.

Назначить Workflow можно для следующих объектов:

Название	Описание

Контент	Workflow применяется ко всем статьям контента (существующим и новым).
Статья	Статья может: 1) наследовать настройки Workflow от контента (поведение по умолчанию), 2) иметь собственный Workflow (даже если на контент не назначен Workflow), 3) не использовать Workflow.

По умолчанию используется режим расщепления статей (Split articles). В этом режиме статья, ранее опубликованная на веб-сайте (получившая максимальный статус), не снимается с публикации после понижения её статуса. Существующая версия статьи выводится на веб-сайте до тех пор, пока статья повторно не будет доведена до максимального статуса. После этого будет опубликована новая версия статьи вместо старой.

В Live-режиме сайта используются только статьи, имеющие максимальный статус. Статьи, для которых не назначен Workflow, всегда имеют максимальный статус.

4.11.1. Статус

Статус используется в Workflow. Создание и изменение статусов осуществляется в бекэнде на уровне сайта (раздел «Статусы» (Statuses) в дереве сущностей).

Существуют следующие системные статусы:

Название	Вес	Описание
None	0	Используется по умолчанию при создании статьи **Примечание:** не может быть использован в Workflow.
Created	10	Статья создана
Approved	50	Статья изменена
Published	100	Статья опубликована

Примечание: создать, изменить или удалить системные статусы невозможно.

Можно создать собственные статусы, присваивая им незанятые веса в интервале значений от 0 до 100. В Workflow могут быть использованы как допустимые системные, так и собственные статусы.

Свойства статуса

Название	Описание
Имя (Name)	Имя статуса
Описание (Description)	Описание статуса
Вес (Weight)	Вес статуса
Цвет (Color)	Цвет для статьи с данным статусом. Используется в ГПИ для списка статей в качестве фонового цвета в блоке с данными по статье. Значения «Цвет» и «Альтернативный цвет» используются для чётных и нечётных статей соответственно. Принимается значение в формате HEX.
Альтернативный цвет (Alternate color)	

4.11.2. Создание и изменение Workflow

Доступные возможности:

1) задание этапов (выбор статусов) Workflow;
2) задание контентов, для статей в которых нужно использовать Workflow.
3) Для каждого этапа:
    a. назначение пользователя или группы пользователей;
    b. задание комментария.

Примечание: при использовании объекта «Publishing Containers» рекомендуется последним статусом делать `Published`, так как объект по умолчанию настроен на вывод статей с данным статусом.

Свойства Workflow

Название	Описание
Имя (Name)	Название Workflow
Описание (Description)	Описание Workflow
Создавать автоматические уведомления (Create Auto Notifications)	Указатель, требуется ли автоматически создать уведомление для Workflow. **Примечание:** относится ко внутренним уведомлениям.
По умолчанию для новых контентов (Default for new contents)	Флаг, определяющий, будет ли текущий workflow выбран в поле Workflow для создаваемого контента
Статусы (Statuses)	Набор статусов для использования в Workflow
Контенты (Contents)	Набор контентов, для которых должен использоваться Workflow. **Примечание:** для выбора доступны только контенты, для которых не задан Workflow.

4.11.3. Использование Workflow

Если на контент или статью назначен Workflow, то в ГПИ бекэнда на странице изменения статьи доступна смена статуса статьи.

При выборе максимального статуса Workflow пользователю становится доступна опция «Отложить публикацию новой версии» (Schedule New Version Publication), которая позволяет установить расписание публикации статьи.

Дополнительные данные

Если на статью назначен Workflow, то вступают в силу дополнительные ограничения прав доступа:

- Только пользователь-участник Workflow (как лично, так и в составе группы) может удалять и изменять статьи, находящиеся в Workflow;
- Пользователь-участник Workflow может удалить статью, находящуюся в Workflow, только если он явно или в составе группы назначен на последний этап Workflow.

Если статья имеет статус, который отсутствует в Workflow, то при следующем изменении её статус будет уменьшен до ближайшего статуса, используемого в данном Workflow.

В случае если пользователь изменяет статью, которая имеет больший статус, чем он может присвоить ей по Workflow, статья автоматически понижается до его максимального статуса.

При создании новой статьи она получает статус:

- `None`, если на контент назначен Workflow;
- `Published`, если Workflow не назначен.

Примечание: при создании копии для статьи, находящейся в Workflow, копия создается со статусом `None`.

4.11.4. Параллельный Workflow

Параллельный Workflow предназначен для ситуаций, когда смена статуса Workflow должна осуществляться только после того, как все пользователи из группы пользователей утвердят это действие.

Использование параллельного Workflow задаётся в для группы пользователей с помощью свойства «Параллельный Workflow» (Parallel Workflow).

Когда статья находится в процессе параллельного утверждения, на странице изменения статьи выводится дополнительный раздел «Параллельный Workflow». Раздел содержит:

1) статус, который должен быть достигнут статьёй в результате параллельного утверждения;
2) список пользователей, которые уже утвердили статью;
3) список пользователей, которые еще не утвердили статью.

Режим параллельного Workflow изменяет поведения стандартных элементов управления Workflow. Смена статуса пользователем не перемещает статью по Workflow на следующий статус, а лишь отмечает данного пользователя, как утвердившего статью на этапе параллельного Workflow. Такое событие называется частичным утверждением статьи. Статья меняет статус после того, как последний пользователь из группы параллельного Workflow утверждает смену статуса.

Если пользователь является Администратором, не входящим в группу параллельного Workflow, он может свободно перемещать статью по Workflow, правила параллельного утверждения на него не распространяются. Если же он является Администратором и входит в группу параллельного Workflow, то никаких дополнительных привилегий на смену статуса он не получает.

Если на последнем этапе Workflow утверждение производится группой параллельного Workflow, то право выставить опцию «Отложить публикацию новой версии» получает последний пользователь из группы, выполняющий утверждение.

4.11.5. Отмена расщепления

Исходные условия для использования функциональной возможности:

1) используется Workflow,
2) включен режим расщепления статей,
3) статья имеет максимальный статус.

Возможные варианты задач:

- Необходимо понизить статус статьи и при этом требуется, чтобы статья не выводилась в Live-режиме, но осталась доступной в Stage-режиме.
- Статус статьи уже понижен, при этом на сайте выводится её предыдущая версия (статья расщеплена) и требуется, чтобы статья не выводилась в Live-режиме, но осталась доступной в Stage-режиме.

Изменение видимости не подходит, так как в этом случае статья не выводится в обоих режимах сайта.

При задании свойства «Отменить расщепление» (Cancel Splitting) статья не будет выводиться в Live-режиме, но продолжит выводиться в Stage-режиме (как если бы они не достигала максимального статуса Workflow).

4.11.1. Комментарий при смене статуса Workflow

Возможность оставить комментарий при смене статуса статьи предназначена для сохранения произвольных данных, связанных с использованием Workflow. Последний созданный комментарий выводится в документе «Форма редактирования сущности».

Примечание: URL в комментарии автоматически преобразовываются в ссылки.

4.11.2. История изменений статуса статьи

История изменений содержит данные о всех изменениях статуса статьи. Доступ к истории изменений выполняется через контекстное меню статьи (пункт «История изменений»).

4.11.3. Удаление Workflow

Удаление Workflow выполняется одним из следующих способов:

1) в контекстном меню Workflow выбрать пункт «Удалить» (Remove),
2) в списке Workflow выбрать требуемый Workflow и нажать кнопку «Удалить» (Remove).

4.12. Пользовательское действие (Custom action)

Пользовательские действия используются для расширения возможностей бекэнда (запуска из бекэнда собственных веб-приложений по URL для их вызова).

Пользовательское действие может быть интерфейсным и неинтерфейсным.

Существует возможность задавать ограничения на сайты и контенты, к которым должно применяться пользовательское действие.

В веб-приложении должна быть выполнена аутентификация в бекэнде с помощью методов класса `QScreen`.

Пользовательские действия добавляются с помощью:

- ГПИ бэкэнда (список статей сущности «Пользовательское действие» → команда «Добавить новое пользовательское действие»);
- Контекстного меню, вызванного для сущности «Пользовательское действие». Меню содержит команду «Новое пользовательское действие», открывающую вкладку с настройками нового пользовательского действия.

Контекстное меню, вызванное для конкретного пользовательского действия, содержит команды:

- «Создать по образцу» (Create Like). Команда создает копию пользовательского действия с именем *Имя пользовательского действия* *Номер копии*.
- «Удалить». Команда удаляет пользовательское действие после ее подтверждения;
- «Свойства». Команда вызывает вкладку со свойствами пользовательского действия.

4.12.1. Свойства пользовательского действия

Название	Описание
Имя (Name)	Имя пользовательского действия. Используется в ГПИ в качестве названия кнопки или пункта контекстного меню.
Описание (Description)	Описание пользовательского действия.
Тип действия (Action Type)	Тип действия, который лучше всего подходит к пользовательскому действию с точки зрения применения прав доступа. Допустимые типы: 1) Список прав доступа (Action Permissions), 2) Добавить (Add New), 3) Изменить все дочерние права доступа (All Child Permission Modify), 4) Удалить все дочерние права доступа (All Child Permission Remove), 5) Архивировать (Archive),

	6) Собрать (Assemble), 7) Собрать родителя (Assemble Parent), 8) Отменить (Cancel), 9) Изменить блокировку (Change Lock), 10) Изменить дочернее право доступа (Child Permission Modify), 11) Удалить дочернее право доступа (Child Permission Remove), 12) Сохранить дочернее право доступа (Child Permission Save), 13) Сравнить (Compare), 14) Создать по образцу (Create Like), 15) Crop, 16) Скачать (Download), 17) Экспорт (Export), 18) Подделка (Fake), 19) Импорт (Import), 20) Библиотека (Library), 21) Список (List), 22) Множественная разблокировка (Multiple Unlock), 23) Переместить (Move), 24) Множественная архивация (Multiple Archive), 25) Множественная сборка (Multiple Assemble), 26) Множественное изменение дочерних прав доступа (Multiple Child Permission Modify), 27) Множественное удаление дочерних прав доступа (Multiple Child Permission Remove), 28) Multiple Export, 29) Множественное удаление (Multiple Remove), 30) Множественное восстановление (Multiple Restore), 31) Множественный выбор (Multiple Select), 32) Multiple Update, 33) Повысить (Promote), 34) Чтение (Read), 35) Удалить (Remove), 36) Восстановить (Restore), 37) Сохранить (Save), 38) Сохранить и вернуться (Save and Up), 39) Поиск (Search), 40) Выбор (Select), 41) Простое обновление (Simple Update), 42) Обновить (Update), 43) Обновить и собрать (Update & Assemble), 44) Обновить и вернуться (Update and Up), 45) Загрузить (Upload).
Тип сущности (Entity Type)	Сущность, с которой связано пользовательское действие. Допустимые сущности: 1) Право доступа к действию (Action Permission), 2) Архивная статья (Archive Article), 3) Статья (Article),

	4) Право доступа к статье (Article Permission), 5) Article Status, 6) Версия статьи (Article Version), 7) Контент (Content), 8) Папка контента (Content Folder), 9) Группа контентов (Content Group), 10) Файл библиотеки контента (Content Library File), 11) Content Link, 12) Право доступа к контенту (Content Permission), 13) Код клиента (Customer Code), 14) Право доступа к типу сущности (Entity Type Permission), 15) Поле (Field), 16) Уведомление (Notification), 17) Страница (Page), 18) Версия формата страницы (Page Format Version), 19) Объект страницы (Page Object), 20) Формат объекта страницы (Page Object Format), 21) Сайт (Site), 22) Папка сайта (Site Folder), 23) Право доступа к папке сайта (Site Folder Permission), 24) Файл библиотеки сайта (Site Library File), 25) Право доступа к сайту (Site Permission), 26) Статус (Status), 27) Стиль (Style), 28) Шаблон (Template), 29) Версия формата шаблона (Template Format Version), 30) Объект шаблона (Template Object), 31) Формат объекта шаблона (Template Object Format), 32) Пользователь (User), 33) Группа пользователей (User Group), 34) Виртуальная статья (Virtual Article), 35) Виртуальный контент (Virtual Content), 36) Виртуальное поле (Virtual Field), 37) Плагин визуального редактора (Visual Editor Plugin), 38) Стиль визуального редактора (Visual Editor Style), 39) Workflow, 40) Право доступа к Workflow (Workflow Permission).
Режим выбора сайтов (Site selection mode)	Доступные режимы: 1) «Показывать на всех сайтах, кроме выбранных» (Show on all sites except selected), 2) «Скрывать на всех сайтах, кроме выбранных» (Hide on all sites except selected).
Выбранные сайты (Selected sites)	Сайты, для которых разработано пользовательское действие.
Режим выбора контентов (Content selection mode)	Доступные режимы: 1) «Показывать во всех контентах, кроме выбранных» (Show in all contents except selected),

	2) «Скрывать во всех контентах, кроме выбранных» (Hide in all contents except selected).
Выбранные контенты (Selected contents)	Контенты, для которых разработано пользовательское действие.
URL	URL, по которому доступно пользовательское действие.
URL иконки (Icon URL)	URL пиктограммы для пользовательского действия. Используется в ГПИ бекэнда.
Порядок (Order)	Порядок вывода пользовательского действия в ГПИ. Значение используется в случае, когда требуется вывести список из нескольких пользовательских действий.
Показывать в контекстном меню (Show in context menu)	Указатель, что вызов пользовательского действия должен быть доступен в контекстном меню для сущности указанного типа.
Показывать на панели инструментов (Show on toolbar)	Указатель, что, вызов пользовательского действия должен быть доступен в ГПИ бекэнда, связанного с сущностью указанного типа.
Для действий (For Actions)	Выбор, на каких экранах ГПИ требуется выводить кнопку для вызова пользовательского действия. **Примечание:** доступные действия зависят от заданного типа сущности.
Интерфейс (Interface)	Указатель, является ли пользовательское действие интерфейсным.
Фраза подтверждения (Confirm Phrase)	Текстовая строка, используемая в качестве запроса у пользователя бекэнда подтверждения на выполнение пользовательского действия. **Примечание:** используется для неинтерфейсного пользовательского действия.
Есть предварительное действие (Has pre-action)	Указатель, что к веб-приложению, содержащему пользовательское действие, необходимо формировать запрос, требуется ли запрашивать у пользователя бекэнда подтверждение на выполнение пользовательского действия.
Окно (Window)	Указатель, что ГПИ для пользовательского действия требуется выводить во всплывающем окне бекэнда. **Примечание:** используется для интерфейсного пользовательского действия.
Ширина окна (Window Width)	Высота окна для ГПИ пользовательского действия. **Примечание:** значение по умолчанию – 500.
Высота окна (Window Height)	Ширина окна для ГПИ пользовательского действия. **Примечание:** значение по умолчанию – 300.

4.12.2. Передача контекста

В Query String передаются следующие параметры:

Название	Описание
backend_sid	Используется для аутентификации.
Параметр контекста	Зависит от типа сущности, указанного в свойствах пользовательского действия. Имя параметра определяется полем `CONTEXT_NAME` в таблице `ENTITY_TYPE`. Примеры: • `site_id` – для сайта; • `content_id` – для контента; • `attribute_id` – для поля; • `content_item_id` – для статьи; • `user_id` – для пользователя; • `usergroup_id` – для группы пользователей.

customer_code	Значение `Customer Code`.

4.12.3. Неинтерфейсное пользовательское действие

Неинтерфейсное пользовательское действие позволяет выполнить какую-либо операцию без вызова ГПИ. Например, можно реализовать, пользовательскую групповую операцию над выбранными статьями – вместо открытия некоторого ГПИ выполняется AJAX-запрос методом POST к заданному URL. В качестве данных должны возвращаться:

Результат	Описание результата
Успех	`null`
Ошибка	`{Type: “Error”, Text: “`*Текст сообщения об ошибке*`”}`

Вызов неинтерфейсного пользовательского действия осуществляется через проксирование на локальном сервере. Это позволяет получать данные о неуспешном выполнении сразу, а не по истечении таймаута, как в случае JSONP. Таким образом, внешнее неинтерфейсное действие должно возвращать ответ в формате JSON (без JSONP-обёртки). Кроме того, URL неинтерфейсного действия должен быть доступен на веб-сервере, на котором запущен бекэнд (из-за локального проксирования запроса).

Примечание: интерфейсные действия должны быть доступны с клиентского компьютера, так как они реализованы через `iframe`.

4.12.4. Подтверждение на выполнение пользовательского действия

В неинтерфейсных пользовательских действиях используется механизм подтверждений. Он позволяет перед выполнением действия запросить у пользователя подтверждение, действительно ли следует выполнить это действие. Подтверждение может быть безусловным и условным.

Безусловное подтверждение используется при любом выполнении действия. Например, при запросе подтверждения для любого действия удаления в ГПИ бекэнда. В качестве текста подтверждения используется значение свойства «Фраза подтверждения».

Условное подтверждение предназначено для вывода подтверждения в зависимости от текущего контекста. Используется, например, для подтверждения удаления статьи, на которую ссылаются другие статьи.

Условное подтверждение реализовано через механизм `PreAction`, поддерживается для URL в формате ASP.NET MVC, заканчивающихся символом `/`. Перед вызовом URL пользовательского действия формируется запрос к URL с суффиксом `PreAction`. Например, если для неинтерфейсного пользовательского действия задан URL `http://`*Домен*`/cms/Custom/Import/`, то запрос будет сформирован для URL `http://`*Домен*`/cms/Custom/ImportPreAction/`. В качестве данных должны возвращаться:

Результат	Описание результата
Подтверждение не нужно	`null`
Подтверждение нужно	`{Type: “Confirm ”, Text: “`*Текст сообщения подтверждения*`”}`

4.13. Стиль визуального редактора (Visual Editor Style)

Стиль визуального редактора позволяет управлять стилями, доступными к использованию при работе с данными в поле типа “Визуальный редактор”.

Управление стилями визуального редактора осуществляется в разделе «Стили визуального редактора» (Visual Editor Styles).

Добавление стиля возможно с помощью:

- ГПИ бэкэнда (список стилей сущности «Стиль визуального редактора» → команда «Добавить новый стиль визуального редактора»);
- Команды «Создать новый стиль виз. редактора», вызванного для сущности «Стиль визуального редактора».

Контекстное меню, вызванное для определенного стиля, содержит команды:

- «Удалить». Команда удаляет стиль визуального редактора после ее подтверждения;
- «Свойства». Команда открывает вкладку со свойствами стиля.

4.13.1. Свойства стиля визуального редактора

Основные параметры (Basic Parameters)

Название	Описание
Имя (Name)	Имя стиля визуального редактора.
Описание (Description)	Описание стиля.
Формат (Format)	Указатель, что элемент содержит форматирование.
Системный (System)	Указатель, что элемент является системным. **Примечание:** системные стили доступны только на чтение (создание, изменение, удаление не допускается).
Порядок (Order)	Приоритет вывода элемента в списке элементов. **Примечание:** используется сортировка по возрастанию значения.

Настройки стиля (Style Settings)

Название	Описание
Тег (Tag)	Тэг, который требуется использовать в визуальном редакторе.
Заменяет тег (Overrides Tag)	Тэг, который требуется заметить при использовании стиля.
HTML - атрибуты (HTML Attributes)	HTML-атрибуты, которые требуется указать в тэге. Для атрибута следует задать параметры: • «Имя» (Name), • «Значение» (Value).
Стили (Styles)	CSS-стили, которые требуется добавить к тэгу с помощью атрибута `style`. Для CSS-стиля следует задать параметры: • «Имя» (Name), • «Значение» (Value).

4.14. Плагин визуального редактора (Visual Editor Plugin)

Плагин визуального редактора позволяет добавлять функциональные возможности, доступные к использованию при работе с данными в поле типа «Визуальный редактор».

Управление плагинами визуального редактора осуществляется в разделе «Плагины визуального редактора» (Visual Editor Plugins). Обращение к плагину осуществляется по URL.

Добавление плагина возможно с помощью:

- ГПИ бэкэнда (список стилей сущности «Плагин визуального редактора» → команда «Добавить новый плагин визуального редактора»);
- Команды «Новый плагин визуального редактора», вызванного для сущности «Плагин визуального редактора».

Контекстное меню, вызванное для определенного плагина, содержит команды:

- «Удалить». Команда удаляет плагин визуального редактора после ее подтверждения;

- «Свойства». Команда открывает вкладку со свойствами плагина.

4.14.1. Свойства плагина визуального редактора

Название	Описание
Имя (Name)	**Внимание:** в качестве значения требуется задавать имя, использованное в коде плагина. Имя плагина визуального редактора.
Описание (Description)	Описание плагина
URL	URL для вызова плагина. В соответствии с требованиями используемого WYSIWYG-редактора «CKEditor» требуется указать URL директории, в которой размещён файл `plugin.js`. Допускается использование относительного пути до файла (без указания протокола и доменного имени). В этом случае для сайта QP требуется создать виртуальный каталог IIS, через который будет доступен код плагина.
Порядок (Order)	Приоритет вывода элемента в списке элементов. **Примечание:** используется сортировка по возрастанию значения.
Команды (Commands)	Для команды требуется указать параметры: • «Имя» (Name), **Внимание:** в качестве значения «Имя» требуется задавать имя, использованное в коде плагина. • «Алиас» (Alias).

4.15. Пользовательская валидация

В базовые возможности QP входят следующие способы валидации данных, вводимых пользователем:

1) ограничение ввода по типу поля,
2) свойство поля «Обязательное»,
3) свойство поля «Уникальное»,
4) для поля типа «Строка»:
   a. маска ввода,
   b. длина значения.

Если требуются дополнительные возможности валидации, то следует использовать пользовательскую валидацию.

4.15.1. Основные принципы

Пользовательская валидация может быть задана на уровне сайта (см. Свойства сайта) или на уровне контента (см. Свойства контента). Используется язык XAML. Валидатор состоит из следующих блоков:

1) описание полей,
2) ресурсный словарь,
3) условия валидации.

4.15.2. Описание синтаксиса

Структура валидатора

```
<XamlValidator xmlns=http://artq.com/validation
               xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<!—Блок описания полей -->
```

```
    <XamlValidator.Definitions> … </XamlValidator.Definitions>
    <ForMember Definition="{x:Definition Name}"> … </ForMember>
    <ForMember Definition="{x:Definition DuplicateName}"> … </ForMember>
    <Must> … </Must>
</XamlValidator>
```

Правила

Название	Описание
Must	Создаёт правило валидации для всей модели (уровня `ValidationSummary`)
ForMember	• Создаёт правило валидации для определённого свойства модели. • Задаёт контекст для всех правил в дереве условий.

Логические операнды

Операнд «And»

Логическое «И». Внутрь можно помещать любые операнды, ветвления или условия в любом количестве. Пример:

```
<And>
    <IsTrue Source="{x:Definition is_mobile}" />
    <IsFalse Source="{x:Definition is_smartphone}" />
    <IsFalse Source="{x:Definition is_tablet}" />
</And>
```

Операнд «If»

Условный оператор. Выполняет действие, если условие принимает значение `true`. Допускается использование условных операторов `If.Then` и `If.Else`. Пример:

```
<If x:Key="condition_alias">
    <IsNullOrEmpty />
    <If.False>
        <If>
            <Matches Expression="^[0-9|a-z|\-|_]+$"
                     RegexOptions="IgnoreCase, CultureInvariant" />
            ...
        </If>
    </If.False>
</If>
```

Операнд «Not»

Операнд отрицания. Внутрь можно помещать только один элемент (любой операнд, ветвление или условие). Пример:

```
<Not>
    <Or>
        <IsTrue Source="{x:Definition is_mobile}" />
        <IsTrue Source="{x:Definition is_smartphone}" />
    </Or>
```

```
</Not>
```

Операнд «Or»

Логическое «ИЛИ». Внутрь можно помещать любые операнды, ветвления или условия в любом количестве. Пример см. выше.

Операнд «RequiredIf»

Сокращённый синтаксис для единичной проверки. Если условие внутри `RequiredIf` выполняется, то устанавливается сообщение об ошибке. Пример:

```
<ForMember Definition="{x:Definition AlternativeName}">
    <RequiredIf Message="Поле AlternativeName Должно быть заполнено, если не заполнено Name">
        <IsNullOrEmpty Source="{x:Definition Name}"/>
    </RequiredIf>
</ForMember>
```

Операнд «Sequence»

Последовательное выполнение условий. Пример:

```
<Sequence StopOnFirstFailure="True">
    <!--поддержка последовательных условий с различными сообщениями-->
    <If>
        <IsNull />
        <If.True>
            <WithMessage Text="Не может быть null" />
        </If.True>
    </If>
    <If>
        <IsNullOrEmpty />
        <If.True>
            <WithMessage Text="Не может быть пустой" />
        </If.True>
    </If>
    <If>
        <Not>
            <Length MinLength="5"/>
        </Not>
        <If.True>
            <WithMessage Text="Длина поля должна быть не менее 5 символов." />
        </If.True>
    </If>
</Sequence>
```

Условия

Условие проверки содержит поле `Source`, в котором должен быть указан псевдоним (alias) поля, значение которого необходимо проверить.

Для части условий существуют дополнительные параметры.

Если выполняются все следующие правила:

- условие находится внутри правила `ForMember`,
- для условия не определено поле `Source`,

то для условия в качестве значения поля `Source` используется значение из правила.

Условие «AreEqual»

Проверка 2 свойств модели на равенство. Пример:

```
<AreEqual Source="{x:Definition password}" Target="{ x:Definition password_copy}" />
```

Условие «Equals»

Проверка равенства параметра заданному значению. Пример:

```
<Equals Source="{x:Definition physical_screen_width}" Value="400" />
```

Условие «GreaterThan»

Сравнение значения со значением, указанным в свойстве `Value`. Пример:

```
<GreaterThan Value="20"/>

<GreaterThan Source="{x:Definition resolution_width}" Value="20"/>
```

Условие «IsFalse»

Проверка, что значение является ложью. Пример:

```
<IsFalse Source="{x:Definition is_tablet}" />
```

Условие «IsTrue»

Проверка, что значение является истиной. Пример:

```
<IsTrue Source="{x:Definition is_smartphone}" />
```

Условие «IsNull»

Проверка, значение параметра на Null.

Условие «IsNullOrEmpty»

Проверка, что параметр имеет значение Null или пустое.

Условие «Length»

Задает длину значения, которое принимает параметр. Пример:

```
<Length MaxLength="512" />
```

Условие «LengthInRange»

Диапазон значений длины, которые принимает параметр. Может содержать параметры `From`, `To`. Пример:

```
<LengthInRange From="1" />
```

Условие «LessThan»

Сравнивает значение поля с указанным значением в свойстве `Value`. Пример:

```
<LessThan Value="320"/>
<LessThan Source="{x:Definition resolution_width}" Value="320"/>
```

Условие «Matches»

Проверяет текстовые значения регулярным выражением. Примеры:

```
<Matches Expression="^samsung\wM[\w]+"
         RegexOptions="IgnoreCase" />
```

```
<Matches Source="{x:Definition marketing_name}"
         Expression="^samsung\wM[\w]+"
         RegexOptions="IgnoreCase" />
```

```
<Must>
    <If>
        <And>
            <!--поддержка логических операндов для условий-->
            <GreaterThan Source="{x:Definition Age}" Value="16" />
            <Or>
                <IsNullOrEmpty Source="{x:Definition Passport}" />
                <Not>
                <!--поддержка Regex условий-->
                <Matches Source="{x:Definition Passport}"
                        Expression="[0-9]{4} [0-9]{6}"
                        RegexOptions="IgnoreCase,CultureInvariant" />
                </Not>
            </Or>
        </And>
        <If.True>
            <WithMessage Text="{x:Resource msg_Pass}" />
        </If.True>
    </If>
</Must>
```

Условие «MatchesForEachLine»

Многострочная проверка регулярным выражением. Пример:

```
<ForMember Definition="{x:Definition String_Field3}">
    <If>
        <Or>
            <IsNullOrEmpty />
            <MatchesForEachLine Trim="True" Expression="^[0-9]+$" />
        </Or>
        <If.False>
```

```
            <WithMessage Text="failure" />
        </If.False>
    </If>
</ForMember>
```

Пример валидации мета-тегов:

```
<ForMember Definition="{x:Definition String_Field}">
    <If>
    <!--установка свойства Condition (оно указано как ContentProperty)-->
        <Or>
            <IsNullOrEmpty />
            <MatchesForEachLine Expression="^<meta[ ]+((name[ ]*=[ ]*"([\w\-])+"[
]+)|(content[ ]*=[ ]*"([\w\-;\.,=\\/? ])+"[ ]+)|(http-equiv[ ]*=[ ]*"([\w\-])+"[
]+))+/>$"
                                RegexOptions="IgnoreCase,CultureInvariant"
                                Trim="True" />
        </Or>
        <If.False>
            <WithLocalizedMessage Text="Введенный текст не является мета-тегами">
                <x:String x:Key="ru-ru">Введенный текст не является мета-
тегами</x:String>
                <x:String x:Key="en-us">Input is not a valid meta.</x:String>
            </WithLocalizedMessage>
        </If.False>
    </If>
</ForMember>
```

Условие «SatisfyExpression»

Условие, в котором можно указать лямбда-выражение в формате Dynamic LINQ.

Пример проверки даты:

```
<ForMember Definition="{x:Definition Date}">
  <If>
    <SatisfyExpression>Convert.ToDateTime(Value) >= DateTime.Now</SatisfyExpression>
      <If.False>
        <!--действие для сообщения об ошибке валидации-->
        <WithMessage Text="дата должна быть не позднее текущей" />
      </If.False>
  </If>
</ForMember>
```

Пример проверки, что поле имеет нечётное значение и больше 10:

```
<ForMember Definition="{x:Definition Age}">
    <If>
        <SatisfyExpression>
            <![CDATA[
            Int32(Value) % 2 == 1 && Int32(Model["Age"]) > 10
            ]]>
```

```
        </SatisfyExpression>
        <If.False>
            <!--действие для сообщения об ошибке валидации-->
            <WithMessage Text="должно быть нечетное и больше 10" />
        </If.False>
    </If>
</ForMember>
```

Условие «IsEmail»

Проверяет, является ли параметр адресом электронной почты. Проверяемое значение должно соответствовать регулярному выражению:

```
"^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?$"
```

Условие «IsJson»

Проверяет, что значение может быть без ошибок десериализовано в формат JSON.

Условие «IsAbsoluteUrl»

Проверяет, является ли значение параметра абсолютным адресом. Проверяемое значение должно соответствовать регулярному выражению:

`"^htt(p|ps):///(([a-z|A-Z|0-9|-|_]+).+)\.[a-z]{1,5}(|\?.+)$"`.

Действия

Название	Описание
WithMessage	Данные по ошибке валидации
WithLocalizedMessage	Данные по ошибке валидации с возможностью локализации сообщения об ошибке
WithValueFormattedMessage	Данные по ошибке валидации с возможностями локализации и вывода значения поля. Есть поддержка `{0}`.

Пример:

```
<WithLocalizedMessage Text="Превышена максимально допустимая длина (700 символов)">
<x:String x:Key="en-us">The maximum length is exceeded (700 characters).</x:String>
</WithLocalizedMessage>
```

4.15.1. Remote-валидация

В QP существует возможность использовать remote-валидацию, расширяющую возможности обычной валидации. В Remote-валидации выполняются дополнительные запросы к базе данных. Это позволяет валидировать соответствие статей и их связей определенной схеме данных.

Введённые пользователем данные передаются на проверку стороннему приложению в формате JSON.

Адрес стороннего приложения, предназначенного для выполнения валидации, рекомендуется задать на уровне сайта в словаре динамических ресурсов (параметр «Словари для валидации XAML»). В этом же параметре следует задать сопоставление ключей и адресов, используемых валидаторов:

```
<DynamicResourceDictionaryContainer xmlns="http://artq.com/validation"
      xmlns:sys="clr-namespace:System;assembly=mscorlib"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
      <DynamicResourceDictionary Name="Urls">
      <x:Uri x:Key="remote_base" x:Name="remote_base">URL приложения</x:Uri>
      <x:Uri x:Key="marketing_products">
          <x:Arguments>
              <x:Reference>remote_base</x:Reference>
              <x:String>MarketingProductValidator</x:String>
          </x:Arguments>
      </x:Uri>
```

Пример использования remote-валидации (валидатор указывается названием словаря и ключа):

```
<ProcessRemoteValidationIf Url="{x:DynamicResource Name=Urls, Key=products}"
HttpMethod="POST" SiteId="35" CustomerCode="backend_name" Timeout="30000" >
    <ProcessRemoteValidationIf.Condition>
        <Not>
            <IsNullOrEmpty Source="{x:Definition MarketingProduct}"/>
        </Not>
    </ProcessRemoteValidationIf.Condition>
    <ProcessRemoteValidationIf.DefinitionsToSend>
        <x:Definition Key="MarketingProduct" />
        <x:Definition Key="Type" />
        <x:Definition Key="Regions" />
        <x:Definition Key="Modifiers" />
        <x:Definition Key="Id" />
        <x:Definition Key="Parameters" />
    </ProcessRemoteValidationIf.DefinitionsToSend>
</ProcessRemoteValidationIf>
```

4.15.2. Поддержка ссылок {x:Reference}

`x:Reference` ссылки позволяют использовать значения элементов, которые уже присутствует в XAML-разметке.

Для того чтобы использовать значение элемента, необходимо:

1) задать имя элемента атрибутом `x:Name`;
2) Для того чтобы получить значение элемента:
    a. указать значение `x:Reference  x:Name` для атрибута тега, в котором должно выводится запрашиваемое значение. Значение `x:Name` должно совпадать со значением пункта 1 списка выше;

b. указать значение `x:Name` в парном теге `x:Reference`, в котором должно выводится запрашиваемое значение. Значение `x:Key` должно совпадать со значением пункта 2 списка выше.

Пример задания имени элемента, значение которого можно использовать в разметке:

```
<If x:Name="condition"> … </If>
```

Пример вызова:

```
<ForMember Definition="{x:Definition strings}" Condition="{x:Reference condition}">
```

4.15.3. Ресурсный словарь

Ресурсный словарь – словарь, содержащий:

- сообщения, которые выводятся при валидации;
- строковые константы или части валидаторов, которые используются повторно в различных правилах.

```
<XamlValidator.Resources>
    <!--константы-->
    <sys:String x:Key="msg_Null">Не может быть null</sys:String>
    <sys:String x:Key="msg_Empty">Не может быть null</sys:String>
    <sys:String x:Key="msg_Pass">Лица старше 16 лет обязаны указывать
паспорт.</sys:String>

    <!--общие для нескольких правил условия-->
    <Sequence x:Key="expr" StopOnFirstFailure="True">
        <!--поддержка последовательных условий с различными сообщениями-->
        <If>
        <IsNull />
        <If.True>
            <WithMessage Text="Не может быть null" />
        </If.True>
        </If>
        <If>
            <IsNullOrEmpty />
            <If.True>
                <WithMessage Text="Не может быть пустой" />
            </If.True>
        </If>
        <If>
            <Not>
                <Length MinLength="5"/>
            </Not>
```

```
            <If.True>
                <WithMessage Text="Длина поля должна быть не менее 5 символов." />
            </If.True>
        </If>
    </Sequence>
</XamlValidator.Resources>

<!--к полю Name применяем условия, указанные в ресурсном словаре-->
<ForMember Definition="{x:Definition Name}" Condition="{x:Resource expr}" />

<!--к полю DuplicateName применяем те же условия, что и к Name-->
<ForMember Definition="{x:Definition DuplicateName}" Condition="{x:Resource expr}"/>

<Must>
    <If>
        <And>
            <!--поддержка логических операндов для условий-->
            <GreaterThan Source="{x:Definition Age}" Value="16" />
            <Or>
                <IsNullOrEmpty Source="{x:Definition Passport}" />
                <Not>
                <!--поддержка Regex условий-->
                <Matches Source="{x:Definition Passport}"
                        Expression="[0-9]{4} [0-9]{6}"
                        RegexOptions="IgnoreCase,CultureInvariant" />
                </Not>
            </Or>
        </And>
        <If.True>
            <WithMessage Text="{x:Resource msg_Pass}" />
        </If.True>
    </If>
</Must>
```

Динамический ресурсный словарь

Динамически ресурсный словарь отличается от ресурсного словаря тем, что его определение задается вне разметки валидатора, и может быть использован в других валидаторах.

Пример создания динамического ресурсного словаря:

```
<DynamicResourceDictionaryContainer
```

```
            xmlns="http://artq.com/validation"
            xmlns:sys="clr-namespace: System;assembly=mscorlib"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
      <!--Словарь Messages валидационные сообщения-->
      <DynamicResourceDictionary Name="Messages">
          …
          <WithMessage x:Key="msg_Required">Не может быть null</WithMessage>
          …
      </DynamicResourceDictionary>
      <!--Словарь Conditions условия-->
      <DynamicResourceDictionary Name="Conditions">
          …


          …
      </DynamicResourceDictionary>
</DynamicResourceDictionaryContainer>
```

Пример использования динамического ресурсного словаря:

```
<XamlObjectValidator xmlns="http://artq.com/validation"
                     xmlns:model="clr-namespace:
QA.Validation.Xaml.Tests.Model;assembly=QA.Validation.Xaml.Tests"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:sys="clr-namespace: System;assembly=mscorlib"
                    Type="{x:Type model:Person}">
      <!--описание полей-->
      <XamlObjectValidator.Definitions>
          …
      </XamlObjectValidator.Definitions>
      <ForMember Definition="{x:Definition Name}">
          <If True="{x:DynamicResource Name=Messages,Key=msg_Required}">
          <!--установка свойства Condition (оно указано как ContentProperty)-->
              <IsNull />
          </If>
      </ForMember>
      …
</XamlObjectValidator>
```

4.15.4. Использование в QP

Создание нового валидатора для статей контента

Валидатор статей контента задается в группе настроек «Custom Validation» свойств контента. Группа настроек содержит следующие элементы:

- Флаг «Создать валидацию XAML по умолчанию» (Create Default XAML Validation). Если флаг установлен, то XAML-разметка валидатора статей контента будет сгенерирована по его структуре. По умолчанию флаг не установлен;
- Текстовое поле «XAML Валидация» (XAML Validation). В поле задается XAML-разметка валидации статей контента. По умолчанию поле не содержит значения и свернуто;
- Флаг «Запретить XAML валидацию» (Disable XAML Validation). Если флаг установлен, то статьи контента не будут валидироваться. По умолчанию флаг не установлен.

Для генерации разметки по умолчанию необходимо установить флаг «Создать валидацию XAML по умолчанию» и сохранить изменения, нажатием по кнопке «Сохранить». После сохранения в поле «XAML Валидация» будет содержаться разметка.

Для задания валидации необходимо внести XAML-разметку в поле «XAML Валидация» и сохранить изменения, нажатием по кнопке «Сохранить». После сохранения в поле «XAML Валидация» будет содержаться разметка.

Создание нового ресурсного словаря

Ресурсный словарь задается в группе настроек «Custom Validation» свойств сайта. Группа настроек содержит следующие элементы:

- Флаг «Создать словарь для валидации XAML по умолчанию» (Create Default XAML Dictionary). Если установить флаг и сохранить настройки сайта, то будет создан пустой словарь. По умолчанию флаг не установлен;
- Текстовое поле «Словари для валидации XAML» (Dictionaries for XAML Validation). В поле задается XAML-разметка ресурсного словаря. По умолчанию поле не содержит значения и свернуто.

4.15.5. Порядок разработки валидатора

1. Установить пакет `QA.Validation.Xaml.Extensions` из NuGet (см. Установка NuGet-пакетов).
2. Создать класс валидатора, реализующий интерфейс `IRemoteValidator2` из пространства имен `QA.Validation.Xaml.Extensions.Rules` (например, `ProductDefinitionValidator`).
3. В реализуемом методе `Validate` доступны экземпляры классов `RemoteValidationContext` и `RemoteValidationResult`. Из контекста можно получить данные и метаданные для валидации, в объект-результат следует выполнять запись ошибок и новых значений (если требуется):
    1) `context.Definitions` – список `Definition` полей, переданных в метод валидации;
    2) `context.ProvideValueExact<T>` – метод для получения конкретного переданного значения по `Definition` поля;
    3) `context.SetValue` – метод для записи значения в объект-результат по `Definition` поля;
    4) `result.Messages` – коллекция сообщений об ошибках в объекте-результате.
4. Необходимо реализовать контроллер, сопоставляющий URL и конкретный валидатор и возвращающий результат валидации в виде JSON.

Пример простейшего контроллера:

```
public class TestValidationController : Controller
    {
        private readonly Func<string, IRemoteValidator2> _validationFactory;

        public TestValidationController(Func<string, IRemoteValidator2>
validationFactory)
        {
            _validationFactory = validationFactory;
        }

        public ActionResult Validate(string validatorKey, RemoteValidationContext
context)
        {
            var result = new RemoteValidationResult();
            try
            {
                result = _validationFactory(validatorKey).Validate(context, result);
            }
            catch (Exception ex)
            {
                result.Messages.Add(ex.Message);
            }
            return Json(result, JsonRequestBehavior.AllowGet);
        }
    }
```

и роута для него:

```
routes.MapRoute(
          name: "TestValidation",
          url: "RemoteValidation/{validatorKey}",
          defaults: new { controller = "TestValidation", action = "Validate" }
                );
```

4.16. Шаблон, страница, объект, формат

Внимание: объекты ветви «Шаблон» являются устаревшими (не рекомендуется использовать для разработки новых Систем; используются для сопровождения Систем, основанных на предыдущих версиях продукта).

Добавление нового шаблона возможно с помощью:

- ГПИ бэкэнда (сущность «Сайт» → сущность «Шаблон» → список шаблонов → команда «Добавить новый шаблон»);

- Команды «Новый шаблон» контекстного меню, вызванного для сущности «Шаблон».

Контекстное меню, вызванное для определенного шаблона, содержит команды:

- «Удалить». Команда удаляет шаблон после ее подтверждения;
- «Свойства». Команда открывает вкладку со свойствами шаблона.

Ветвь иерархии объектов QP служит для организации кода для веб-сайта с использованием бекэнда.

Название	Описание
Шаблон (Template)	Предназначен для объединения страниц, имеющих одинаковое или похожее оформление и для хранения общих настроек страниц и объектов, которые он содержит. Также имеет собственный код, который выполняется в самом начале выполнения страницы. Обычно содержит вызовы одного или нескольких объектов шаблона.
Страница (Page)	Веб-страница (ASP или ASPX), которая будет создана в результате процесса сборки.
Объект (Object)	Минимальный элемент, из набора которых собираются страницы. В сборке ASP.NET каждый объект собирается в пользовательский элемент управления (ASCX). С точки зрения сборки ASP.NET шаблон также является обычным элементом управления (ASCX), при этом он вызывается первым. Отличие страниц друг от друга определяется набором объектов, которые они используют. Объекты могут быть объектами страницы или шаблона. Разница между ними в области видимости: объекты страницы можно вызвать только на данной странице, объекты шаблона – на любой странице данного шаблона, а также с помощью специального формата вызова из других шаблонов. Объект страницы может переопределить объект шаблона. Это означает, что у них будут одинаковые имена и при вызове объекта по имени будет вызван объект страницы, а не объект шаблона.
Формат (Format)	Каждый объект может иметь несколько модификаций (форматов), которые вызываются пользовательским кодом в зависимости от условий. Каждый объект должен иметь минимум один формат. Основная часть пользовательского кода содержится именно в форматах (и небольшая часть в шаблонах). Формат, созданный первым, становится форматом по умолчанию.

Концепция разделения кода аналогична ASP.NET. В шаблоне и формате допускается задание:

Название	Описание
Presentation	Содержит разметку
Code Behind	Содержит логику отображения

Объекты можно вызывать как из Presentation, так и из Code Behind, но синтаксис вызовов будет различаться.

В ASP-сборке разделение кода отсутствует.

4.16.1. Шаблон

Шаблоны и объекты, которые они содержат, определяют внешний вид и разметку страниц сайта. Управление шаблонами осуществляется в бекэнде в разделе «Шаблоны» (Templates) сайта.

В ГПИ бэкенда доступны следующие функции управления шаблонами:

1) создание нового шаблона,
2) сборка шаблона,

3) изменение свойств шаблона,
4) удаление шаблона.

Создание шаблона

Создание шаблона выполняется одним из следующих способов:

1) выбрать раздел «Шаблоны», нажать «Добавить новый шаблон» (New Template);
2) в контекстном меню раздела «Шаблоны» выбрать пункт «Новый шаблон» (New Template).

Свойства шаблона

Собственные свойства

Название	Описание
Имя (Name)	Алфавитно-цифровое имя шаблона. Используется в ГПИ. Название должно быть уникальным в пределах сайта. Также оно не должно совпадать с названием какого-либо объекта этого шаблона, чтобы обеспечить корректность вызовов.
Имя .NET-класса (.NET Class Name)	Используется для создания .NET-класса. Если значение поля «Название шаблона» содержит только английские буквы, цифры и пробелы, то значение данного свойства будет сгенерировано автоматически. Для ASP-сборки поле отсутствует.
Описание (Description)	Произвольное описание шаблона.
Язык .NET (.NET Language)	Выбор языка для разработки Code Behind. Доступны: 1) C# (используется по умолчанию), 2) VB.NET. Для ASP-сборки поле отсутствует. **Примечание:** объекты шаблона наследуют значение этого поля.
Имя папки (Folder name)	Выбор отдельной директории для данных шаблона. Путь задаётся относительно корневой директории веб-сайта. Если директория не существует, она создается.
Максимальное число хранимых версий форматов (Max Number of Format Stored Versions)	Количество версий для форматов шаблона.
Посылать заголовок No-Cache (Send No-Cache header)	Указатель, требуется ли передавать значение `no-cache` в HTTP-заголовках.
Формат (Format)	Используется для генерации пользовательского элемента управления шаблоном, который будет загружаться первым на странице. Содержит поля ввода кода для: 1) Presentation, 2) Code Behind.

Групповые свойства

Относятся в большей степени к объектам QP, для которых шаблон является контейнером (страницы, объекты).

Свойство ViewState

Название	Описание

Разрешить ViewState (Enable ViewState)	Указатель, требуется ли использовать механизм ViewState для элемента управления шаблоном. По умолчанию опция включена по всей иерархии.
Применить к существующим страницам и объектам (Apply To Existing Pages and Objects)	Опция используется, чтобы изменить настройки по всей иерархии. Отключить ViewState также возможно на уровне страниц и объектов. **Примечание:** отключение ViewState на верхнем уровне иерархии блокирует все нижние уровни независимо от их собственных настроек.

Связывание с данными (Data Binding)

Название	Описание
Отключить автоматическое связывание с данными (Disable Auto Data Binding)	По умолчанию опция выключена для всей иерархии. Изменение опции отключает вызов метода DataBind для элемента управления шаблоном.
Применить к существующим объектам (Apply To Existing Objects)	Опция используется, чтобы изменить настройки по всей иерархии. Отключить связывание с данными также можно на уровне объектов.

Наследование

«Наследование страниц» (Pages Inheritance) и «Наследование контролов» (Controls Inheritance) – набор свойств, обеспечивающий расширение функциональных возможностей страниц и элементов управления QP. Предполагается, что разработчик напишет свою .NET-сборку (которую затем поместит в директорию `bin` веб-сайта), а в ней он унаследует свои классы от базовых классов QP:

- «Страница» – от класса `QPage`,
- «Шаблон» – от класса `QUserControl`,
- Объект типа «Generic» – от класса `QUserControl`,
- Объект типа «Publishing Container» – от класса `QPublishControl`.

Для того, чтобы в дальнейшем использовать новые классы, при сборке необходимо указать имена этих классов. Настройки наследования уровня шаблона могут быть переопределены на уровне страницы или уровне объекта.

Кроме того, при использовании собственных классов обычно бывает необходимо добавить свои пространства имён, что может быть достигнуто с помощью опции «Дополнительные пространства имён» (Additional Namespaces).

Стоит отметить, что необязательно наследовать все классы: для большинства задач хватает только нового класса для страниц.

Региональные и языковые настройки (Regional and Language Settings)

Название	Описание
Набор символов (Charset)	Выбор набора допустимых символов, которые могут использоваться на страницах шаблона. Влияет на процесс сборки: в БД код и данные контентов хранятся в UTF-8, а при генерации файлов используется выбранный пользователем набор символов. Рекомендуется использовать UTF-8.
Локализация (Locale)	Выбор правил форматирования для чисел, дат, денежных величин.

Сборка шаблона

Сборка осуществляется из свойств шаблона и его объектов:

- если шаблон пустой (нет страниц), то собираются все объекты шаблона;

- если в шаблоне есть страницы, то проводится сборка страниц шаблона (по алгоритму сборки нескольких страниц).

Также выполняется сборка уведомлений шаблона.

Для сборки шаблона в списке шаблонов следует выбрать требуемый шаблон, нажать «Собрать».

Изменение свойств шаблона

Изменение свойств шаблона выполняется одним из следующих способов:

1) в контекстном меню шаблона выбрать пункт «Свойства» (Properties),
2) в списке шаблонов выбрать требуемый шаблон и нажать кнопку «Свойства» (Properties).

Удаление шаблона

Удаление шаблона выполняется одним из следующих способов:

1) в контекстном меню шаблона выбрать пункт «Удалить» (Remove),
2) в списке шаблонов выбрать требуемый шаблон и нажать кнопку «Удалить» (Remove).

4.16.2. Страница

Свойства страницы

Общие свойства (Basic parameters)

Название	Описание
Имя (Name)	Имя страницы.
Имя файла (File Name)	Имя файла со страницей.
Описание (Description)	Описание страницы.
Пользовательский класс (Custom class)	Имя пользовательского класса.
Имя папки (Folder Name)	Позволяет задать отдельную директорию для данных страницы. Путь задаётся относительно корневой директории веб-сайта. Если директория не существует, она создается.

Свойства ViewState

Название	Описание
Разрешить ViewState (Enable ViewState)	Указатель, требуется ли использовать механизм ViewState для страницы.
Применить к существующим объектам (Apply to existing objects)	Опция используется, чтобы изменить настройки по всей иерархии. Отключить ViewState также возможно на уровне страниц и объектов. **Примечание:** отключение ViewState на верхнем уровне иерархии блокирует все нижние уровни независимо от их собственных настроек.

Кэширование

Название	Описание
Посылать заголовок No-Cache (Send No-Cache header)	Указатель, требуется ли передавать значение `no-cache` в HTTP-заголовках.
Посылать заголовок Last-Modified (Send Last-Modified header)	Указатель, требуется ли передавать значение `Last-Modified` в HTTP-заголовках.
Кэширование на proxy (Proxy caching)	Указатель, что следует выполнять кэширование на прокси-сервере.
Кэширование в браузере (Browser Caching)	Указатель, что следует выполнять кэширование в браузере.
Истекает через (ч.)	Срок годности данных в кэше.

Региональные и языковые настройки (Regional and Language Settings)

Название	Описание

Набор символов (Charset)	Выбор набора допустимых символов, которые могут использоваться на страницах шаблона. Влияет на процесс сборки: в БД код и данные контентов хранятся в UTF-8, а при генерации файлов используется выбранный пользователем набор символов. Рекомендуется использовать UTF-8.
Локализация (Locale)	Выбор правил форматирования для чисел, дат, денежных величин.

Сборка страницы

Сборка требуемого набора страниц осуществляется кнопкой «Собрать» (Assemble).

Копирование страницы

Копирование страницы выполняется одним из следующих способов:

1) в контекстном меню страницы выбрать пункт «Создать по образцу» (Create Like),
2) в списке страниц выбрать требуемую страницу и нажать кнопку «Создать по образцу» (Create Like).

Для копии используются следующие значения полей:

Название	Описание
Название страницы	Формат имени – `Copy of` *Название Исходной Страницы*)
Имя файла	Формат имени – `copy_of_`*имя Файла Исходной Страницы*)

4.16.3. Объект

Объекты делятся на следующие группы:

1) объекты шаблона,
2) объекты страницы.

Внутреннее отличие заключается только в области видимости, которая проявляется при вызове объекта. В ГПИ управление объектами осуществляется на уровне страницы.

Используются следующие типы объектов:

Название	Описание
Generic	Базовый объект, используется в большинстве случаев.
Publishing Container	Объект для вывода данных из контента.
Publishing Form	Объект для ввода данных в контент.
Style Sheet(CSS)	Любой из этих типов фактически является типом «Generic» и носит только маркерную функцию. Исключение составляют объекты типов «CSS» и «JavaScript»: у них имеется дополнительная опция «Глобальный» (Global).
JavaScript	
Macromedia Flash	
Windows Media	
Meta Keywords	
Meta Description	

Свойства объекта

Общие

Название	Описание
Переопределить родительский объект (This object overrides parent object)	Указатель, что новый объект должен быть создан путём переопределения существующего объекта шаблона. Значения параметров «Название объекта» и «Название объекта .NET» наследуются от родительского объекта. Допускается задание типа объекта, отличного от типа родительского объекта.

Название объекта (Object Name)	Идентификатор объекта. Название объекта должно быть уникально в пределах объектов шаблона или объектов страницы. Также оно не может совпадать с названием шаблона. Название объекта используется в коде для вызова этого объекта.
Название объекта .NET (.NET Object Name)	Название объекта. Используется для создания .NET-класса. Если значение поля «Название объекта» содержит только английские буквы, цифры и пробелы, то значение данного свойства будет сгенерировано автоматически. Для ASP-сборки поле отсутствует.
Описание (Description)	Описание объекта.
Глобальный (Global)	Опция доступна только для объектов типов «JavaScript» и «CSS» уровня шаблона.
Тип (Type)	Тип объекта.
Разрешить OnScreen (Allow OnScreen)	**Внимание:** Режим OnScreen существует только в предыдущих версиях продукта. Указатель, что должна быть возможность работать в режиме OnScreen. **Примечание:** Режим OnScreen для объектов поддерживается только ASP-сборкой в целях совместимости. **Примечание:** значение свойства для объекта «Publishing Container» не влияет на работу режима для полей внутри этого объекта.
Разрешить ViewState (Enable ViewState)	Управляет возможностью использования ViewState для объекта. По умолчанию выключена. Аналогичная опция также доступна на уровне шаблона и страницы. Отключение ViewState на верхнем уровне иерархии блокирует все нижние уровни независимо от их собственных настроек.
Отключить автоматическое связывание с данными (Disable auto data binding)	Управляет возможностью вызова метода `DataBind` для элементов управления, генерируемых из данного объекта. Отключение связывания с данными на уровне шаблона распространяется только на элемент управления шаблоном и никак не влияет на объекты, если только не использовать опцию «Применить к существующим объектам» (Apply To Existing Objects). По умолчанию выключена.
Пользовательский класс (Custom Class)	Позволяет задать класс, от которого будет унаследован объект. По умолчанию значение берётся с уровня шаблона. Отсутствие данных означает, что будет использован класс по умолчанию в зависимости от типа объекта и мобильных настроек шаблона. Позволяет для текущего объекта перекрыть пользовательский класс, заданный на уровне шаблона.
Использовать значения по умолчанию (Use Default Values)	Установка опции позволяет задать значения по умолчанию (Default Values).

Свойства объекта типа «Publishing Container»

Выбор источника

Название	Описание

Контент (Контент)	Контент, из которого будут выводиться данные. В случае динамического изменения контента опция представляет контент по умолчанию.
Разрешить динамическое изменение контента (Allow Dynamic Content Change)	Флаг, отвечающий за использование режима динамического изменения контента.
Переменная, содержащая название контента (Dynamic Content Variable)	Название параметра из коллекции `Values`, который содержит имя контента. Поддерживается вызов контентов из различных сайтов (с использованием имени контента в формате *SiteName*.*ContentName*). В целях защиты от SQL-инъекции существует ограничение: параметр с данным именем не должен существовать в коллекциях `HttpRequest.QueryString` и `HttpRequest.Form`, он должен быть добавлен с помощью метода `AddValue`. Крайне не рекомендуется подставлять в эту переменную данные, полученные от пользователя, желательна проверка данных по принципу белого списка. В случае, если переданное значение не соответствует действительному имени контента, будут выведены данные из контента по умолчанию (задаётся в поле «Контент»). Во избежание ошибок рекомендуется, чтобы структура контентов, между которыми осуществляется динамическое переключение, была одинаковой.

Фильтрация

Название	Описание
Фильтр (Filter)	Позволяет настроить выборку из статей контента согласно заданному критерию. Можно использовать динамический фильтр. Фильтр представляет собой часть предложения `WHERE` SQL-запроса, другая часть предложения `WHERE` формируется на основании прочих полей и служебных данных. При динамическом формировании фильтра обычно используется механизм `Values`. **Примечание:** рекомендуется создавать индекс для поля, по которому осуществляется фильтрация.
Использовать расписание статей (Use Schedule)	По умолчанию включено. Сброс флага позволяет игнорировать поле `Visible` (используется инструментом «Расписание»), в этом случае статья будет отображаться на веб-сайте вне зависимости от значения указанного поля.
Показать архивированные (Show archived)	Позволяет включить в результат статьи из архива (игнорировать значение поля `Archive`). По умолчанию архивные статьи не попадают в результирующий набор.
Статусы (Statuses)	Статусы Workflow статей, которые требуется выводить на веб-сайте. Опция доступна, когда в качестве источника данных выбран контент, на который назначен Workflow. **Примечание:** опция используется только в Live-режиме (в Stage-режиме на веб-сайт выводятся статьи всех статусов).
Использовать правила доступа бекэнда (Use Security)	Указатель, что требуется вместе со статьями получать данные об уровнях доступа пользователей бекэнда к этим статьям, что даёт возможность применять правила доступа бекэнда. По умолчанию выключена. **Внимание:** требуется предварительная дополнительная настройка интегрированных правил доступа. Доступны следующие режимы работы:

	• «Возвращать только подходящие статьи» (Return only matching articles); В результирующий набор попадут только статьи, для которых текущий пользователь имеет уровень доступа в интервале, заданном в свойствах «Начальный уровень» и «Конечный уровень». • «Возвращать все статьи вместе с уровнями доступа» (Return All articles with specified permission level). Фильтрация по уровням доступа не производится. Использовать в случае, если задача решена в коде Разработчика.
Начальный уровень (Start Level)	Значение минимального уровня доступа.
Конечный уровень (End Level)	Значение максимального уровня доступа.

Настройка интегрированных правил доступа бекэнда

Часто требуется создать закрытый раздел на веб-сайте с возможностью входа для зарегистрированных пользователей. Данная функциональность позволяет использовать для этой цели пользователей QP вместо создания собственного хранилища пользовательских данных на базе контентов QP.

В конфигурационном файле QP нужно определить переменные, через которые будут передаваться идентификаторы пользователя и группы.

Для поддерживаемых языков .NET используются собственные переменные. По умолчанию используются следующие названия:

Язык	Название переменной
C#	• `Session["qp_UID"]` • `Session["qp_GID"]`
VB.NET	• `Session("qp_UID")` • `Session("qp_GID")`

```
<app_var app_var_name="security_UID_varname_VB">Session("qp_UID")</app_var>
<app_var app_var_name="security_GID_varname_VB">Session("qp_GID")</app_var>
<app_var app_var_name="security_UID_varname_CSharp">Session["qp_UID"]</app_var>
<app_var app_var_name="security_GID_varname_CSharp">Session["qp_GID"]</app_var>
```

Для аутентификации пользователя следует использовать метод `AuthenticateUser`. В случае успеха результат выполнения метода будет содержать идентификатор пользователя, который нужно сохранить в переменной.

```
Session["qp_UID"] = Quantumart.QPublishing.Permissions.AuthenticateUser(username,
password);
```

Альтернативный вариант – сохранить в переменной `Session["qp_GID"]` идентификатор группы и выставить `Session["qp_UID"]` в `NULL`. Тогда QP будет использовать правила доступа для группы. Список групп для пользователя можно получить с помощью метода `GetRootGroupsForUser`.

Если посетители веб-сайта имеют право создать нового пользователя, то следует использовать метод `AddUser`.

```
Quantumart.QPublishing.Permissions.AddUser(username, password, First_Name, Last_Name, Email);
```

Для добавления новых пользователей в одну или несколько групп следует использовать метод `AddUserToGroup`:

```
Quantumart.QPublishing.Permissions.AddUserToGroup(int userId, int groupId);
```

Для нахождения нужной группы следует использовать метод `GetGroupInfo`:

```
DataTable dt = GetGroupInfo(group_name); //group_name is a string
int group_id = dt.Rows[0]("group_id");
```

QP позволяет назначать права на статьи пользователям и группам. Это полезно при наличии на веб-сайте содержимого, создаваемого пользователями. В таком случае пользователю можно предоставить возможность самому указать, каким пользователям (группам) дать доступ к собственным данным.

QP также поддерживает вложенные группы, где дочерняя группа наследует права родительской, таким образом косвенно влияя на права пользователей. Существуют методы, позволяющие устанавливать отношения между группами.

В QP8 API имеются методы для работы с режимом интегрированных прав доступа:

- `AddUserToItemPermission`,
- `AddGroupToItemPermission`,
- `GetPermissionLevels`,
- `AddChildGroupToParentGroup`.

Сортировка

Название	Описание
Return Articles	Возможные значения: 1) `Sequentially`, 2) `Randomly`.
Порядок по умолчанию (Default Sorting)	Позволяет задать одно или несколько правил сортировки: 1) указать поле контента, по которому должна выполняться сортировка; 2) задать направление сортировки.
Динамический порядок (Use Dynamic Sorting)	Переключатель между динамической и статической сортировкой. По умолчанию выключен, что соответствует статической сортировке. Статическая сортировка задается набором элементов управления «Порядок по умолчанию». При установленном флаге объект «Publishing Container» будет использовать «Выражение динамической сортировки», которое обычно передается через механизм `Values`. По умолчанию выполняется сортировка по возрастанию значения служебного свойства `CONTENT_ITEM_ID`. **Примечание:** для поля, по которому осуществляется сортировка, рекомендуется создавать индекс.
Выражение динамической сортировки (Dynamic Sorting Expression)	Правило для динамической сортировки.

Постраничный вывод

Название	Описание
Selection is starting	Возможные значения: 1) «From the first article» (по умолчанию), 2) «From the article with the specified number». **Примечание:** если значение должно быть динамическим, то следует передавать его через механизм `Values` и использовать метод `NumValue`.
Selection includes	Возможные значения: 1) «All articles» (по умолчанию), 2) «Specified Number Of Articles». **Примечание:** значение может быть динамическим (передавать с использованием механизма `Values`). Следует использовать метод `NumValue`. В сочетании с динамической опцией «Номер первой отображаемой записи» (First article number) реализует постраничный вывод.

Кэширование

Название	Описание
Разрешить кэширование данных (Enable Data Caching)	Разрешает кэширование набора данных, возвращаемых объектом «Publishing Container». Ключ кэширования формируется на основе идентификатора формата и предложения `WHERE` SQL-запроса. Таким образом, для одного объекта «Publishing Container» может существовать несколько записей в кэше. Интервал кэширования определяется значением опции «Длительность (мин)» (Duration (min)). Устаревание кэша – абсолютное (по истечении интервала кэширования запись удаляется из кэша вне зависимости от частоты обращений к ней).
Возвращать дату последней модификации (Return Last Modified)	Выводит самую последнюю дату изменения из дат в наборе возвращаемых статей. Используется для формирования HTTP-заголовка `Last-Modified` и настраивается на уровне страницы.

Свойства объекта типа «Publishing Form»

Название	Описание
Контент (Content)	Контент, в который должны будут добавляться данные.
Генерировать код для создания/удаления статей (Generate code for updating/appending articles)	По умолчанию опция отключена. В этом случае предполагается, что Разработчик сам напишет требуемый код.
Перейти после заполнения формы на страницу (Submission response page)	Выбор страницы, на которую будет осуществлён автоматический переход после успешной отправки формы на сервер.
Язык .NET (.NET Language)	Язык, который требуется использоваться при автоматической генерации формата.

При создании объекта типа «Publishing Form» среди объектов текущего шаблона генерируется объект `_Check_Content_Article_Form_Script` типа JavaScript, в который вставляется универсальный скрипт проверки для объектов типа «Publishing Form». Операция используется только для первого объекта «Publishing Form», созданного в пределах шаблона.

Сборка объекта

Сборка требуемого набора объектов осуществляется кнопкой «Собрать» (Assemble).

Вызовы объектов

Объекты могут быть вызваны из страницы или других объектов. В ASP объект представляет собой функцию, в ASP.NET – пользовательский элемент управления.

Способы вызова объектов различаются в зависимости от типа сборки и места вызова:

- ASP.NET Presentation:

```
<qp:placeholder calls="CSS" runat="server"/>
```

- ASP.NET Code Behind:

```
ShowObject("CSS", this);
```

- ASP:

```
<%=Object("CSS")%>
```

Поддерживаются следующие форматы вызова:

Название	Описание
O	*ObjectName*. Наиболее распространённый формат вызова. Сначала ищется объект страницы с таким именем, затем объект шаблона. Если объект не найден, то выдается ошибка. В случае успеха загружается формат по умолчанию.
OF	*ObjectName*.*FormatName*. Аналогичен формату TO, есть возможность явно выбрать загружаемый формат.
TO	*TemplateName*.*ObjectName*. Позволяет вызывать объекты других шаблонов, в случае успеха вызывается формат по умолчанию.
TOF	*TemplateName*.*ObjectName*.*FormatName*. Аналогичен формату TO, есть возможность явно выбрать загружаемый формат.

Поддерживаются рекурсивные вызовы. По умолчанию глубина рекурсии ограничена 32 уровнями.

Примечание: не рекомендуется вызывать объект рекурсивно из Presentation.

Presentation

Вызов объекта из Presentation:

```
<qp:placeholder calls="[template_name.]object_name[.format_name]" runat="server"/>
<qp:placeholder calls="[template_name.]object_name[.format_name]" simple="true" runat="server"/>
```

По умолчанию используется модифицированная последовательность событий QP. Для использования оригинальной последовательности событий следует использовать вызов с параметром `simple="true"`.

Code Behind

Вызовы осуществляются с помощью методов классов `QPage` и `QUserControl`.

Механизм Values (передача данных между объектами)

В QP для передачи данных между объектами можно использовать механизм `Values`. `Values` – это коллекция уровня страницы. Срок жизни коллекции `Values` – текущий запрос (не сохраняется в сессии, не передается между запросами). Данные в коллекцию `Values` попадают из следующих основных источников:

- `HttpRequest.QueryString` (для ASP – `Request.QueryString`),
- `HttpRequest.Form` (для ASP – `Request.Form`),
- Вызов метода AddValue из пользовательского кода.

Идея передачи данных между объектами через коллекцию `Values` хорошо ложится в русло технологии ASP, так как в ней применяется прямое исполнение кода. В случае же ASP.NET из-за событийно-ориентированного подхода есть ряд ограничений использования этого механизма.

Передача данных от родительского объекта к дочернему

Рассмотрим дерево элементов управления некоторой страницы ASP.NET. При статической загрузке элементов управления событие `Init` распространяется в обратном порядке (от листьев к корню), поэтому передача данных в этом случае невозможна. Поэтому QP использует динамическую загрузку вместе с собственной последовательностью событий `Init`. Стоит отметить, что при загрузке объекта через Presentation используется элемент управления `qp:placeholder` и предполагается, что он должен загружаться статически, но фактически он представляет собой оболочку, которая, в свою очередь, динамически загружает нужный элемент управления. Так как сама оболочка загружается статически, то невозможна передача данных из родительского объекта в дочерний, который вызван через `qp:placeholder`. У данной реализации есть побочный эффект: из-за использования собственной последовательности событий Init перестаёт правильно работать назначение элементам управления клиентских идентификаторов, а, следовательно, потом страница не может корректно восстановить ViewState. Это приводит к тому, что при использовании серверных элементов управления внутри элементов управления QP первые могут сохранить свое состояние. Решение этой проблемы – заключение такого элемента управления или группы элементов управления в отдельный объект шаблона (страницы), который должен быть вызван одним из следующих способов:

- с использованием атрибута `simple = true` для `qp:placeholder`:

```
<qp:placeholder calls="News" simple="true" runat="server" />
```

- с помощью метода `ShowObjectSimple`:

```
ShowObjectSimple("News", this);
```

При этом передача данных в такой объект работать не будет.

Передача данных между итерациями Publishing Container

Если для вывода данных объекта «Publishing Container» используется сгенерированный по умолчанию Repeater, то записывать значение для следующей итерации в `Value` лучше всего в обработчике события `OnItemDataBound`, а использовать в логике для формирования текущего вывода – в обработчике события `OnItemCreated`.

Использование Values

При использовании `Values` в качестве составных частей фильтров объекта «Publishing Container» рекомендуется использовать метод `NumValue` или `StrValue`. Первый подходит для числовых параметров: если на входе не число, то он возвращает значение `0`, тем самым гарантируя числовой результат. Второй метод подходит для строковых параметров: он удваивает апострофы (в отличие от метода `Value`, который их удаляет). Рекомендованные способы применения методов в фильтрах:

```
"[content_item_id] = " + NumValue("id")
"[Name] = '" + StrValue("name") + "'"
```

Аккуратная работа с апострофами необходима, чтобы избежать возможности SQL-инъекции. Если нужно получить значение из коллекции `Values` в неизменном виде (без удалений и замен апострофов, преобразования к числу), то следует использовать метод `DirtyValue`. Он может быть полезен при передаче фильтра целиком через коллекцию `Values`. В этом случае при формировании фильтра необходимо удостовериться в невозможности SQL-инъекции. Добавить новое значение можно с помощью метода `AddValue`.

Прямая работа с коллекцией `Values` не рекомендуется.

Механизм переопределения объектов

Переопределение объектов играет ключевую роль при создании сайта на объектах QP. Под переопределением объекта шаблона на уровне страницы понимается создание на уровне страницы объекта с тем же именем. Таким образом, когда будет вызван объект с этим названием (форматы вызова O и OF), то будет загружен объект шаблона, а не страницы. При создании объекта страницы не нужно вводить совпадающее имя, достаточно просто выбрать требуемый родительский объект.

Связь между родительским и дочерним объектом проявляется только на уровне вызовов, никакой другой связи (например, на уровне классов) нет. Единственное ограничение, накладываемое механизмом вызова объектов – имена родительского и дочернего объектов должны совпадать.

Из объекта страницы с помощью форматов вызова TOF и TO можно вызвать объект шаблона с тем же именем и таким образом реализовать сценарий «доопределения» объекта шаблона.

4.16.4. Формат

В списке форматов можно изменить формат по умолчанию. Также формат по умолчанию можно изменить в свойствах объекта. Опция влияет на программный вызов объекта, содержащего данный формат.

Свойства формата

Название	Описание
Название формата (Format Name)	Алфавитно-цифровое имя формата. Является уникальным в пределах объекта. Используется в ГПИ.
Название формата .NET (.NET Format Name)	Имя формата. Используется для создания .NET-класса. Если значение поля «Название формата» содержит только английские буквы, цифры и пробелы, то значение данного свойства будет сгенерировано автоматически. Для ASP-сборки свойство отсутствует.
Описание (Description)	Текст для описания объекта.
Язык .NET (.NET Language)	Выбор языка для разработки Code Behind для элемента управления данного формата. Доступны: 1) C# (используется по умолчанию), 2) VB.NET. Для ASP-сборки свойство отсутствует.
Presentation	Содержимое поля используется для генерации пользовательского элемента управления форматом.
Code Behind	

4.16.5. Поиск по коду для объектов

В ГПИ существует возможность выполнить поиск по коду для объектов QP. ГПИ доступен в контекстном меню для сайта.

Поиск ведётся с помощью `LIKE`-синтаксиса SQL по:

1) Code Behind и Presentation шаблонов (если область поиска это допускает),
2) Code Behind и Presentation форматов,
3) полям объектов «Publishing Container», которые содержат динамические значения:
   - «Фильтр» (Filter),
   - «Номер первой отображаемой записи» (First article number),
   - «Показывать все» (Show All),
   - «Выражение динамической сортировки» (Dynamic order expression),
   - «Значения по умолчанию» (Default Values).

4.17. Прочее

4.17.1. Библиотека сайта

Библиотека сайта позволяет работать с файлами из определённой директории в файловой системе. Библиотека сайта предназначена для размещения файлов, общих для всего сайта или нескольких контентов. Взаимодействие пользователя с Библиотекой осуществляется с использованием ГПИ бекэнда.

Открыть ГПИ библиотеки сайта можно с помощью команды «Библиотека» контекстного меню, вызванного для сайта.

4.17.2. Библиотека контента

По функциональности совпадает с Библиотекой сайта. Открыть ГПИ библиотеки можно с помощью команды «Библиотека» контекстного меню, вызванного для определенного контента.

Особенности:

- содержит только файлы, используемые в одном определённом контенте;
- права доступа на директории Библиотеки наследуются от контента.

4.17.3. Кэширование

Кэширование вывода

Реализуется с помощью директивы `OutputCache` для Presentation на странице свойств формата. В этом случае используется кэширование вывода на уровне пользовательских элементов управления. Необходимо учитывать, что в случае получения данных из кэша Code Behind выполняться не будет, что может быть неприемлемо.

Обычный синтаксис:

```
<%@ OutputCache Duration="100" VaryByParam="location;count" %>
```

Замечания:

- Для объектов шаблона желательно использовать параметр «Shared» (позволяет уменьшить нагрузку на память сервера).
- Крайне не рекомендуется использование `VaryByParam="*"` из-за вероятных проблем с производительностью.

Кэширование данных

Реализуется в зависимости от того, как осуществляется доступ к данным.

Publishing Container

В свойствах контейнера есть настройки кэширования. В качестве ключа кэширования используется полная совокупность параметров хранимой процедуры `qp_GetContentPage`, использующейся для получения данных.

QP Cache API

Доступен в виде методов класса `DBConnector`. Используется для кэширования:

1) результатов SQL-запросов,
2) коллекций объектов LINQ to SQL,
3) произвольных объектов.

4.17.4. Блокировка сущностей

Блокировка экземпляра сущности пользователем бекэнда – это запрет на изменение данного экземпляра другими пользователями. Заблокированный экземпляр может разблокировать:

1) пользователь, выполнивший блокировку;
2) пользователь, входящий в группу пользователей «Администраторы»;
3) пользователь, входящий в группу пользователей с включённой настройкой «Члены группы могут разблокировать сущности».

Блокировка используется для следующих сущностей:

- «Сайт»,
- «Статья»,
- «Шаблон»,
- «Объект»,
- «Формат»,
- «Страница».

Блокировка устанавливается автоматически после входа пользователя на страницу изменения свойств сущности. По окончании редактирования блокировка автоматически снимается. У пользователя имеется возможность указать, что блокировка должна существовать постоянно.

Примечание: постоянная блокировка автоматически используется в следующих случаях:

- работа веб-приложения была завершена в аварийном режиме,
- окно браузера было закрыто.

Для объекта раздельно блокируются:

- общие свойства (для всех типов объектов),
- свойства, специфичные для объекта «Publishing Container».

4.17.5. Механизм автозамены URL

Механизм позволяет избежать обязательных изменений содержимого БД при изменении домена веб-сайта. В БД в значения полей типов «Строка», «Текстовое окно» и «Визуальный редактор» сохраняется не URL, а заполнитель (placeholder), который при выводе на веб-сайт заменяется на значение, автоматически формируемое на основании текущих свойств сайта. Механизм автозамены URL включен по умолчанию. Для отключения такой возможности, необходимо в свойствах сайта снять указатель «Заменять URLS на заполнители (Replace urls to placeholders)».

Правила работы:

Название операции	Исходное значение	Итоговое значение
Сохранение в БД	`http://`*DNS*/*URI загрузки*	Заполнитель `<%=upload_url%>`.
	`http://`*DNS*/*Виртуальный путь в Live*	Заполнитель `<%=site_url%>`.
	`http://`*Stage DNS*/*Виртуальный путь в Stage*	
Вывод данных на веб-сайт	`<%=upload_url%>`	Значение «Виртуальный путь» из группы свойств «Расположение загруженных файлов» сайта. При активном свойстве «Использовать абсолютный виртуальный путь» к результату добавляется значение свойства «Префикс виртуального пути».
	`<%=site_url%>`	Зависит от режима работы веб-сайта. Значение «Виртуальный путь» из группы свойств сайта: • «Расположение страниц в Основном режиме», • «Расположение страниц в Тестовом режиме». Если в конфигурационном файле веб-сайта (`web.config`) задано `UseAbsoluteSiteUrl = 1`, то к результату добавляется: • `http://`DNS • `http://`*Stage DNS*

Обратная замена заполнителей на URI происходит только при:

- вызове методов QP8 API:
  - `Field` (для вывода данных с использованием объекта «Publishing Container»),
  - `FormatField` (замена в произвольной строке);
- использовании классов LINQ to SQL.

4.17.6. Запись и воспроизведение действий

Существует функциональная возможность записать в файл последовательность всех действий пользователей в бекэнде (в пределах текущего Customer Code), приводящих к изменению данных в БД, а также в дальнейшем автоматически выполнить в бекэнде все действия из подобного файла.

Внимание: в текущей версии QP не поддерживается запись следующих действий:

- действия, связанные с изменением и удалением дочерних прав доступа;
- «Пересоздать изображения»;
- «Применить значение по умолчанию»;
- все действия с файлами (предполагается, что копирование файлов будет осуществлено отдельно).

Возможность полезна в ситуациях, когда Система существует в нескольких окружениях, при этом в части окружений возможность вносить изменения прочими штатными средствами QP отсутствует

или занимает много времени, например, из-за корпоративных политик. Запись действий осуществляется в формате XML.

Примечание: данные по воспроизведению действий приведены в руководстве администратора.

Запись действий на сервере-источнике

Для включения записи требуется активировать свойство «Записывать действия в файл» (Record actions into file) (пункт контекстного меню «Настройки» для корневого элемента в дереве сущностей).

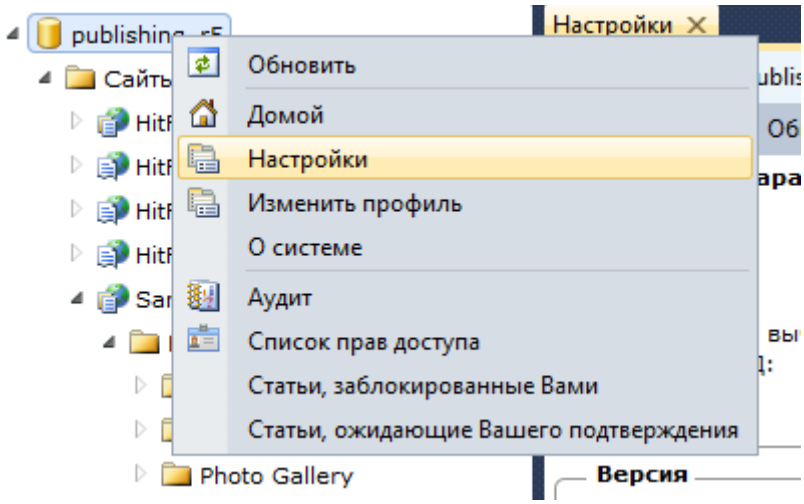

Рисунок 4.17.6-1. Настройки в контекстом меню.

Свойство «Очистить файл, если он существует» (Empty file if exists), следует активировать, если действия нужно записывать в пустой файл. В противном случае запись будет осуществляться в конец существующего файла.

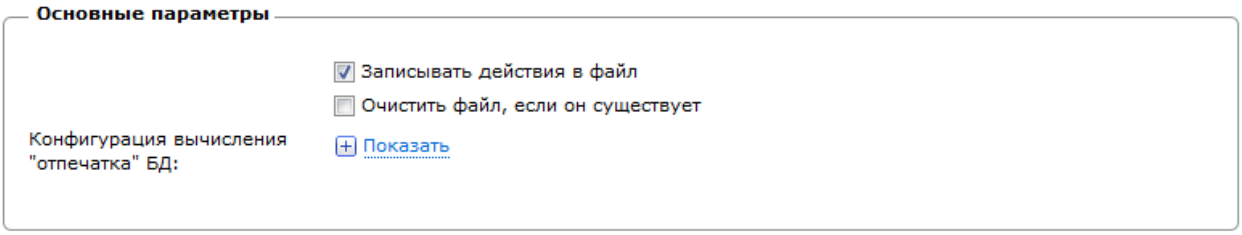

Рисунок 4.17.6-2. Параметры записи действий.

XML-файл размещается во временной директории QP, имя файла совпадает с Customer Code.

В файле каждому действию соответствует элемент `action`, содержимое которого представляет собой набор полей формы вместе с их значениями.

Отпечаток БД (DB fingerprint)

Механизм проверки отпечатка БД (DB fingerprint) предназначен для определения, что БД, в которой выполнена запись XML-файла и БД, в которой требуется выполнить действия, не имеют существенных отличий. Решение об отличии принимается на основании вычисления отпечатка БД.

Если отпечатки двух БД совпадают, то они считаются идентичными или несущественно различающимися, что означает, что действия из XML-файла, записанные в одной БД, можно выполнить во второй.

Конфигурация отпечатка БД задаётся в свойстве «Конфигурация вычисления "отпечатка" БД» (DB fingerprint computing configuration).

Если конфигурация отпечатка БД задана, то при создании нового файла в него будет записано вычисленное значение отпечатка и конфигурация, при которой он вычислялся.

Пример конфигурации отпечатка БД:

```
<fingerprint>
	<entityType code="site">
		<included>
			<id>34</id>
		</included>
	</entityType>
	<entityType code="content" considerIdentity="true">
		<included>
			<parentId>34</parentId>
		</included>
		<excluded>
			<id>285</id>
		</excluded>
	</entityType>
	<entityType code="field">
		<excluded>
			<parentId>285</parentId>
		</excluded>
	</entityType>
	<entityType code="content_link" />
</fingerprint>
```

Параметры:

Название	Описание

entityType	Задаёт сущности QP, по которым должен вычисляться отпечаток.
	Возможные значения кодов:
	`public const string None = "";`
	`public const string CustomerCode = "db";`
	`public const string Site = "site";`
	`public const string ContentGroup = "content_group";`
	`public const string Content = "content";`
	`public const string Article = "article";`
	`public const string OldArticle = "content_item";`
	`public const string ArchiveArticle = "archive_article";`
	`public const string ArticleVersion = "article_version";`
	`public const string FieldGroup = "field_group";`

```
public const string Field = "field";
public const string VirtualField = "virtual_field";
public const string VirtualArticle = "virtual_article";
public const string Notification = "notification";
public const string VisualEditorPlugin = "visual_editor_plugin";
public const string VisualEditorStyle = "visual_editor_style";
public const string VisualEditorCommand = "visual_editor_command";
public const string ContentFolder = "content_folder";
public const string SiteFolder = "site_folder";
public const string OldSiteFolder = "folder";
public const string ContentFile = "content_file";
public const string SiteFile = "site_file";
public const string Workflow = "workflow";
public const string WorkflowRule = "workflow_rule";
public const string VirtualContent = "virtual_content";
public const string StatusType = "status_type";
public const string Style = "style";
public const string Snippet = "snippet";
public const string User = "user";
public const string UserGroup = "user_group";
public const string CustomAction = "custom_action";
public const string BackendAction = "backend_action";
public const string SitePermission = "site_permission";
public const string ContentPermission = "content_permission";
public const string ArticlePermission = "article_permission";
public const string WorkflowPermission = "workflow_permission";
public const string SiteFolderPermission = "site_folder_permission";
public const string EntityTypePermission = "entity_type_permission";
public const string ActionPermission = "action_permission";
public const string PageTemplate = "template";
public const string Page = "page";
public const string TemplateObject = "template_object";
public const string PageObject = "page_object";
public const string TemplateObjectFormat = "template_object_format";
public const string PageObjectFormat = "page_object_format";
public const string PageObjectFormatVersion =
"page_object_format_version";
```

	`public const string TemplateObjectFormatVersion = "template_object_format_version";` `public const string ContentLink = "content_link";`
Included Excluded ParentId Id	Можно ограничить вычисление отпечатка по определённому критерию, включив или исключив определённые сущности по их собственному или родительскому идентификаторам. Ограничения учитывают иерархию сущностей (к дочерним сущностям ограничение будет применено автоматически). Например, если нужно ограничить вычисление определенным сайтом (`Id = 34`), то требуется задать это ограничение только для `entityType` со значением `site`.
considerIdentity	Указатель, требуется ли при вычислении отпечатка учитывать текущее значение счётчика для идентификатора сущности. Например, если создать поле и затем сразу же удалить его, то содержимое БД до операции будет идентично содержимому БД после операции, но значение счётчика увеличится.

5. Создание Систем с использованием продукта

Возможные архитектурные решения для Системы:

Название	Описание
Использование Виджетной платформы	Рекомендуемое решение (см. Использование Виджетной платформы)
Использование исключительно объектов QP (объекты ветви «Шаблон»)	Подход устарел, не рекомендуется для разработки новых Систем (см. Использование объектов QP)
Использование собственной архитектуры	-

Способы разработки кода:

Название	Описание
QP8 API	Нетипизированный доступ к данным (см. QP8 API)
Классы LINQ to SQL	Типизированный доступ к данным (см. Классы LINQ to SQL)
Entity Framework	См. Использование Entity Framework

Существует механизм JS-интеграции для организации двустороннего взаимодействия бекэнда и веб-приложения для пользовательского действия (см. JS-интеграция для веб-приложений пользовательских действий).

5.1. Установка NuGet-пакетов

Для работы с NuGet-пакетами используется репозиторий, который доступен по адресу `http://nuget.dev.qsupport.ru/nuget`.

Пакеты устанавливаются с помощью библиотеки пакетов или с помощью командной строки менеджера пакетов среды разработки Visual Studio.

Для установки с помощью библиотеки, необходимо:

1) Вызвать в `Solution Explorer` контекстное меню для пункта `References`;
2) Выбрать пункт `Manage NuGet Packages for Solution`. В настройках библиотеки пакетов задать адрес репозитория. В открывшейся вкладке отобразятся доступные пакеты.
3) Для того чтобы установить пакет, необходимо нажать на кнопку `Install`. Описание пакета выводится в поле `Description`, зависимости в поле `Dependencies`.

Для установки с помощью командной строки менеджера пакетов, необходимо:

1) Вызвать консоль `Package Manager Console`;
2) В выпадающем списке `Default project` выбрать проект, в который требуется добавить пакет;
3) Установить пакет с помощью команды `Install-Package`: `Install-Package <имя пакета>`;

Пакеты, описанные в руководстве:

1) `QA.Validation.Xaml.Extensions` (см. Порядок разработки валидатора);
2) `Quantumart.QP8.BLL` (см. Классы из пространства имён «Quantumart.QP8.BLL.Services.API»);
3) `Quantumart` (см. QP8 API (кроме подраздела «Классы из пространства имён „Quantumart.QP8.BLL.Services.API“»), Классы LINQ to SQL, Использование Entity Framework);
4) `QP8.EntityFramework6` (см. Использование Entity Framework);
5) `QP8BackendApi.Interaction` (см. JS-интеграция для веб-приложений пользовательских действий);

6) `QA.Core.Engine` (см. Процедура создания и добавления нового типа страницы в Систему, Процедура создания и добавления нового типа виджета в Систему);

5.2. Использование Виджетной платформы для ASP.NET MVC

5.2.1. Архитектура платформы

В продукте используются понятия «Страница» и «Виджет».

Под страницей понимается объект, который требуется использовать для работы с данными на странице веб-сайта. Экземпляр страницы обладает URL.

Виджет предназначен для использования на различных страницах веб-сайта в качестве одной из её составляющих. Экземпляр виджета не обладает URL.

Примечание: в структуре данных продукта эти объекты имеют минимальные отличия.

Название	Описание
Контент «ItemDefinition»	Данные о типах элементов (страницы веб-сайта, виджеты). Контент содержит базовые поля, общие для всех типов виджетов и страниц веб-сайта.
Группа контентов «Pages and widgets»	Контенты-расширения для страниц и виджетов. Контент-расширение содержит поля, уникальные для типа страницы или виджета (отсутствующие в контенте «ItemDefinition»).
Контент «AbstractItem»	Данные обо всех экземплярах страниц и виджетов, созданных в Системе

Структура контентов

Поля контента «ItemDefinition»

Название	Тип	Описание
Title	String	Название страницы или виджета
Name	String	Идентификатор страницы или виджета
PreferredContentId	Numeric	Идентификатор контента-расширения
FriendlyDescription	String	Уникальное описание страницы или виджета. **Примечание:** в качестве значения допускаются только латиница и цифры.
CategoryName	String	Название категории виджета. Используется для группировки виджетов при выводе их в ГПИ виджетной платформы.
Description	String	Описание страницы или виджета
IconUrl	Image	Файл с пиктограммой для виджета. **Примечание:** используется в ГПИ виджетной платформы.
IsPage	Boolean	Указатель, что статья содержит данные о странице, а не виджете

Поля контента «AbstractItem»

Название	Тип	Описание
Title	String	Имя элемента
Name	String	URI для страницы. **Примечание:** используется только для страниц. **Примечание:** применяется в URL страницы.
Parent	One-to-Many Relation	Родительский элемент
IsVisible	Boolean	Флаг доступности для отображения
IsPage	Boolean	Флаг, является ли данный элемент страницей
Regions	Many-to-Many Relation	Регионы, для которых отображается данный элемент
ZoneName	String	Имя зоны, в которой находится виджет.

		Примечание: используется только для виджетов.
Description	String	Описание
Discriminator	One-to-Many Relation	Тип страницы или виджета
VersionOf	One-to-Many Relation	Оригинальная страница (страница, для которой создается версия)
Culture	One-to-Many Relation	Языковая культура
ExtensionId	Numeric	Идентификатор контента присоединённой статьи

5.2.2. Создание экземпляра страницы

Для управления страницами продукт содержит пользовательское действие «Manage Pages».

Пользовательское действие «Manage Pages» содержит ГПИ, в котором Разработчику доступны следующие возможности:

1) управление (CRUD) структурой страниц,
2) получение данных по интересующей странице.

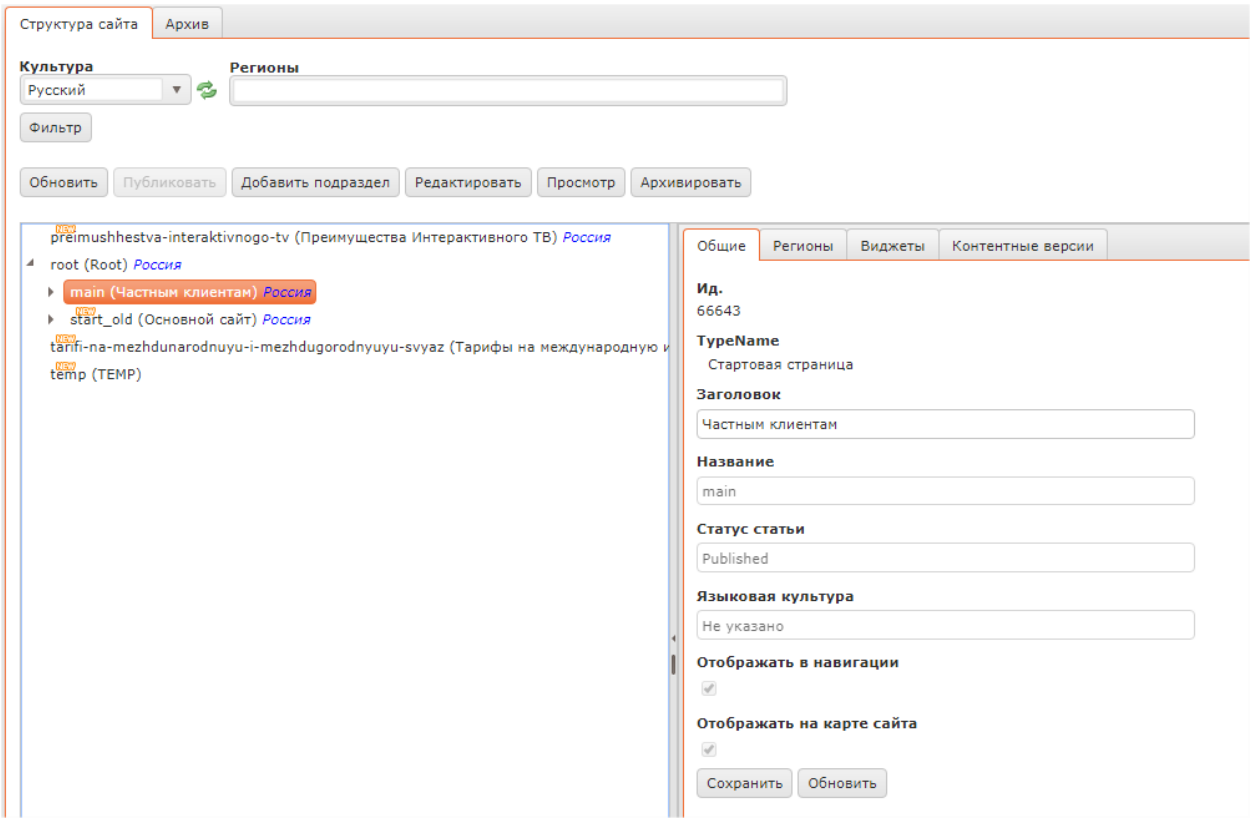

Рисунок 5.2.2-1. ГПИ пользовательского действия «Manage Pages».

Структура страниц представлена в виде дерева статей из контента «AbstractItem». Для добавления новой страницы требуется:

1) выбрать в дереве статей страницу, которая должна стать родительской страницей для новой страницы;
2) нажать «Добавить подраздел».

После этого Разработчику потребуется указать основные данные для экземпляра страницы:

Название	Описание
Заголовок	Значение поля «Title»
Название	Значение поля «Name»
Тип страницы	Выбор типа страницы из списка существующих.

	Значение поля «Discriminator».

Затем с помощью формы создания статьи Разработчику предлагается завершить создание статьи в контенте «AbstractItem» с заполнением прочих полей данными, которые должен использовать экземпляр.

Примечание: в форме создания статьи часть полей автоматически заполняется данными, заданными Разработчиком изначально.

После создания экземпляра страницы он может быть вызван по URL.

В случае, когда существующие в Системе типы страницы не подходят для решения существующей задачи, требуется создать новый тип страницы.

5.2.3. Процедура создания и добавления нового типа страницы в Систему

1. В бекэнде в контенте «ItemDefinition» создать новый элемент для типа страницы.
2. При необходимости использования на страницах данного типа дополнительных полей создать в бекэнде контент-расширение. В контенте создать все необходимые дополнительные поля для использования на странице. Затем необходимо указать идентификатор нового контента в созданной ранее статье в контенте «ItemDefinition».
3. В бекэнде выполнить генерацию классов LINQ to SQL. В случае использования Visual Studio скопировать созданные файлы в проект в Visual Studio.

   Примечание: для генерации классов используется шаблон `ItemDefinitions.tt` (содержится в дистрибутиве продукта в исходном проекте).

4. Создать модель для страницы.
   В атрибутах класса следует указать `PageDefinitionAttribute`. Пример:

   ```
   [PageDefinition("Страница новостей", Discriminator =
   ItemDefinitions.NewsPageExtension)]
   ```

   Класс требуется наследовать от `AbstractPage`.
5. Создать контроллер.
   У класса контроллера следует указать `ControlsAttribute`. Пример:

   ```
   [Controls(typeof(NewsPage))].
   ```

   Класс контроллера требуется наследовать от `ContentControllerBase<T>`, где `T` – класс страницы.
6. Создать представление.

Внимание: для использования на странице ГПИ Виджетной платформы в разметку страницы следует добавить код для вызова ГПИ.

5.2.4. Создание экземпляра виджета

Для создания экземпляра виджета рекомендуется использовать пользовательское действие «Manage widgets».

Внимание: для работы пользовательского действия «Manage widgets» в его настройках в качестве значения URL следует указать URL, по которому доступен разрабатываемый веб-сайт (в случае локального использования допускается использовать значение `localhost`).

Пользовательское действие «Manage widgets» содержит панель управления, в которой Разработчику доступны следующие возможности:

Название	Описание
Edit page	Выводит страницу «Свойства статьи» бекэнда для текущей страницы веб-сайта
Organize widgets	На текущей странице веб-сайта выводятся зоны для размещения виджетов. Также выводится список категорий виджетов. Каждая категория может быть развёрнута для получения списка относящихся к ней виджетов.

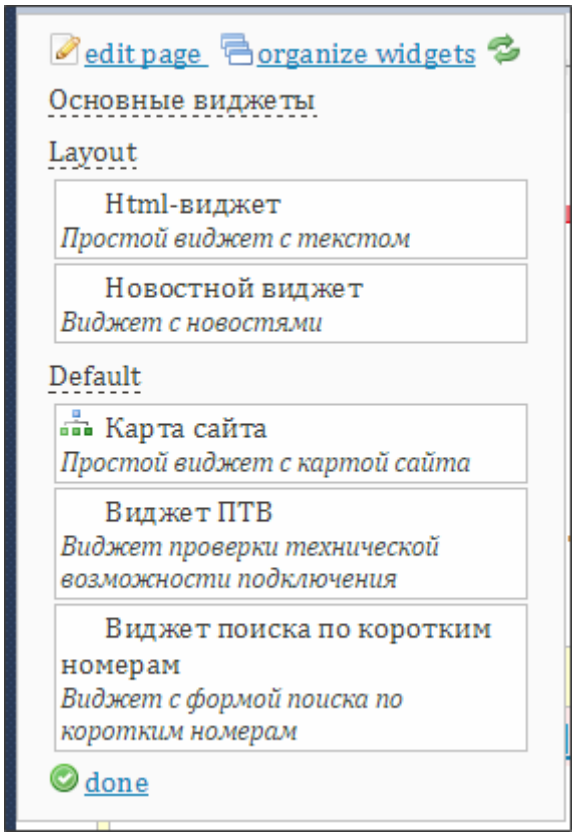

Рисунок 5.2.4-1. Панель управления для Виджетной платформы.

С использованием ГПИ платформы можно разместить любой из существующих виджетов в любой из зон страницы путём перетаскивания виджета из списка в зону страницы. После размещения виджета в подходящей зоне Разработчику предлагается создать статью (в контенте «AbstractItem») с указанием в ней данных, которые должен использовать выбранный тип виджета.

Примечание: в форме создания статьи часть полей автоматически заполняется данными, заданными Разработчиком изначально.

Также Разработчику предлагается ввести данные в поля из контента-расширения. После создания статьи для экземпляра виджета на страницу в заданной зоне выводится ГПИ виджета с данными, заданными в статье при его создании.

Примечание: если осуществляется кэширование вёрстки (изменения на странице в результате размещения на ней виджета появляются только после устаревания кэша), то для отключения кэширования в конфигурационном файле веб-сайта (`web.config`) для элементов раздела `outputCacheProfiles` следует задать `enabled="false"`.

Для управления уже размещёнными на странице виджетами в ГПИ доступны следующие опции:

Название	Описание
Edit	Выводит для виджета страницу бекэнда с формой изменения статьи
Remove	Удаляет виджет из зоны страницы

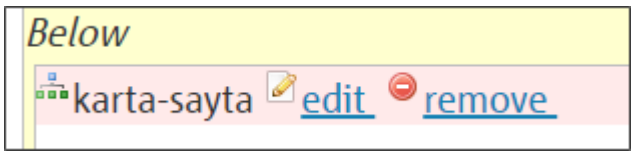

Рисунок 5.2.4-2. Опции для работы с виджетом, размещённым на странице в зоне «Below».

В случае, если существующие в Системе типы виджета не позволяют решить существующую задачу, требуется создать новый тип виджета.

5.2.5. Процедура создания и добавления нового типа виджета в Систему

1. Подготовить в бекэнде структуру данных для виджета.

В контенте «ItemDefinition» создать статью с основными сведениями по типу нового виджета.

При необходимости использования в виджете уникальных данных:

1) создать контент-расширение (в группе контентов «Pages and widgets»),
В контенте должна быть задана связь с контентом «ItemDefinition» (поле `ItemId`, тип «Связь» O2M, активировано свойство «Агрегированное»).
2) указать идентификатор созданного контента-расширения в статье с основными сведениями по типу виджета (в поле `PreferredContentId`).

2. Для контентов сайта сгенерировать классы LINQ to SQL и/или сопоставление для EF (пункт «Собрать контенты» (Assemble contents) в контекстном меню сайта).

Примечание: дополнительно генерируются MAP-файлы с данными о соответствии структуры объектов QP структуре БД.

Примечание: директория, в которую размещаются созданные файлы, задаётся в свойствах сайта (группа свойств «Расположение файлов сборок» (Assembly Files Location)).

Примечание: доступна возможность загрузить файлы сразу после их создания (см. Основные параметры (Basic parameters), параметр «Внешняя разработка»).

Внимание: повторную генерацию классов и MAP-файлов требуется осуществлять после любого изменения структуры данных сайта.

3. Использовать полученные классы и MAP-файлы при разработке виджета. В случае использования Visual Studio следует скопировать файлы в среду разработки.
4. Разместить код созданного виджета в Системе.

Описание кода для виджета

Модель

Взаимодействие с БД осуществляется с использованием классов LINQ to SQL (см. Классы LINQ to SQL) либо сопоставления для EF (см. Использование Entity Framework).

Представление

Представление разрабатывается на основе Razor-шаблонов (CSHTML).

Используется частичное представление (partial view).

При создании разметки следует подготовить зоны для размещения виджетов. Пример кода:

```
@{ Html.WidgetZone("Название зоны").Render(); }
```

Зона может быть:

1) локальная (используется на одной странице),

2) рекурсивная (используется на странице и всех её дочерних страницах).

Контроллер

Объект `CurrentItem` содержит данные из полей контента-расширения.

5.3. Использование Виджетной платформы для ASP.NET Core

5.3.1. Список библиотек и общее описание

Название	Версия	Описание
QA.DotNetCore.Caching	1.5.*	Интерфейс и реализация провайдера кэширования для .NET Core.
QA.DotNetCore.Engine.Abstractions	1.5.*	Интерфейсы и абстрактные классы для виджетной
QA.DotNetCore.Engine.AbTesting	1.5.*	Возможности A/B тестирования на виджетной платформе
QA.DotNetCore.Engine.AbTesting.Configuration	1.5.*	Простое конфигурирование A/B тестирования на виджетной платформе
QA.DotNetCore.Engine.CacheTags	1.5.*	Инфраструктура поддежрки кэш-тегов в виджетной платформе
QA.DotNetCore.Engine.OnScreen	1.5.*	Интеграция с админкой OnScreen.
QA.DotNetCore.Engine.Persistent.Dapper	1.5.*	Доступ к данным структуры сайта через Dapper
QA.DotNetCore.Engine.Persistent.Interfaces	1.5.*	Интерфейсы доступа к данным структуры сайта
QA.DotNetCore.Engine.QpData.	1.5.*	Реализация структуры сайта на основе QP
QA.DotNetCore.Engine.QpData.Configuration	1.5.*	Простое конфигурирование структуры сайта на основе QP
QA.DotNetCore.Engine.Reflection	1.5.*	Реализация ITypeFinder для .NET Core
QA.DotNetCore.Engine.Routing	1.5.*	Роутинг для виджетной платформы
QA.DotNetCore.Engine.Targeting	1.5.*	Таргетирование для виджетной платформы
QA.DotNetCore.Engine.Widgets	1.5.*	Серверный рендеринг для виджетов
QA.DotNetCore.Engine.Xml	1.5.*	Реализация структуры сайта на основе XML
QA.DotNetCore.Engine.Xml.Configuration	1.5.*	Простое конфигурирование структуры сайта на основе XML

5.3.2. Подключение структуры сайта из QP

Классический кейс использования виджетной платформы - использовать структуру сайта, заведенную в QP, для функционирования сайта, а именно:

- создание/управление страницами,
- роутинг,
- виджеты (с серверным рендерингом).

Структура контентов

Необходимо, чтобы в QP была создана база и в ней существовала определенная структура контентов.

Внимание: Платформе не важно, как называется тот или иной контент (т.е. поле *Content Name* из свойств контента не проверяется) – важно только ***NetName*** (поле ***Name (singular)*** из LINQ Mapping Parameters).

Аналогично с полями контентов: инвариантами являются LINQ ***Property Name***, а не ***Name***. Это сложившийся подход: в старой версии платформы для доступа к данным использовался linq2sql-слой доступа к этим данным, который опирался на ***NetName***.

Ниже приведено описание необходимой структуры в QP (приведены ***NetName*** контентов и необходимых полей):

- Контент **QPAbstractItem**. Его необходимые поля:
  - ***Name,***
  - ***Title,***
  - ***Parent,***
  - ***IsVisible,***
  - ***ZoneName,***
  - ***IndexOrder,***
  - ***ExtensionId,***
  - ***VersionOf,***
  - ***Discriminator.***

  Разработчики могут заводить и другие поля, если проект того требует, специальных ограничений нет.

- Контент **QPDiscriminator** (обычное название, как правило, **ItemDefinition**). Его необходимые поля:
  - ***Name,***
  - ***IsPage,***
  - ***TypeName,***
  - ***Title,***
  - ***Description,***
  - ***IconUrl,***
  - ***IconClass,***
  - ***IconIntent,***
  - ***PreferredContentId.***

- Контенты-расширения для **QPAbstractItem** для типов страниц и виджетов.

  Netname и обычные названия для этих контентов не регламентированы. На практике принято оставлять у этих контентов пустой ***netname*** и называть их как:

  `[название типа страницы/виджета]Extension`, например, `TextPageExtension.`

Единственное правило для этих контентов: они должны быть расширениями контента **QPAbstractItem**. Это достигается добавлением поля ***ItemId***, являющееся O2M-связью на **QPAbstractItem** с включенным признаком *Aggregated*. (подробнее о контентах-расширениях см. в документации QP).

Примечание: Назначение контентов структуры сайта.

QPAbstractItem – это контент, где заведены все страницы и виджеты, из которых состоит структура сайта. Структура имеет древовидный характер (за счёт поля ***Parent***).

QPDiscriminator – это контент с описанием типов страниц и виджетов.

Контенты-расширения для **AbstractItem** хранят в себе дополнительные поля, специфичные только для конкретных типов страниц или виджетов (например: если виджет новостей должен иметь свойство «Количество новостей для отображения», то это свойство должно быть заведено в контенте-расширении для этого виджета).

Правила контентного наполнения структуры сайта

Структура сайта – это дерево страниц и виджетов. Страница или виджет определяется свойством *IsPage* у **QPDiscriminator**.

Структура страниц задаёт роутинг на сайте:

- стартовая страница имеет адрес `/`,
- её дочерняя страница 1 уровня `/page1` (где page1 – это значение её поля ***Name***),
- её дочерняя страница 2 уровня `/page1/page2` и т.д.

Если виджет является дочерним элементом по отношению к странице, то он размещен на ней.

У виджета должно быть обязательно заполнено поле ***ZoneName***, которое означает место размещения на странице.

Какие именно зоны поддерживает страница определяет разработчиком - размещая в cshtml-шаблонах конструкции вида `@await Component.WidgetZone(Html, "Meta")`

Если у виджета не будет указана зона, или указанная зона не будет поддерживаться страницей, то он будет проигнорирован.

Примечание: зоны могут быть *обычными*, *глобальными* или *рекурсивными*.

Тип зоны определяется согласно конвенции наименования:

- если она начинается на префикс **Site** или **Global**, то она *глобальная*,
- если на **Recursive**, то *рекурсивная*,
- иначе *обычная*.

Если необходимо разместить один и тот же виджет на большом количестве страниц, то можно привязать его только раз к одной странице в такой зоне.

Виджеты в глобальных зонах отличаются от рекурсивных тем, что обязательно должны быть привязаны к стартовой странице (рекурсивные могут быть привязаны к любой), и распространяются не только на страницы структуры сайта, но и на весь сайт, включая «обычные» страницы, не обслуживаемые контроллерами структуры сайта.

Есть возможность ограничить распространение рекурсивного/глобального виджета по сайту: нужно добавить в **QPAbstractItem** поля ***AllowedUrlPatterns*** и ***DeniedUrlPatterns***, в которых можно задать белые или черные списки паттернов адресов страниц для отображения виджета.

Виджет может быть размещен не только на странице, но и в виджете, если тип родительского виджета поддерживает нужную зону.

Внимание: Страница не может быть размещена в виджете (т.е. родительским элементом для страницы не может быть виджет): это контентная ошибка, такие страницы не будут работать.

Структура сайта всегда должна иметь *корневую страницу* (т.е. страницу с определённым типом *«корневая страница»*). Она не несёт особой функциональности, и ей не соответствует никакой реальный URL-адрес страницы. Она нужна как маркер для виджетной платформы: её дискриминатор должен быть равен ***root_page*** (может быть переопределено при конфигурации свойством **RootPageDiscriminator**).

Второй уровень структуры сайта, после корневой страницы — это уровень *стартовых страниц*. Может быть больше одной стартовой страницы.

Например, если необходимо, чтобы одна база обслуживала несколько сайтов, допустим `somesite.ru` и `lk.somesite.ru.` У этих сайтов свои независимые структуры, но они могут использовать в них какие-то общие типы страниц и виджетов.

Для каждого такого сайта нужно заводить отдельную стартовую страницу. Для типа *«стартовая страница»* в таком случае нужно предусмотреть некое поле, исходя из которого можно ассоциировать запросы, идущие к `somesite.ru` и `lk.somesite.ru`, к своим стартовым страницам.

На практике обычно используется поле **DnsBindings**, в котором перечисляются шаблоны URL-хостов, которые должны относится к этой стартовой странице.

Как использовать

Инструкция для разработчика, как подключить функциональность структуры сайта от виджетной платформы:

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.QpData.Configuration**.
2. Сконфигурировать **IServiceCollection,** в `Startup.cs,` вызвав метод `services.AddSiteStructureEngine (options => …);` описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. В `Startup.cs` использовать middleware, добавляющую структуру сайта в ***pipeline*** запроса `app.UseSiteStructure();`
4. Зарегистрировать `route/endpoint`, который будет обрабатывать запросы к страницам структуры сайта.
   Поддерживается 2 варианта:
   - В ASP.NET Core до 3 версии можно активировать с помощью `services.AddMvc(o => o.EnableEndpointRouting = false),` добавить `route:` в методе `app.UseMvc` зарегистрировать `routes.MapContentRoute("default", "{controller}/{action=Index}/{id?}");`
   - в ASP.NET Core 3 можно активировать с помощью `EndpointRouting`, который используется по умолчанию, и в методе `app.UseEndpoints` вызвать `endpoints.MapSiteStructureControllerRoute();`
5. Для всех типов страниц и виджетов создать классы моделей.

   Желательно заводить их все вместе в одной сборке для удобства регистрации типов в `services.AddSiteStructureEngine (options.TypeFinder.RegisterFromAssemblyContaining).`

   Свой класс нужно завести также и для корневой и стартовой страницы.

Классы страниц должны наследоваться от **AbstractPage**, классы виджетов от **AbstractWidget**. Стартовая страница должна также реализовывать интерфейс **IStartPage**.
В этих классах можно создавать поля, которые соответствуют полям, заведенным в соответствующих контентах-расширениях в QP.

Это следует делать с помощью метода `GetDetail`.

Например, `public string Text { get { return GetDetail("Text", String.Empty); } }`.

Если поле является many-to-many, то вместо **GetDetail** надо использовать **GetRelationIds**, что даст список ID объектов, соответствующих этому полю.

6. Для всех типов страниц создать контроллеры.

   Правила заведения контроллеров зависят от конвенции того, как классам страниц соответствуют контроллеры.

   Конвенция по умолчанию – это конвенция **Name**, она предполагает, что контроллер должен называться также как класс страницы (`TextPage -> TextPageController`).
   Конвенция **Attribute** предполагает, что контроллер должен быть помечен атрибутом `SiteStructureControllerAttribute`, в котором должен быть указан тип страницы. Конвенцию можно настраивать при конфигурировании с помощью опции **ControllerMapperConvention**.

   Для возможности удобно получать **AddSiteStructureEngine** из контроллера текущую и стартовую страницы, его следует унаследовать от **ContentControllerBase<T>**, где **T** – это класс типа страницы.

7. Для всех типов виджетов создать viewcomponent.

   Правила заведения viewcomponent зависят от конвенции того, как классам виджетов соответствуют viewcomponent.
   Конвенция по умолчанию – это конвенция **Name**, она предполагает, что viewcomponent должен называться также как класс виджета (`TextWidget -> TextWidgetViewComponent`).

   Конвенция **Attribute** предполагает, что viewcomponent должен быть помечен атрибутом `SiteStructureComponentAttribute`, в котором должен быть указан тип виджета.

   Конвенцию можно настраивать при конфигурировании **AddSiteStructureEngine** с помощью опции **ControllerMapperConvention**. Для возможности удобно получать из viewcomponent текущий виджет, его следует унаследовать от, **WidgetComponentBase<T>**где **T** – это класс типа виджета.

8. Создать нужные view для методов контроллеров и для viewcomponent.
9. Для подключения таргетирования см. «Подключение таргетирования к сайту»
10. Для подключения кэштегов см. «Подключение функциональности кэштегов»
11. Для подключения onscreen см. «Подключение и использование режима Onscreen»
12. Для поиска элементов по структуре сайта, нужно использовать **IItemFinder**, он уже добавлен в IoC-контейнер. В него нужно передать предикат для поиска и элемент, в чьём поддереве нужно выполнять поиск.

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddSiteStructureEngine (options => …)` с параметром **Action<SiteStructureEngineOptions>**. Таким образом, можно задать и переопределить опции, которые есть в объекте **SiteStructureEngineOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	Настройки для взаимодействия с QP (строка подключения, **SiteId, live/stage** итд), в котором хранится структура сайта.

options.TypeFinder.RegisterFromAssemblyContaining

Обязательная	Регистрация сборки, которая содержит все классы страниц и виджетов, созданных разработчиком. Если они не будут зарегистрированы, то ни один узел дерева структуры сайта не будет инстанцирован. Пример: `options.TypeFinder.RegisterFromAssemblyContaining<RootPage, IAbstractItem>()` (т.е. необходимо зарегистрировать все типы, реализующие **IAbstractItem**, находящиеся в той же сборке, где находится класс **RootPage**.)

options.UrlHeadPatterns

Необязательная	Значение по умолчанию - `new List<HeadUrlMatchingPattern> { new HeadUrlMatchingPattern { Pattern = "/"} }` (т.е. пустой паттерн, ни в хосте, ни в первых сегментах пути нет никаких таргетирующих токенов).

Примечание: Пояснение к понятиям «голова»/«хвост» URL

URL любой страницы (без querystring и fragment) можно представить как `{схема}://{хост}/{путь}.`

URLстраницы сайта, работающего на виджетной платформе можно представить немного по-другому, в виде: `{схема}://{голова}/{путь от стартовой страницы}/{хвост}.`

«Голова» - это хост сайта + опционально первые несколько сегментов пути; в этих сегментах и хосте могут быть заключены значения таргетирования (например, регион, культура итд).

«Хвост» - это последние сегменты пути; в этих сегментах могут быть **routevalues** (например, **action**, **id** итд).

Примеры:

`https://msk.somesite.ru/en-us/welcome/general-information/.`

`//msk.somesite.ru/en-us` – это голова, причём токены moskva и en-us могут быть использованы как значения таргетирования.

Путь от стартовой страницы - `welcome/general-information`. Хвост пустой.

`https://msk.somesite.ru/customers/products/mobile/tariffs/details/ne-odin-doma/`
`//msk.somesite.ru` – это голова.

Путь от стартовой страницы - `customers/products/mobile/tariffs.`

Хвост - `details/ne-odin-doma.`

options.RegisterUrlHeadTokenPossibleValues

Необязательная	по умолчанию ни одного I**HeadTokenPossibleValuesProvider** не зарегистрировано. Используется в связке с **options.UrlHeadPatterns**. Если в **UrlHeadPatterns** заданы токены, то можно через **RegisterUrlHeadTokenPossibleValues** зарегистрировать один или несколько классов-реализаций **IHeadTokenPossibleValuesProvider**. К примеру, если `UrlHeadPatterns = new List<HeadUrlMatchingPattern> { new HeadUrlMatchingPattern { Pattern = "/{region}"} },` т.е. первым сегментом пути может идти токен со значением текущего региона. Когда платформа обрабатывает запрос `http://somesite.com/segment1/segment2/segment3`, для корректной работы ей нужно понимать: `segment1` – это допустимое значение для токена **region** или нет? Если допустимое, то «головой» URL будет считаться `//somesite.com/segment1`, иначе – просто `//somesite.com.`

options.DefaultUrlTailPattern

Необязательная	Имеет смысл ТОЛЬКО для **Endpoint routing** (дефолтный режим роутинга в ASP.NET Core с 3 версии). Значение по умолчанию - `new TailUrlMatchingPattern {Pattern = "{action=Index}/{id?}" }` Шаблон по умолчанию для всех контроллеров.

options.UrlTailPatternsByControllers

Необязательная	Имеет смысл ТОЛЬКО для **Endpoint routing** (дефолтный режим роутинга в ASP.NET Core с 3 версии). Значение по умолчанию – пустое, т.е. по умолчанию у всех контроллеров только один шаблон «хвоста» - тот, что задан в **options.DefaultUrlTailPattern**. Шаблоны «хвоста» URL, которые будут использоваться только для определенных контроллеров структуры сайта (т.е., по сути, для определенных типов страниц). Некоторые типы страниц могут иметь свои специфичные потребности, например страница - новостей, где «хвост» может иметь вид: `/{year}/{month}/{day}/{article_alias}.`

options.ItemDefinitionConvention

Необязательная	Значение по умолчанию – ***ItemDefinitionConvention.Name*** Конвенция того, как **ItemDefinition** сопоставляется с .Net классами страниц и виджетов, которые заводит разработчик. **Примечание: ItemDefinition** – это контент с netname **QPDiscriminator** - в нем хранятся все типы страниц и виджетов. **ItemDefinitionConvention.Name** - предполагает совпадение поля **TypeName** у **ItemDefinition** и имени класса .Net **ItemDefinitionConvention.Attribute** – **[пока не реализовано]** предполагает наличие атрибута у класса .Net, в котором задаётся дискриминатор **ItemDefinition**.

options.ControllerMapperConvention

Необязательная	Значение по умолчанию – **ControllerMapperConvention.Name** Конвенция того, как .Net классам страниц соответствуют контроллеры. **ControllerMapperConvention.Name** - предполагает, что контроллер должен называться также как класс страницы (`TextPage -> TextPageController`) **ControllerMapperConvention.Attribute** - предполагает, что контроллер должен быть помечен атрибутом **SiteStructureControllerAttribute**, в котором должен быть указан тип страницы.

options.ComponentMapperConvention

Необязательная	Значение по умолчанию – **ComponentMapperConvention.Name** Конвенция того, как .Net классам виджетов соответствуют viewcomponent. **ComponentMapperConvention.Name** - предполагает, что компонент должен называться также как тип виджета (`TextWidget -> TextWidgetViewComponent`) **ComponentMapperConvention.Attribute** - предполагает, что компонент должен быть помечен атрибутом **SiteStructureComponentAttribute**, в котором должен быть указан тип виджета.

options.LoadAbstractItemFieldsToDetailsCollection

Необязательная	Значение по умолчанию – ***true***. Нужно ли загружать ли в нетипизированную коллекцию всех свойств страниц и виджетов поля из **AbstractItem**. Нетипизированная коллекция всех свойств страниц и виджетов – это коллекция к которой имеют доступ методы **GetDetail** и **GetRelationIds**. Если эта настройка ***false***, то она состоит только из полей контента-расширения, если ***true*** – то из полей и **QPAbstractItem**, и контента-расширения.

options.QpConnectionString

Не нужно использовать	если строка подключения уже задана через **options.UseQpSettings**

options.QpDatabaseType

Не нужно использовать	если уже задано через **options.UseQpSettings**. Возможные значения: ***postgres***/***mssql***. Значение по умолчанию – ***mssql***.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. ***Id*** сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.RootPageDiscriminator

Необязательная	Значение по умолчанию - ***root_page***. Дискриминатор корневого элемента структуры сайта (поле ***Name*** в контенте **QPDiscriminator** для записи, соответствующей корневой странице).

options.UploadUrlPlaceholder

Необязательная	Значение по умолчанию - `<%=upload_url%>`. Плейсхолдер, который заменяет собой путь до библиотеки сайта в QP.

options.SiteStructureCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 20, 0).` Длительность кэширования уже построенной структуры сайта.

options.QpSchemeCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 0, 30)`. Длительность кэширования схемы QP (таблица SITE, CONTENT_ATTRIBUTE и т.п.).

options.ItemDefinitionCachePeriod

Необязательная	Значение по умолчанию - `new TimeSpan(0, 20, 0)`. Длительность кэширования **ItemDefinition**.

5.3.3. Использование структуры сайта из QP как сервис

Иногда возникают задачи, когда необходимо получить структуру сайта, и при этом полноценный сайт с серверным рендерингом по ней строить не нужно. Пример такой задачи – это построение API для сайтов, работающих на *react*.

Структура контентов и правила наполнения базы QP такие же как для кейса «Подключение структуры сайта из QP».

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.QpData.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddSiteStructure(options => …)`; См. Опции конфигурации.
3. Для работы со структурой сайта заинжектить **IAbstractItemStorageProvider** в нужный класс, использовать у него метод `Get()`.

Опции конфигурации

Все опции для настройки по этому кейсу подробно описаны в кейсе «Подключение структуры сайта из QP». Список этих опций:

- **UseQpSettings**
- **QpConnectionString**
- **QpDatabaseType**
- **SiteId**
- **IsStage**
- **RootPageDiscriminator**
- **UploadUrlPlaceholder**
- **LoadAbstractItemFieldsToDetailsCollection**
- **SiteStructureCachePeriod**
- **QpSchemeCachePeriod**

5.3.4. Подключение таргетирования к сайту

На практике таргетирование используется совместно с кейсом «Подключение структуры сайта из QP». Некоторые концепции таргетирования (такие как, провайдеры таргетирования) не зависит от структуры сайта, и могут использоваться без неё.

Общая информация

Одна и та же страница сайта может выглядеть по-разному (или даже существовать, или нет) для разных пользователей сайта. Примеры:

1) У авторизованных пользователей на страницах сайта могут появляться новые элементы, появляться целые новые разделы на сайте.
2) У разных типов авторизованных пользователей (например, пользователи b2c и b2b) могут быть свои разделы.
3) Пользователи разных регионов могут видеть разные страницы по-разному (например, какие-то виджеты могут показываться только для пользователей определенных регионов; продукты в разных регионах могут отличаться).
4) Мультиязычность: пользователи разных культур (допустим на сайте можно выбирать язык) могут видеть разные версии сайта – как на разных языках, так и структурно разные (т.е., например, английская версия сайта может быть очень урезанной и с малым числом страниц)

Пользователи сайта делятся на аудитории по некоторым признакам, которые называются ***признаками таргетирования***. Признаки таргетирования могут влиять по-разному на серверный процесс обработки http-запроса от пользователя: например, признак «Выбранный язык» может влиять на выбор локализации файлов ресурсов resx, а признак «Регион пользователя» на набор продуктов, доступных в этом регионе.

Ниже описано как виджетная платформа работает с признаками таргетирования и как они влияют (при должной настройке) на её работу.

Реализуем специальные классы: ***провайдеры и фильтры таргетирования***, а также использовать **ITargetingContext** – ***контекст таргетирования***.

Провайдеры таргетирования

Их назначение – предоставлять словарь
`([ключ таргетирования], [текущее значение таргетирования])`.
Должны реализовывать интерфейс **ITargetingProvider** с единственным методом `IDictionary<string, object> GetValues()`.
Для каждого сайта можно создавать более одного провайдера. Единственное ограничение, чтобы они возвращали словари с разными ключами, потому что middleware **UseTargeting** вызовет метод `GetValues` у каждого из них и объединит в единый словарь.

Контекст таргетирования

В любой класс, которому нужно знать текущие значения таргетирования, нужно инжектить **ITargetingContext** – контекст таргетирования. С помощью его метода `object GetTargetingValue(string key);` можно получать значение, вычисленное в соответствующем провайдере таргетирования.

Фильтры таргетирования

Если сайт построен на базе структуры сайта (кейс «Подключение структуры сайта из QP»), и необходимо, чтобы таргетирование воздействовало на него (различные версии сайта для различных значений таргетирования), следует использовать фильтры таргетирования.

Такой фильтр — это класс, реализующий интерфейс **ITargetingFilter** или наследующийся от **BaseTargetingFilter (namespace QA.DotNetCore.Engine.Abstractions.Targeting)**.

Основные методы этого интерфейса, которые требуется реализовать:

- `bool Match(IAbstractItem item),`
  где **IAbstractItem** – это элемент структуры сайта, т.е. страница или виджет.
  Смысл метода: «удовлетворяет ли элемент структуры сайта item текущим значениям признаков таргетирования?».
- `IEnumerable<T> Pipe<T>(IEnumerable<T> items) where T : IAbstractItem;`
  Смысл метода: отфильтровать множество смежных элементов уровня.
  Его стандартная реализация из **BaseTargetingFilter** базируется на методе `Match` (Pipe возвращает только те элементы, которые проходят проверку Match). Её можно переопределять, в случае если на проекте используется более сложная логика таргетирования, когда для того, чтобы отфильтровать элемент, требуется знание об его смежных элементах.

Несколько фильтров можно объединять в композицию фильтров (с помощью класса **UnitedFilter** или операндов **+/&**).

С помощью фильтра (и его метода **Pipe**) можно отфильтровать всю структуру сайта, как страницы, так и виджеты. Как правило для этого все страницы (и виджеты) должны иметь в QP некоторое специальное поле (поля), участвующие в логике таргетирования: например, many-to-many поле **«Регионы»** или **«Культура»**.

Рассмотрим пример, как может быть реализован простой фильтр по региону. Сначала сделаем специальный интерфейс-маркер для тех элементов структуры сайта, у которых есть many-to-many поле **«Регионы»** и реализуем во всех типах страниц и виджетов, в которых это нужно.

```
public interface IDemoAbstractItem : IAbstractItem
{
    IEnumerable<int> Regions { get; }
}
```

Код фильтра может выглядеть так

```
public class RegionTargetingFilter : BaseTargetingFilter
{
        public RegionTargetingFilter(ITargetingContext ctx)
        {
            Ctx = ctx;
        }

        public ITargetingContext Ctx { get; }

        public override bool Match(IAbstractItem item)
        {
            if (item is IDemoAbstractItem)
            {
                var siteItem = item as IDemoAbstractItem;

                var ctxRegion = Ctx.GetTargetingValue(TargetingKeys.Region);
                if (ctxRegion == null || !(ctxRegion is RegionDto))
                {
                    return false;//если из контекста запроса не смогли определить
регион
                }

                var itemRegionIds = siteItem.Regions;
                if (itemRegionIds == null || !itemRegionIds.Any())
                {
                    return true;//если у текущего элемента не указано ни одного
региона, то значит этот элемент предназначается для всех
                }

                return itemRegionIds.Contains((ctxRegion as RegionDto).Id);
            }
            return false;
        }
    }
```

Регистрация

Инструкция для разработчика, как подключить к сайту функциональность таргетирования от виджетной платформы.

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.Targeting**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод **services.AddTargeting()**
3. Написать и зарегистрировать в IoC-контейнере с нужным **lifetime** все нужные провайдеры и фильтры таргетирования (уже реализованный провайдер **UrlTokenTargetingProvider** регистрировать в IoC не нужно, он добавляется туда при вызове `services.AddTargeting()`)
4. Использовать middleware, вычисляющую для запроса значения таргетирования **app. UseTargeting();** при вызове этого метода нужно указать все провайдеры таргетирования, которые платформа должна использовать.

   Например:

   ```
   providers => { providers.Register<UrlTokenTargetingProvider>(); }
   ```

5. Подключить фильтры структуры сайта (если они нужны), вызвав метод `app.UseSiteStructureFilters`, при вызове этого метода нужно указать все фильтры таргетирования, которые платформа должна использовать.

Уточнения для кейса, когда в URL сайта есть токены таргетирования

Для случая, когда на сайте используется структура сайта, и в «голове» URL есть токены таргетирования (см. настройку options.UrlHeadPatterns) уже существует готовый провайдер таргетирования **UrlTokenTargetingProvider** – он извлекает значения таргетирования из URL, согласно шаблонам **options.UrlHeadPatterns**.

В этом случае полезно также уметь генерировать для каждой страницы из структуры сайта URL с учётом текущих значений таргетирования (например, при генерации главного меню). Если просто вызвать метод `GetUrl()` у страницы из структуры сайта, то вернется URL без учёта таргетирования. Если же передать инстанс **ITargetingUrlTransformator** в `GetUrl()`, то вернется URL с учётом текущих значений таргетирования и шаблонов **options.UrlHeadPatterns**. **ITargetingUrlTransformator** уже добавлен в IoC-контейнер, его просто нужно заинжектить в нужное место.

Контентные и структурные версии

Сайт при таргетировании позволяет иметь несколько смежных страниц с одним алиасом, но разными значениями полей таргетирования на одном уровне.

Пример:

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> /tariffs (id=3 regions=spb)
---> /tariffs (id=4 regions=novosib)
```

Такие страницы еще называют версиями, в зависимости от значения таргетирования пользователю покажется только какая-то одна из них.

Можно выделить два вида версий: контентные и структурные. Концептуально они различаются тем, делят ли они между собой поддерево страниц и виджетов или же имеют каждый своё собственное поддерево.

Пример контентных версий (поддерево наследуется):

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> ---> /special (id=5)
---> ---> [crosssale widget] (id=6)
---> /tariffs (id=3 regions=spb versionOf=2)
---> /tariffs (id=4 regions=novosib versionOf=2)
```

В этом случае, даже несмотря на то, что у страниц 3 и 4 нет своих подстраниц, они наследуют подстраницу special и crosssale widget, и у пользователя в регионах spb и novosib они будут.

Пример структурных версий (поддерево не наследуется)

```
/ (id=1)
---> /tariffs (id=2 regions=moscow)
---> ---> /special (id=5)
---> ---> [crosssale widget] (id=6)
---> /tariffs (id=3 regions=spb)
---> ---> /special (id=7)
---> ---> [other widget] (id=8)
---> /tariffs (id=4 regions=novosib)
---> ---> /special-novosib (id=9)
```

В этом случае поддерево у всех версий своё, редактируется независимо, и может иметь различную структуру.

Какой из вариантов версий использовать – зависит от требований и ситуации.

Примечание: В админке структуры сайта (Manage Pages) контентные и структурные версии показываются по разному: контентные показываются в отдельной вкладке:

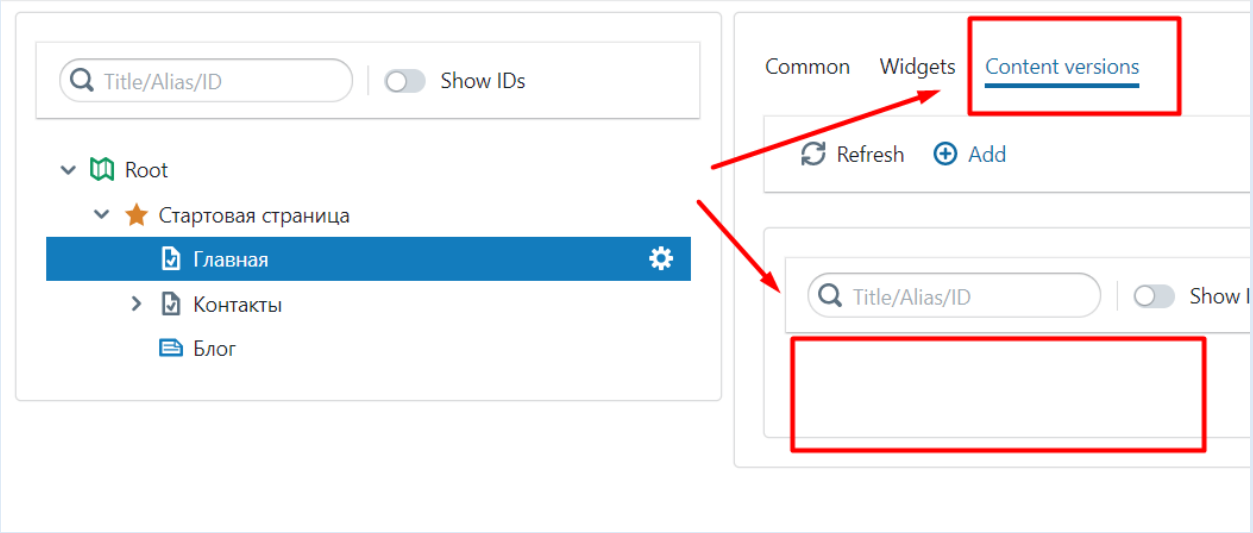

Рисунок 5.3.4-1. Manage pages – Content versions.

Структурные версии отображаются как обычные страницы, просто с тем же алиасом.

5.3.5. Подключение движка AB-тестирования к сайту

Кейс использования виджетной платформы, никак не связанный с виджетами: использовать структуру, заведенную в QP, для возможности проведения AB-тестирования на сайте.

Этот кейс может использоваться независимо от кейса «Подключение структуры сайта из QP».

Структура контентов

Для этого кейса необходимо, чтобы существовала база QP и в ней существовала определенная структура контентов.

Примечание: существует xml-скрипт, который можно «проиграть» в QP и который автоматически создаст всю необходимую структуру в базе.

Примечание: по аналогии с «Подключение структуры сайта из QP» платформе всё равно как называется тот или иной контент или поле – важно только так называемое ***NetName***. Более подробно см. в описании кейса «Подключение структуры сайта из QP».

Ниже приведено описание необходимой структуры в QP (приведены ***NetName*** контентов и необходимых полей)

1) Контент **AbTest**.
   В нем хранятся сами тесты и их базовые настройки. Тест – это некоторый эксперимент, который может состоять из одной или нескольких частей (так называемых *контейнеров*), каждая из которых по-своему меняет сайт. Также у теста должно быть хотя бы 2 варианта (иначе он не имеет смысла), может быть и больше – подробнее см. поле **Percentage**.

   Структура этого контента:
   a) **Title**.
      Наименование теста.
   b) **Percentage**.
      Вероятности выпадения того или иного варианта теста. Должно быть задано в специальном формате: положительные целые числа через запятую или точку с запятой.
      Количество чисел в этом поле определяет количество вариантов теста.
      Сумма чисел не должна быть равной 100 или любому другому числу, вероятности вариантов вычисляются исходя из пропорций указанных чисел.
      Например: «1,1» значит, что в тесте два равновероятных варианта (по 50% каждый), «1,1» означает то же что и «50,50», или скажем, «1234,1234». «1,2,3,4» значит, что в тесте 4 варианта: вариант 0 – 10%, вариант 1 – 20%, вариант 2 – 30%, вариант 3 – 40%.
   c) **StartDate**.
      Дата начала теста. С помощью него можно регулировать время, когда тест появится на сайте.
   d) **EndDate**.
      Дата окончания теста. С помощью него можно регулировать время, когда тест пропадёт с сайта.
   e) **Comment**.
      Комментарий, незначимое для функциональности поле.
   f) **Enabled**.
      Признак того, включен ли тест.

2) Контент **AbTestBaseContainer**.
   Это базовый контент для *контейнеров* тестов (т.е. составных частей тестов).
   Контейнер – это одно воздействие теста на сайт. Оно может заключаться или в добавлении js-скрипта на сайт, или в редиректе на какую-то страницу сайта. И js-скрипт, и адрес для редиректа задаются отдельно для каждого варианта теста.

Поэтому контейнер может быть 2 типов: *контейнер скриптов* или *контейнер клиентских редиректов*, их общие свойства находятся в **AbTestBaseContainer**, а различающиеся – в контентах расширениях **AbTestScriptContainer** и **AbTestClientRedirectContainer**.

Структура этого контента:

a) **Description**.
Описание воздействия контейнера на сайт. Например, «Изменение цвета такой-то кнопки».

b) **AllowedUrlPatterns**.
Ограничение для контейнера, на каких страницах сайта он должен быть. Шаблоны адресов страниц, размещенные на разных строках, возможно использование wildcard-символа * в конце шаблона.

c) **DeniedUrlPatterns**.
Исключения из значений, заданных в **AllowedUrlPatterns**.

d) **Domain**.
Домен сайта, на котором должен работать контейнер. Если не указывать – домен не будет проверяться.

e) **Precondition**.
Условие срабатывания контейнера (таргетирование).
Значение этого поля –Javascript-выражение. При его написании можно использовать переменную *ctx*. В ней есть все значения таргетирования, вычисленные на сервере и имеющиеся в контексте таргетирования (см. Контекст таргетирования). С помощью этого можно ограничивать выполнение контейнера теста только некоторой аудиторией сайта. Допустим, на сайте есть таргетирование userType, вычисляемое некоторым провайдером таргетирования (см. Провайдеры таргетирования). Тогда можно ограничить контейнер только b2c-пользователями, задав **Precondition**, например, ctx.userType == 'b2c'.

f) **Arguments**.
Объект javascript, который может быть использован в Precondition как ctx.arguments.

g) **Type**.
Тип контейнера(скрипты или редирект).

3) **AbTestScriptContainer** - контент-расширение для **AbTestBaseContainer** и контент **AbTestScript**. **AbTestScriptContainer** имеет только одно поле ScriptsInContainer, являющее many-to-many ссылкой на **AbTestScript**. Каждый привязанный к контейнеру скрипт – это вариант скрипта в определенной версии теста.

Структура **AbTestScript**

a) **Description**.
Описание скрипта.

b) **VersionNumber**.
Zero-based номер версии скрипта в контейнере. Числовое значение 0,1,2..

c) **ScriptText**.

d) Текст скрипта.

4) **AbTestClientRedirectContainer** - контент-расширение для **AbTestBaseContainer** и контент **AbTestClientRedirect**. **AbTestClientRedirectContainer** имеет только одно поле ClientRedirectsInContainer, являющее many-to-many ссылкой на **AbTestClientRedirect**. Каждый привязанный к контейнеру **AbTestClientRedirect** – это вариант редиректа в определенной версии теста.

Структура **AbTestClientRedirect**

a) **VersionNumber**.
Zero-based номер версии в контейнере. Числовое значение 0,1,2..

b) **RedirectUrl**.
URL редиректа

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.AbTesting.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddAbTestServices(options => …)`; описание options для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. Подключить к главному шаблону сайта (как правило `Views/Shared/_Layout.cshtml`) специальный viewcomponent, в котором есть js-скрипт, который получает АВ-тесты с сервера по специальному endpoint.
`@await Component.InvokeAsync("AbTest")`. Рекомендуется размещать его в нижней части <head>.

Правила работы АВ-тестов

За счёт **AbTestViewComponent**, на все страницы сайта, на которых возможно АВ-тестирование, размещается скрипт, который обращается на сервер по специальному endpoint, чтобы получить все скрипты, которые нужно выполнить на этой странице.

Сервер при обработке этого запроса в первую очередь определяет все контейнеры всех включенных тестов, которые подходят текущей странице.

Примечание: Тест считается включенным, когда у него `Enabled = true`, и текущая дата находится между **StartDate** и **EndDate**, если они заданы.

Примечание: Контейнер считается подходящим текущей странице, если URL этой страницы удовлетворяет полям **AllowedUrlPatterns, DeniedUrlPatterns, Domain.**

Далее сервер делает выбор варианта для каждого теста, задействованного на странице.

Количество вариантов теста определяется его полем **Percentage**. Вариант теста – zero-based значение, т.е. может принимать значения 0,1..N-1, где N – кол-во вариантов.

Выбор варианта для теста делается следующим образом:

1) Проверяем специальный query-параметр `test-{id теста}`. Он имеет самый большой приоритет. Ожидается целое неотрицательное значение. Например, test-12345=0.
2) Если выбор еще не сделан, смотрим передан ли вариант для этого теста в cookies. Имя cookie `abt-{id теста}`.
3) Если выбор еще не сделан, то выбираем один из вариантов с помощью класса **Random** и вероятности выпадения того или иного варианта теста, заданного в **Percentage**.

Примечание: обратите внимание, что выбор делается для теста, а не для контейнера, тем самым гарантируется, что для разных контейнеров, объединенных в один тест, будет сделан одинаковый выбор версии.

После этого сервер собирает ответ: каждый контейнер в нём представлен определенным js-скриптом. Для контейнера скриптов он имеет вид:

```
(function(ctx, window){{
    if(ctx && !({precondition})) return;
    window.abTestingContext['{AbTestChoiceResolver.CookieNamePrefix + test.Test.Id}'].targetedCids.push({container.Id});
    {script.ScriptText}
}})(ctx, window);
```

Для контейнера редиректов:

```
(function(ctx, window){{
    if(ctx && !({precondition})) return;
    window.location = '{redirect.RedirectUrl}';
}})(ctx, window);
```

В этом скрипте есть проверки, заданные через поле **Precondition**, а также сам скрипт/редирект выбранной версии (т.е. тот, чей **VersionNumber** равен выбранному варианту этого теста).

Дополнительно, в этот ответ помещается скрипт сохранения выбора варианта по каждому тесту в cookie. Это нужно для консистентности пользовательского опыта: чтобы при навигации по сайту у него не менялись постоянно варианты тестов, а оставались выбранные изначально при первом заходе на сайт.

Браузер, получив этот ответ, запускает все скрипты оттуда.

Опции конфигурации

При регистрации сервисов для работы АВ-тестов в `Startup.cs` вызывается метод `services.AddAbTestServices(options => …)` с параметром **Action<AbTestOptions>**.

Таким образом здесь можно задать и переопределить опции, которые есть в объекте **AbTestOptions**.

Часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	Настройки для взаимодействия с QP (**SiteId, live/stage, QpConnectionString, QpDatabaseType**), в которой хранятся тесты/контейнеры/скрипты.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. ***Id*** сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.QpConnectionString

Не нужно использовать	если строка подключения уже задана через **options.UseQpSettings.**

options.QpDatabaseType

Не нужно использовать	если уже задано через **options.UseQpSettings**. Возможные значения ***postgres*** и ***mssql***. Значение по умолчанию – ***mssql***.

options.TestsCachePeriod

Необязательная	Значение по умолчанию – `new TimeSpan(0, 1, 0)`. Длительность кэширования описаний тестов.

options.ContainersCachePeriod

Необязательная	Значение по умолчанию – `new TimeSpan(0, 1, 0)`. Длительность кэширования контейнеров тестов.

5.3.6. Подключение функциональности кэштегов

Общая информация

Для разработки высокопроизводительного сайта необходимо кэширования. В частности, оно используется и в библиотеках виджетной платформы, например, при построении структуры сайта из QP или получении AB-тестов. Также кэширование должен использовать и разработчик сайта, например, для сохранения часто запрашиваемых и относительно редко меняющихся данных из базы.

Инвалидация кэша – процесс, посредством которого удаляются записи в кэше. Стандартная стратегия инвалидации – это инвалидация по таймауту, когда разработчик указывает сколько должна «жить» та или иная запись в кэше до момента её обновления.

Есть также дополнительная стратегия: *инвалидация по кэштегам*. Она предполагает, что разработчик при добавлении записи в кэш помечает её одним или несколькими тегами. *Тег* – это некоторая строка. Помимо того, должен существовать определенный отслеживающий процесс, который может сбрасывать все кэши, привязанные к определенному тегу, при наступлении каких-то условий. При грамотном использовании такого подхода можно:

- Существенно увеличить таймаут для кэширования (если знать, что данные не меняются, нет смысла делать запросы в базу для их обновления);
- Улучшить согласованность сохраненных в кэше данных (данные, привязанные к одному тегу, как правило связаны и сбрасываются одновременно);
- Уменьшить время между обновлением данных и появлением их на сайте.

Кэштрекеры

Виджетная платформа для инвалидации кэша предполагает использование специальных классов, *кэштрекеров*.

Кэштрекер – это класс, реализующий интерфейс **ICacheTagTracker**. Он используется для отслеживания изменений по тегам, его единственный метод:

```
IEnumerable<CacheTagModification> TrackChanges()
```

Этот метод возвращает множество всех тегов, за которые отвественен кэштрекер, с привязкой к времени их последнего обновления.

Виджетная платформа может использовать множество трекеров. В определенное время, когда наступает время проверки тегов, платформа запускает метод **TrackChanges** у всех зарегистрированных кэштрекеров, отслеживает те теги, которые обновились с момента предыдущей проверки тегов, и запускает метод **InvalidateByTags** у **ICacheProvider**.

Кэштеги и QP

В проектах, использующих QP, использование тегов обычно базируется на таблице **CONTENT_MODIFICATION**. В ней хранится информация о времени последнего изменения данных по каждому контенту, причём по ***live*** и ***stage*** отдельно (например, если изменится какая-то статья в статусе **None**, то **CONTENT_MODIFICATION** для этого контента обновится только для ***stage***, для ***live*** останется прежним). Это даёт возможность использовать контенты QP с суффиксами ***live/stage*** как кэштеги: можно отслеживать изменения в **CONTENT_MODIFICATION** и инвалидировать кэш по соответствующим тегам.

В виджетной платформе уже реализован кэштрекер, работающий с таблицей **CONTENT_MODIFICATION** – ***QpContentCacheTracker***, разработчику самостоятельно нужно реализовывать кэштрекеры только для собственных систем кэштегов.

Теги у ***QpContentCacheTracker*** имеют вид `{contentName}_{siteId}_{(isStage ? "Stage" : "Live")}`

При использовании ***QpContentCacheTracker*** имеет смысл использовать механизм кодогенерации для создания **enum** с тегами в указанном выше формате для всех существующих контентов.

Пример tt шаблона:

CacheTags.tt

Пример результата работы этого шаблона:

CacheTags.cs

Пример работы с кэштегами (класс **CacheTagUtilities** создаётся tt шаблоном):

```
CacheProvider.GetOrAdd($"NewsService.GetAllByOrganization_{organizationId}",
                CacheTagUtilities.Merge(CacheTags.NewsItem,        CacheTags.NewsTag,
CacheTags.NewsTheme),
                TimeSpan.FromSeconds(30),
                () => GetAllByOrganization(organizationId));
```

Внимание: сам файл кодогенерации не является частью библиотек виджетной платформы.

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.CacheTags**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddCacheTagServices(options => …);` описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. Подключить `app.UseCacheTagsInvalidation(trackers => …).` При вызове этого метода нужно указать все кэштрекеры, которые платформа должна использовать.
4. Например, `trackers => { trackers.Register<QpContentCacheTracker>(); }`
5. [опционально] Добавить в проект **tt** шаблон для удобного использования кэштегов QP, зарегистрировать автосгенерированный класс `CacheTagUtilities в IoC: services.AddScoped<CacheTagUtilities>();`

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddCacheTagServices(options => …)` с параметром **Action<CacheTagsRegistrationConfigurator>**. Таким образом мы можем здесь задать и переопределить опции, которые есть в объекте **CacheTagsRegistrationConfigurator**.

options.InvalidateByTimer(TimeSpan interval)

Значение по умолчанию отсутствует.	Процесс инвалидации кэштегов будет запускаться по таймеру в фоновом процессе с периодичностью равной ***interval***. Если используется эта настройка, то **options.UseTimer** и **options.TimerInterval** не нужно использовать.

options.UseTimer

Значение по умолчанию отсутствует.	Значение по умолчанию - ***false*** Признак того, что процесс инвалидации кэштегов будет запускаться по таймеру.

options.TimerInterval

Значение по умолчанию отсутствует.	Значение по умолчанию - `TimeSpan.FromSeconds(30)` Периодичность запуска процесса инвалидации кэштегов по таймеру. Не имеет смысла при `options.UseTimer = false`.

options.InvalidateByMiddleware(string excludeRequestPathRegex)

Значение по умолчанию отсутствует.	Процесс инвалидации кэштегов будет запускаться в middleware (её регистрировали с помощью **app.UseCacheTagsInvalidation**), т.е. при каждом запросе, но при условии, что **Request.Path** не матчится **excludeRequestPathRegex**. С помощью **excludeRequestPathRegex** можно настраивать, для каких запросов нужно, и для каких нет, запускать инвалидацию. Например, стоит отсекать запросы к статическим ресурсам, типа js, css итд. Если используется эта настройка, то **options.UseMiddleware** и **options.ExcludeRequestPathRegex** не нужно использовать.

options.UseMiddleware

Значение по умолчанию - ***false***	Признак того, что процесс инвалидации кэштегов будет запускаться в middleware (её регистрировали с помощью **app.UseCacheTagsInvalidation**).

options.ExcludeRequestPathRegex

Значение по умолчанию отсутствует.	**Regex** шаблон для **Request.Path**, определяющий для каких запросов НЕ нужно запускать процесс инвалидации. Не имеет смысла при `options.UseMiddleware = false`.

Пример использования

Можно настроить по-разному процесс инвалидации для режимов сайта ***Stage*** и ***Live***. Как правило, желательно настроить ***stage***-сайт так, чтобы он отображал максимально новую информацию, производительность для него не так важна, и он используется гораздо меньшим числом пользователей, чем ***Live***.

Поэтому довольно распространенный подход состоит в том, чтобы для ***stage***-режима была настроена инвалидация по middleware, а для ***live*** – по таймеру.

```
services.AddCacheTagServices(options =>
            {
                if (isStage)
                {
                    options.InvalidateByMiddleware(@"^.*\/(__webpack.*|.+\.[a-zA-Z0-
9]+)$");//отсекаем левые запросы для статики (для каждого сайта может настраиваться
индивидуально)
                }
                else
                {
                    options.InvalidateByTimer(TimeSpan.FromSeconds(30));
                }
            });
```

5.3.7. Подключение и использование режима onscreen

Если сайт использует хотя бы один из кейсов «Подключение структуры сайта из QP» или «Подключение движка AB-тестирования к сайту», то к нему также можно подключить режим onscreen.

Onscreen – это особый режим редактирования сайта, когда пользователь может смотреть на stage-версию сайта и оттуда же управлять некоторыми его аспектами: виджетами (если подключена структура сайта из QP) и/или AB-тестами (если подключен движок AB-тестирования). Более подробно о функциональности этого режима.

Требования

Для возможности подключения Onscreen необходимо, чтобы был развернут специальный сервис **OnscreenAdmin**. Это консолидированный сервис, т.е. который подключается не к одной базе, а к целому семейству баз, прописанных в конфигурации customer-кодов QP. Нужно чтобы был развернут **OnscreenAdmin**, который подключен к той же QP, что и сайт.

Как использовать

1. Gодключить NuGet-пакет **QA.DotNetCore.Engine.OnScreen**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод **services.AddOnScreenIntegration**(mvcBuilder, options => …);
   mvcBuilder – это результат вызова `services.AddMvc()`; описание options для этой конфигурации описано в отдельном пункте. См. Опции конфгурации
3. Подключить middleware **app.UseOnScreenMode(customerCode)**, проверяющую возможность работы в режиме onscreen, добавляющая OnScreenContext в pipeline запроса.
4. Подключить к главному шаблону сайта (как правило `Views/Shared/_Layout.cshtml`) специальный viewcomponent, инициализирующий взаимодействие с OnscreenAdmin. `@await Component.InvokeAsync("OnScreen")`.

Как активируется этот режим

Чтобы активировать этот режим недостаточно открыть сайт в браузере. Этот режим доступен только пользователям QP, у которых есть доступ к редактированию этого сайта, для таких пользователей должен существовать с прописанным URL -адресом ***stage***-версии сайта.

При открытии **Custom Action** админка QP открывает фрейм с этим URL **Custom Action**, добавляя некоторые свои параметры в *query*-строку. По этим ***query***-параметрам производится авторизация пользователя для приложения **OnscreenAdmin**, именно из-под этого пользователя будут производиться все изменения в базе, которые можно сделать в этом режиме.

После первичной авторизации на сайте будет сохранена специальная cookie (её название определяется опцией **AuthCookieName**), которая позволит какое-то время открывать сайт в режиме Onscreen не во фрейме QP, а напрямую в браузере.

Опции конфигурации

При регистрации сервисов для интеграции Onscreen в `Startup.cs` вызывается метод `services.AddOnScreenIntegration(options => …)` с параметром **Action<OnScreenConfigurationOptions>.** Таким образом, здесь можно задать и переопределить опции, которые есть в объекте **OnScreenConfigurationOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.UseQpSettings

Обязательная	настройки для взаимодействия с QP (***SiteId, live/stage, CustomerCode***) с которой работает сайт.

options.AdminSiteBaseUrl

Обязательная	URL компонента **OnscreenAdmin**, который работает с тем же QP, что и сайт.

options.SiteId

Не нужно использовать	если уже задано через **options.UseQpSettings**. Id сайта в QP.

options.IsStage

Не нужно использовать	если уже задано через **options.UseQpSettings**. Режим работы сайта – ***Stage/Live***.

options.CustomerCode

Не нужно использовать	если уже задано через **options.UseQpSettings**. Customer-код сайта в QP.

options.AvailableFeatures

Обязательная	Значение по умолчанию - ***OnScreenFeatures.None*.** Фичи режима onscreen, которые нужно включить. Задаётся как набор флагов. Доступные значения флагов: • **Widgets**. фича OnScreen для работы с виджетами и зонами, имеет смысл только при использовании кейса «подключение структуры сайта из QP». • **AbTests**. фича OnScreen для работы с AB-тестами, имеет смысл только при использовании кейса «подключение движка AB-тестирования к сайту». Как правило, имеет смысл делать режим Onscreen активным только для ***stage***-версии сайта, как минимум потому, что у ***live***-версии могут быть более жесткие стратегии кэширования. В таких случаях можно задать эту опцию, например, так: `options.AvailableFeatures = qpSettings.IsStage ? OnScreenFeatures.Widgets | OnScreenFeatures.AbTests : OnScreenFeatures.None;`

options.AuthCookieName

Необязательная	Значение по умолчанию - ***"qa_onscreen_token"***. Имя cookie, в которой хранится информация об аутентификации onscreen.

options.AuthCookieLifetime

Необязательная	Значение по умолчанию - `TimeSpan.FromMinutes(60)`. Время жизни cookie, в которой хранится информация об аутентификации onscreen.

options.BackendSidQueryKey

Необязательная	значение по умолчанию - ***"backend_sid"***. Имя query-параметра с **backend_sid**, который создаёт QP для **custom action**, открытых через фрейм.

options.ApiApplicationNameInQp

Необязательная	значение по умолчанию - ***"onscreen-api"***. Под каким именем админка onscreen авторизуется в QP (должно совпадать с соответствующей настройкой у **OnscreenAdmin**).

options.OverrideAbTestStageModeCookieName

Необязательная	значение по умолчанию - ***"qa_onscreen_abtests_stage"***. Имя cookie, в которой хранится переопределенная для режима AB-тестов настройка **isStage**.

options.PageIdQueryParamName

Необязательная	значение по умолчанию - ***"pageId"***. Имя query-параметра с текущим **Id** страницы структуры сайта.

options.SkipWidgetTypes

Необязательная	значение по умолчанию - ***null***. Типы виджетов, которые надо игнорировать в режиме onscreen (не подсвечивать, обрамлять рамками итд). Массив из дискриминаторов (это поле ***Name*** у **QPDiscriminator**).

5.3.8. Подключение структуры сайта из XML

Альтернативный возможный кейс использования структуры сайта виджетной платформы: когда источник структуры сайта не QP, а XML-файл.

В реальных проектах этот кейс еще не был опробован. Причин для создания возможности получения страниц и виджетов не из QP, а из другого источника, в данном случае – XML-файл, несколько:

- Использование QP накладывает довольно большое ограничение на проект, теоретически может быть полезно использовать XML в качестве источника данных вместо базы, для каких-то подходящих кейсов.
- Продемонстрировать некий proof of concept, что концепция структуры сайта не завязана на использование QP, могут существовать и другие источники данных.

Структура XML

Пример того, как может выглядеть структура

```
<XmlRootPage>
	<XmlStartPage Alias="main">
		<XmlTextPage Alias="page1" Title="page1" Text="page1 content" />
		<XmlTextPage Alias="page2" Title="page2" Text="page2 content">
			<XmlTextPart	Alias="widget1"	Title="widget1"	Zone="Above"
Text="widget1 content"/>
		</XmlTextPage>
		<XmlTextPage Alias="page44" Title="page4" Text="page4 content" />
	</XmlStartPage>
</XmlRootPage>
```

Все правила, упоминающиеся в правилах контентного наполнения структуры сайта (см. кейс «Подключение структуры сайта из QP») актуальны и для наполнения XML.

Как использовать

1. Подключить NuGet-пакет **QA.DotNetCore.Engine.Xml.Configuration**.
2. Сконфигурировать **IServiceCollection** в `Startup.cs` вызвав метод `services.AddSiteStructureEngineViaXml(options => …)`; описание опций для этой конфигурации описано в отдельном пункте. См. Опции конфигурации
3. В `Startup.cs` использовать middleware, добавляющую структуру сайта в pipeline запроса **app.UseSiteStructure**();
4. Зарегистрировать route, который будет обрабатывать запросы к страницам структуры сайта. Сейчас поддерживается только старый вариант роутинга: до 3 версии ASP.NET Core, в 3 версии его можно активировать с помощью

   `services.AddMvc(o => o.EnableEndpointRouting = false)`.

   В методе **app.UseMvc** зарегистрировать `routes.MapContentRoute("default", "{controller}/{action=Index}/{id?}");`
5. Для всех типов страниц и виджетов создать классы моделей. Желательно заводить их все вместе в одной сборке для удобства регистрации типов в `services.AddSiteStructureEngineViaXml` (речь о `options.TypeFinder.RegisterFromAssemblyContaining`).

   Свой класс нужно завести также и для корневой и стартовой страницы. Классы страниц должны наследоваться от **XmlAbstractPage**, классы виджетов от. Называться классы должны также как соответствующие xml-ноды. Стартовая страница должна также

реализовывать интерфейс IStartPage **XmlAbstractWidget**. В этих классах можно создавать поля, которые соответствуют атрибутам, заведенным в XML для такого типа страницы. Это следует делать с помощью метода **GetDetail**. Например, `public string Text { get { return GetDetail("Text", String.Empty); } }`.

6. Остальные действия, такие как создание контроллеров, viewcomponent и т.д. описаны в инструкции по использованию кейса «Подключение структуры сайта из QP»

Опции конфигурации

При регистрации сервисов для структуры сайта в `Startup.cs` вызывается метод `services.AddSiteStructureEngineViaXml(options => …)` с параметром **Action<XmlSiteStructureEngineOptions>**. Таким образом мы можем здесь задать и переопределить опции, которые есть в объекте **XmlSiteStructureEngineOptions**.

Большую часть из них явно задавать не нужно, они уже имеют некоторые значения по умолчанию:

options.TypeFinder.RegisterFromAssemblyContaining

Обязательная	Регистрация сборки, которая содержит все классы страниц и виджетов, созданных разработчиком. Если они не будут зарегистрированы, то ни один узел дерева структуры сайта не будет инстанцирован. Пример: `options.TypeFinder.RegisterFromAssemblyContaining<RootPage, IAbstractItem>()` – читай как зарегистрируй все типы, реализующие **IAbstractItem**, находящиеся в той же сборке, где находится класс **RootPage**.

options.Settings.FilePath

Обязательная	Путь до xml-файла со структурой сайта.

options.ControllerMapperConvention

Необязательная	Значение по умолчанию – ***ControllerMapperConvention.Name*** Конвенция того, как .NET классам страниц соответствуют контроллеры. Описание опции такое же как у аналогичной опции в кейсе «Подключение структуры сайта из QP».

options.ComponentMapperConvention

Необязательная	Значение по умолчанию – ***ComponentMapperConvention.Name*** Конвенция того, как .NET классам виджетов соответствуют viewcomponent. Описание опции такое же как у аналогичной опции в кейсе «Подключение структуры сайта из QP».

5.4. QP8 API

Программный интерфейс доступа к QP реализован в виде .NET-сборки `Quantumart.dll`. Для использования API требуется установить NuGet-пакет `Quantumart.QP8.BLL`.

Примечание: файл сборки должен быть размещён в директории `bin` для веб-сайта.

Существует два основных сценария использования QP8 API:

Название сценария	Используемый класс
Вызовы QP8 API из сторонних веб-сайтов и приложений (включая веб-сайты, реализованные на классах LINQ to SQL)	DBConnector
Внимание: объекты QP для ветви «Шаблон» являются устаревшими (не рекомендуется использовать для разработки новых Систем; используются для сопровождения Систем, основанных на предыдущих версиях продукта). Вызовы QP8 API из страниц и элементов управления веб-сайта, реализованного на объектах QP	• QPage • QUserControl

В обоих случаях с помощью сборки также могут быть решены вспомогательные задачи следующего плана:

Название	Описание
Использование универсального механизма кэширования	Методы класса `DBConnector`: • `GetCachedEntity`, • `GetCachedData`, • `GetCachedFileContents`.
Аутентификация для пользовательских действий	Класс `QScreen`
Управление пользователями и группами пользователей QP	Класс `Permissions`
CRUD-функциональность (создание, чтение, изменение, удаление) для статей по логике, используемой в бекэнде. Используется для создания собственных панелей управления, а также для расширения QP с помощью пользовательских действий	Класс `ContentItem`
Внимание: объекты QP для ветви «Шаблон» являются устаревшими (не рекомендуется использовать для разработки новых Систем; используются для сопровождения Систем, основанных на предыдущих версиях продукта). Работа с объектом «Publishing Container».	Класс `QPublishControl`

5.4.1. Класс DBConnector

Конструкторы

```
public DBConnector(string strConnectionString)
```

В качестве значения параметра `strConnectionString` следует передавать строку подключения к БД.

```
public DBConnector()
```

Если строка подключения не задана, то значение берётся из статического свойства `ConnectionString`.

Основные свойства

Свойство ConnectionString

```
public static string ConnectionString { get; set; }
```

Строка подключения по умолчанию. Используется при создании экземпляра `DBConnector`, если в конструкторе строка подключения не была задана. Инициализируется строкой подключения `qp_database` из конфигурационного файла веб-сайта.

Свойство CustomConnectionString

```
public string CustomConnectionString { get; set; }
```

Строка подключения, специфичная для экземпляра класса.

Примечание: если строка подключения была указана в конструкторе, то указанное значение хранится здесь и может быть изменено.

Свойство IsStage

```
public bool IsStage { get; set; }
```

Определяет, в каком режиме (Live или Stage) работает экземпляр класса. Значение по умолчанию – `false`.

Примечание: если сайт сделан на объектах QP, то свойство инициализируется в соответствии с режимом, в котором был собран сайт. В противном случае учитывается значение параметра `isLive` конфигурационного файла веб-сайта. Также свойство может быть задано из кода.

Примечание: оказывает влияние на методы:

- `GetContentItemLinkQuery`,
- `GetContentItemLinkIds`,
- `SendNotification`,
- `GetMapFileContents`,
- `GetDefaultMapFileContents`.

Свойство CacheData

```
public bool CacheData { get; set; }
```

Определяет, должен ли экземпляр класса кэшировать получаемые из БД данные. Значение по умолчанию – `true`.

Примечание: интервалы кэширования задаются в конфигурационном файле веб-сайта. Данные могут кэшироваться как локально (в экземпляре), так и в общем кэше `HttpRuntime.Cache` в зависимости от доступности общего кэша и значения свойства `ForceLocalCache`.

Свойство ForceLocalCache

```
public bool ForceLocalCache { get; set; }
```

При значении `true` экземпляр класса использует собственный локальный кэш вместо общего кэша `HttpRuntime.Cache`.

Примечание: если общий кэш недоступен, то всегда будет использоваться локальный кэш.

Примечание: если `CacheData = false`, то значение данного свойства игнорируется.

Свойство UpdateManyToMany

```
public bool UpdateManyToMany { get; set; }
```

Определяет, требуется ли обновлять поля «Связь» типа M2M и M2O при создании и изменении статьи методом `AddFormToContent`. Значение по умолчанию – `true`.

Примечание: используется классами LINQ to SQL.

Методы для кэширования

Метод GetCachedEntity

```
public T GetCachedEntity<T>(string key, Func<string, T> fillAction) where T : class
public T GetCachedEntity<T>(string key, double interval, Func<string, T> fillAction)
where T : class
public T GetCachedEntity<T>(string key, Func<T> fillAction) where T : class
public T GetCachedEntity<T>(string key, double interval, Func<T> fillAction) where T
: class
```

Обобщённый метод кэширования, который может использоваться для кэширования объектов любых ссылочных типов (например, коллекций объектов, полученных на основе сгенерированных в QP классов LINQ to SQL). Также поддерживается кэширование `NULL`-значений.

На основе метода построены специализированные методы кэширования в сборке `Quantumart.dll`.

Примечание: в веб-среде возвращаемый объект является общим для нескольких потоков, поэтому его изменение может привести к непредсказуемым последствиям. Поэтому при необходимости модифицировать такие объекты рекомендуется синхронизировать доступ к ним. В то же время использование таких объектов для чтения в большинстве случаев потокобезопасно (предварительно рекомендуется изучить документацию на конкретный тип).

Примечание: для кэширования значимого типа (value type) необходимо выполнить принудительную упаковку (например, число следует преобразовать в строку).

Параметры:

Название	Описание
fillAction	Ссылка на метод заполнений кэша. При использовании версии `fillAction` с параметром туда передается параметр `key` метода `GetCachedEntity`, что позволяет использовать один и тот же метод заполнения для получения нескольких объектов.
interval	Интервал кэширования. **Примечание:** если значение не задано, то используется значение параметра `InternalExpirationTime` из конфигурационного файла веб-сайта.
key	Ключ, под которым данные хранятся в кэше. При использовании версии `fillAction` с параметром может использоваться для изменения логики метода заполнения.

Пример использования:

```
DBConnector cnn = new DBConnector();
List<Banner> banners = cnn.CacheManager.GetCachedEntity<List<Banner>>(pageAddress,
10, GetPageBanners);
```

В примере `GetPageBanners` – метод, возвращающий `List<Banner>`, который будет вызван только при заполнении кэша.

Метод GetCachedData

```
public DataTable GetCachedData(string queryString)
public DataTable GetCachedData(string queryString, double cacheInterval)
```

Выполняет указанный SQL-запрос и возвращает `DataTable` с сохранением результата в кэше.

Продолжительность кэширования определяется значением параметра `cacheInterval`. При отсутствии значения используется значение параметра `InternalExpirationTime` из конфигурационного файла веб-сайта.

Внимание: в веб-среде возвращаемая таблица является общей для нескольких потоков (пользовательских сессий), поэтому её изменение может привести к непредсказуемым последствиям в других потоках. Например, нельзя менять свойство `RowFilter` у DefaultView. Вместо этого нужно создать новый DataView на основе DataTable и менять свойство `RowFilter` у него.

Ключ кэширования строится на основании текста запроса. К ключу добавляется префикс, в который входят имя БД и имя сервера БД, что позволяет работать с несколькими экземплярами класса в одной веб-среде.

Примечание: при необходимости выполнить запрос без кэширования следует использовать метод GetRealData.

Метод GetCachedFileContents

```
public string GetCachedFileContents(string path)
```

Считывает текстовый файл по заданному пути и сохраняет его в кэше.

Внимание: при изменении файла кэш сбрасывается.

Методы для чтения данных статей

Метод GetContentData

```
public DataTable GetContentData(string siteName, string contentName, string fields,
string whereExpression, string orderExpression, long startRow, long pageSize, ref
long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, bool cacheResult, double cacheInterval, bool useClientSelection,
bool withReset)

public DataTable GetContentData(string siteName, string contentName, string fields,
string whereExpression, string orderExpression, long startRow, long pageSize, ref
long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, bool cacheResult, double cacheInterval)

public DataTable GetContentData(string siteName, string contentName, string
whereExpression, string orderExpression, long startRow, long pageSize, ref long
totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte
includeArchive)

public DataTable GetContentData(string siteName, string contentName, string fields,
string whereExpression, string orderExpression, long startRow, long pageSize, ref
long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive)
```

Метод является упрощённым вариантом получения данных из контента (вместо более сложного способа для объекта «Publishing Container»).

Внимание: метод не выполняет автоматическую замену заполнителей, поэтому при использовании текстовых полей, которые могут содержать URL, нужно вызывать метод `FormatField`.

Примечание: метод возвращает из кэша не саму таблицу, а ее копию, так что её можно изменять.

Примечание: для поля «Связь» типов M2M и M2O метод возвращает данные, которые хранятся в таблице контента (идентификатор связи для M2M и идентификатор базового поля для M2O). Используя эти данные, можно получить реальные данные связей статьи вызовом метода `GetContentItemLinkIDs`.

Обязательные параметры:

Название	Описание
siteName	Имя сайта.

contentName	Имя контента.
whereExpression	Пользовательская часть предложения `WHERE` SQL-запроса. **Примечание:** аналог поля «Фильтр» (Filter) для объекта «Publishing Container».
orderExpression	Предложение `ORDER` SQL-запроса. **Примечание:** аналог поля «Выражение динамической сортировки» (Dynamic order expression) для объекта «Publishing Container».
startRow	Параметр постраничного вывода. **Примечание:** аналог поля «Номер первой отображаемой записи» (First article number) для объекта «Publishing Container».
pageSize	Параметр постраничного вывода. **Примечание:** аналог поля «Количество» (Count) для объекта «Publishing Container».
totalRecords	Количество записей с учётом фильтрации, но без учёта постраничного вывода.
useSchedule	Дополнительная фильтрация статей, для которых выставлено расписание. **Примечание:** аналог поля «Использовать расписание статей» (Use Articles Schedule) для объекта «Publishing Container».
statusName	Имя статуса. **Примечание:** аналог поля «Статус» (Status) для объекта «Publishing Container».
showSplittedArticle	Указывает, какую версию статьи возвращать. При значении `0` возвращается опубликованная версия статьи (Live-режим), иначе – текущая (Stage-режим).
includeArchive	Определяет, включать ли архивные статьи в результат. **Примечание:** аналог поля «Показать архивированные» (Show archived) для объекта «Publishing Container».

Необязательные параметры:

Название	Описание
fields	Набор полей, которые необходимо вернуть. При отсутствии значения возвращаются все поля. **Примечание:** по умолчанию служебные поля не попадают в результат, при необходимости их нужно указывать.
cacheResult	Указатель, следует ли кэшировать результат. Значение по умолчанию – `false`.
cacheInterval	Продолжительность кэширования. **Примечание:** по умолчанию используется значение параметра `InternalExpirationTime` из конфигурации веб-сайта.
useClientSelection	При значении `true` фильтрация, сортировка и постраничный вывод осуществляются на стороне клиента, что позволяет уменьшить объём кэшируемых данных. **Примечание:** фильтрация на стороне клиента поддерживает только часть возможностей серверной. Значение по умолчанию – `false`.
withReset	При значении `true` для вызова осуществляется принудительный сброс кэша и его повторное заполнение. Значение по умолчанию – `false`.

Пример:

```
long total = 0;
DataTable dt = cnn.GetContentData("Sandbox NET", "Events", "[content_item_id],
```

```
[Title]", "[Location] = 1662", "[Modified] DESC", 1, 20, ref total, 1, "Published",
0, 0);
```

Метод GetCachedContentData

```
public DataTable GetCachedContentData(string siteName, string contentName, string
whereExpression, string orderExpression, long startRow, long pageSize, ref long
totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte
includeArchive)
public DataTable GetCachedContentData(string siteName, string contentName, string
whereExpression, string orderExpression, long startRow, long pageSize, ref long
totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte
includeArchive, bool useClientSelection)
public DataTable GetCachedContentData(string siteName, string contentName, string
whereExpression, string orderExpression, long startRow, long pageSize, ref long
totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte
includeArchive, double cacheInterval)
public DataTable GetCachedContentData(string siteName, string contentName, string
whereExpression, string orderExpression, long startRow, long pageSize, ref long
totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte
includeArchive, double cacheInterval, bool useClientSelection)
public DataTable GetCachedContentData(string siteName, string contentName, string
fields, string whereExpression, string orderExpression, long startRow, long pageSize,
ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive)
public DataTable GetCachedContentData(string siteName, string contentName, string
fields, string whereExpression, string orderExpression, long startRow, long pageSize,
ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, bool useClientSelection)
public DataTable GetCachedContentData(string siteName, string contentName, string
fields, string whereExpression, string orderExpression, long startRow, long pageSize,
ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, double cacheInterval)
public DataTable GetCachedContentData(string siteName, string contentName, string
fields, string whereExpression, string orderExpression, long startRow, long pageSize,
ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, double cacheInterval, bool useClientSelection)
public DataTable GetCachedContentData(string siteName, string contentName, string
fields, string whereExpression, string orderExpression, long startRow, long pageSize,
ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle,
byte includeArchive, bool useClientSelection, bool withReset)
```

Аналог метода `GetContentData`, кэширующий результат во всех случаях.

Внимание: исключением является ситуация, когда кэширование отключено на уровне экземпляра класса.

Метод GetContentItemLinkIds

```
public string GetContentItemLinkIDs(string linkFieldName, long itemId)
public string GetContentItemLinkIDs(string linkFieldName, string itemIds)
public string GetContentItemLinkIDs(int linkId, long itemId)
public string GetContentItemLinkIDs(int linkId, string itemIds)
public string GetContentItemLinkIDs(int linkId, long itemId, bool isManyToMany)
public string GetContentItemLinkIDs(int linkId, string itemIds, bool isManyToMany)
```

Выводит данные для поля «Связь» типов M2M и M2O.

Параметры:

Название	Описание
linkFieldName	Имя поля «Связь».
itemId	Идентификатор статьи, для которой требуется получить данные.
itemIds	Идентификаторы статей, для которых требуется получить данные. **Примечание:** в качестве значения задавать идентификаторы через запятую.
linkId	Для типа M2M – идентификатор связи (запись в таблице `CONTENT_TO_CONTENT`). Для типа M2O – идентификатор базового поля «Связь» типа O2M (запись в таблице `CONTENT_ATTRIBUTE`). **Примечание:** идентификаторы хранятся в полях таблицы контента (`CONTENT_`*NNN*) и могут быть получены методом `GetContentData`, что позволяет избежать дополнительных запросов к БД.
isManyToMany	Указатель, что используется поле «Связь» типа M2M. Значение по умолчанию – `true`.

Результат возвращается в виде строки идентификаторов связанных статей через запятую.

Внимание: в результат всегда добавляется `0`, так что он никогда не является пустым и может быть использован в SQL-конструкции `IN` без дополнительного кода.

Примечание: учитываются значения свойств `IsStage` и `CacheData`. Если `IsStage = true`, то возвращаются текущие версии полей, иначе только опубликованные. Если `CacheData = true`, то результат выполнения кэшируется на интервал времени `InternalShortExpirationTime`.

Метод GetRealContentItemLinkIds

```
public string GetRealContentItemLinkIDs(string linkFieldName, long itemId)
public string GetRealContentItemLinkIDs(string linkFieldName, string itemIds)
public string GetRealContentItemLinkIDs(int linkId, long itemId)
public string GetRealContentItemLinkIDs(int linkId, string itemIds)
public string GetRealContentItemLinkIDs(int linkId, long itemId, bool isManyToMany)
public string GetRealContentItemLinkIDs(int linkId, string itemIds, bool
isManyToMany)
```

Метод аналогичен методу `GetContentItemLinkIds`. Отличие в том, что результат выполнения не кэшируется вне зависимости от значения свойства `CacheData`.

Метод GetContentItemLinkQuery

```
public string GetContentItemLinkQuery(string linkFieldName, long itemId)
public string GetContentItemLinkQuery(string linkFieldName, string itemIds)
public string GetContentItemLinkQuery(int linkId, long itemId)
public string GetContentItemLinkQuery(int linkId, string itemIds)
public string GetContentItemLinkQuery(int linkId, long itemId, bool isManyToMany)
public string GetContentItemLinkQuery(int linkId, string itemIds, bool isManyToMany)
```

Метод аналогичен методу `GetContentItemLinkIds`. Отличие в том, что он возвращает не результат запроса к БД, а сам SQL-запрос.

Методы для создания, изменения, удаления статей

Примечание: более удобной альтернативой являются методы класса `ContentItem`.

Метод AddFormToContent

```
public int AddFormToContent(int site_id, string content_name, string status_name, ref Hashtable Values, ref HttpFileCollection Files, int content_item_id, bool updateEmpty)

public int AddFormToContent(int site_id, string content_name, string status_name, ref Hashtable Values, ref HttpFileCollection Files, int content_item_id)

public int AddFormToContent(int site_id, string content_name, string status_name, ref Hashtable Values, ref HttpFileCollection Files)

public int AddFormToContent(int site_id, string content_name, string status_name, ref Hashtable Values, int content_item_id)

public int AddFormToContent(int actualSiteId, int content_id, string status_name, ref Hashtable Values, ref HttpFileCollection Files, int content_item_id, bool updateEmpty, int attribute_id)

public int AddFormToContent(int actualSiteId, int content_id, string status_name, ref Hashtable Values, ref HttpFileCollection Files, int content_item_id, bool updateEmpty, int attribute_id, bool visible, bool archive, bool returnModified, ref DateTime modified)
```

Создаёт новые статьи, изменяет существующие.

Поддерживается полноценная работа с расщеплёнными статьями, включая поля «Связь» типов M2M и M2O. Значения таких полей должны представлять собой строку идентификаторов связанных статей, разделённых запятыми.

Внимание: метод не поддерживает отправку уведомлений. Необходимо использовать метод `SendNotification` или использовать класс `ContentItem`.

Производятся следующие проверки на сохраняемые поля:

- тип поля («Числовой», «Дата»),
- обязательность,
- уникальность,
- допустимая длина и маска ввода (для поля типа «Строка»).

Дополнительно:

- при обновлении статьи создается её версия,
- вся работа с БД происходит в одной транзакции.

Параметры:

Метод	Метод
site_id	Идентификатор сайта, с которого производится вызов.
actualSiteId	Фактический идентификатор сайта, в который добавляется статья.
contentName	Имя контента, в который добавляется статья. **Внимание:** контент не может быть виртуальным. При указании виртуального контента будет выдано исключение. **Примечание:** поддерживается имя контента в формате *SiteName*.*ContentName*. Если сайт *SiteName* будет найден, то его идентификатор будет использован в качестве значения параметра `actualSiteId`, иначе будет использован параметр `site_id`.
content_id	Идентификатор контента, в который добавляется статья.

	Внимание: контент не может быть виртуальным. При указании виртуального контента будет выдано исключение.
statusName	Имя статуса Workflow, который должен быть присвоен статье.
Values	Значения полей статьи в коллекции `Values`, ключи которой должны быть сформированы с помощью метода `FieldName`.
Files	Коллекция файлов, переданных вместе с веб-запросом. Для веб-формы её содержимое надо взять из `Request.Files`, иначе передать переменную со значением `NULL`.
content_item_id	Идентификатор статьи. При ненулевом значении метод работает в режиме обновления существующей статьи, при нулевом – в режиме добавления новой статьи. **Примечание:** значение по умолчанию – `0`.
updateEmpty	Указатель, следует ли обнулять поля, которые в коллекции `Values`: 1) имеют пустые значения, 2) отсутствуют. Значение по умолчанию – `true`, то есть при обновлении статьи в коллекции `Values` должны быть представлены значения всех полей. Если нужно обновить одно или несколько полей и задать в коллекции `Values` только их, то для параметра следует задать значение `false`.
attribute_id	Идентификатор поля. При ненулевом значении выполняется изменение указанного поля, при нулевом – обычный режим обновления. **Примечание:** значение по умолчанию – `0`.
visible	Значение флага видимости для статьи. **Примечание:** значение параметра не меняется в тех версиях метода, где он не передаётся. **Примечание:** используется в классах LINQ to SQL.
archive	Значение флага архивации для статьи. **Примечание:** значение параметра не меняется в тех версиях метода, где он не передаётся. **Примечание:** используется в классах LINQ to SQL.
returnModified	Указатель, что требуется вернуть дату модификации статьи в параметре `modified`. **Примечание:** используется в классах LINQ to SQL.
modified	Дата модификации статьи. Возвращается в случае `returnModified = true`. **Примечание:** используется в классах LINQ to SQL.

Примечание: в директорию контента сохраняются только те файлы из коллекции Files, которые корректно представлены в коллекции `Values` (сформированы методом `FieldName`). Если имя нового файла совпадает с именем уже существующего файла, то в имя нового файла будет добавлен числовой индекс (например, `image[1].jpg`), при этом изменение имени файла будет учтено в коллекции `Values`.

Примечание: вне зависимости от заполнения коллекции `Files` для статьи будут сгенерированы (перегенерированы) все динамические изображения.

Примечание: в случае невозможности предоставить коллекцию `Files` файлы необходимо сохранять вручную, используя для определения целевого пути метод `GetUrlForFileAttribute`.

Метод UpdateContentItem

```
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, int
content_item_id)
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, int
```

```
content_item_id, bool updateEmpty)
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, int
content_item_id, bool updateEmpty, string statusName)
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, ref
HttpFileCollection Files, int content_item_id)
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, ref
HttpFileCollection Files, int content_item_id, bool updateEmpty)
public int UpdateContentItem(int site_id, int content_id, ref Hashtable Values, ref
HttpFileCollection Files, int content_item_id, bool updateEmpty, string statusName)
```

Аналогичен методу `AddFormToContent` в режиме изменения существующей статьи.

Метод UpdateContentItemField

```
public void UpdateContentItemField(int site_id, string content_name, string
field_name, int content_item_id, ref Hashtable Values, ref HttpFileCollection Files)
public void UpdateContentItemField(int site_id, string content_name, string
field_name, int content_item_id, ref Hashtable Values)
```

Аналогичен методу `AddFormToContent` в режиме изменения одного поля (параметр `field_name`).

Метод DeleteContentItem

```
public void DeleteContentItem(int content_item_id)
```

Удаляет существующую статью.

Примечание: метод не поддерживает отправку уведомлений. Необходимо использовать метод `SendNotification` или использовать класс `ContentItem`.

Методы для массового создания и изменения статей

Метод MassUpdate

Массово создаёт новые статьи, изменяет существующие.

Примечание:

1. Используется при создании и изменении статей с использованием LINQ to SQL и EF. В LINQ to SQL вызывается для каждой статьи, в EF – для группы статей контента.
2. Изменяются те поля статей, названия которых явно передаются в виде ключей коллекции.
3. При расщеплении статьи, метод копирует все поля. После обновляет только переданные.

```
public void MassUpdate(int contentId, IEnumerable<Dictionary<string, string>> values,
int lastModifiedBy)
public void MassUpdate(int contentId, IEnumerable<Dictionary<string, string>> values,
int lastModifiedBy, MassUpdateOptions options)
```

```
    public class MassUpdateOptions
    {
        public MassUpdateOptions()
        {
            CreateVersions = true;
            ReturnModified = true;
            ReplaceUrls = true;
```

```
        }

        public bool CreateVersions { get; set; }

        public bool ReturnModified { get; set; }

        public bool ReplaceUrls { get; set; }
    }
```

Отличия `MassUpdate` от `ImportToContent`:

1) создаётся версия (опция `CreateVersions`),
2) возвращается дата модификации статьи (опция `ReturnModified`),
3) заменяются URL (опция `ReplaceUrls`),
4) создаются динамические изображения,
5) создаются версии файлов для полей типа «Файл» и «Изображение»,
6) выполняется базовая валидация полей.

Метод ImportToContent

```
public void ImportToContent(int contentId, IEnumerable<Dictionary<string, string>>
values, int lastModifiedBy = 1, int[] attrIds = null)
```

По умолчанию можно рассматривать как метод `MassUpdate`, массово создающий новые статьи и изменяющий существующие.

Если указан параметр `attrIds`, то метод обновит значения указанных полей, даже если они отсутствуют.

Если поле для одной статьи передано в виде ключа коллекции, а для второй статьи не задано, то такое поле будет обнулено для второй статьи.

Примечание: можно использовать для импорта данных в QP.

Параметры:

Название	Описание
contentId	Идентификатор контента, в который осуществляется импорт.
values	Импортируемые в контент данные. **Примечание:** Каждая коллекция соответствует статье контента. Ключ коллекции – имя поля (поддерживаются как системные, так и пользовательские поля), значение в коллекции – значение соответствующего ключу поля статьи в виде строки.
lastModifiedBy	Идентификатор пользователя, от имени которого выполняется импорт.
attrIds	Ограничитель набора полей, в которых необходимо выполнить изменения. В качестве значения следует передавать массив идентификаторов полей контента.

Внимание: ограничения метода:

- не поддерживается поле «Связь» типа M2O,
- не создаются версии статьи,
- не генерируются объекты «Динамическое изображение»,

- не формируются уведомления,
- не фиксируется аудит,
- не производится валидация форм.

Примечание: Поддерживается режим расщепления статей для Workflow. Поддерживается поле «Связь» типа M2M в виде списка идентификаторов полей через запятую.

Методы для выполнения произвольных SQL-запросов

Метод GetRealData

```
public DataTable GetRealData(string queryString)
```

Выводит данные из БД по SQL-запросу `queryString` в виде `DataTable`.

Примечание: метод не использует кэш. Если необходимо использовать кэш, то рекомендуется метод `GetCachedData`.

Метод GetData

```
public DataTable GetData(string queryString)
```

Аналогичен методу `GetRealData`. Отличие в том, что поведение метода зависит от значения параметра `CacheGetData` конфигурационного файла веб-сайта. При значении `1` метод работает, как `GetCachedData` с интервалом кэширования по умолчанию, в противном случае – как `GetRealData`.

Примечание: параметр `CacheGetData` нужен для быстрого решения проблем с большим количеством SQL-запросов на старых веб-сайтах. При написании нового кода для предсказуемости поведения Системы рекомендуется использовать методы `GetCachedData` и `GetRealData`.

Метод ProcessData

```
public void ProcessData(string queryString)
```

Формирует к БД UPDATE- и DELETE-запросы `queryString`.

Методы для работы с файлом отображения

Метод GetMapFileContents

```
public string GetMapFileContents(int site_id, string fileName)
```

Ищет MAP-файл и выводит его содержимое в виде строки.

Примечание: результат кэшируется до первого изменения с помощью метода `GetCachedFileContents`.

Параметры:

Название	Описание
site_id	Задаёт сайт в рамках текущей БД. Для сайта определяется путь к директории `App_Data`. **Примечание:** для определения требуемого режима работы веб-сайта (Live или Stage) проверяется свойство `IsStage`.
fileName	Определяет имя MAP-файла. Файл ищется в директории `App_Data` сайта.

Метод GetDefaultMapFileContents

Внимание: метод доступен в NuGet-пакете `Quantumart` версии 2.1.0 и выше (см. Установка NuGet-пакетов).

```
public string GetDefaultMapFileContents(int site_id)

public string GetDefaultMapFileContents(int siteId, string contextName)
```

Метод аналогичен методу `GetMapFileContents`. Позволяет получать MAP-файл программно при наличии Connection String, Id сайта и имени контекстного класса (без учёта регистра).

Примечание: имя файла для основного контекстного класса может быть определено в свойствах сайта (значение по умолчанию – `QPDataContext`).

Пример использования:

```
var dbc = new DBConnector(Global.ConnectionString);
var map1 = dbc.GetDefaultMapFileContents(Global.SiteId, "qpcontext");
dbc.IsStage = true;
var map2 = dbc.GetDefaultMapFileContents(Global.SiteId, "qpcontext");
```

По умолчанию в директории `App_Data` сайта ищется уже сгенерированный MAP-файл (для обратной совместимости). Если файл не найден, то выполняется программная генерация MAP-файла (без создания промежуточных файлов на диске). Результат возвращается в виде строки.

Примечание: в старых версиях пакета `Quantumart` при отсутствии файла выдавалось исключение.

Методы для получения физических и виртуальных путей

Примечание: если для экземпляра класса включено кэширование, то, если не указано иное, результаты всех методов кэшируются на время, заданное в конфигурационном файле веб-сайта.

Метод GetImagesUploadUrl

```
public string GetImagesUploadUrl(int site_id)
public string GetImagesUploadUrl(int site_id, bool asShortAsPossible)
```

Возвращает URI Библиотеки сайта.

Параметры:

Название	Описание
site_id	Идентификатор сайта
asShortAsPossible	Указатель, требуется ли вернуть результат в виде URL. Значение по умолчанию – `false`. **Примечание:** если для сайта активировано свойство «Использовать абсолютный виртуальный путь», то к результату добавляется значение свойства «Префикс виртуального пути».

Метод GetSiteLibraryDirectory

```
public string GetSiteLibraryDirectory(int site_id)
```

Возвращает физический путь к Библиотеке сайта.

Метод GetActualSiteUrl

```
public string GetActualSiteUrl(int site_id)
```

Возвращает URL корневой директории страниц сайта.

Примечание: учитывается режим работы сайта.

Метод GetSiteUrl

```
public string GetSiteUrl(int site_id, bool isLive)
```

Возвращает URL корневой директории страниц сайта. Значение параметра `isLive` определяет требуемый режим работы сайта.

Метод GetSiteUrlRel

```
public string GetSiteUrlRel(int site_id, bool isLive)
```

Возвращает URI корневой директории страниц сайта. Значение параметра `isLive` определяет требуемый режим работы сайта.

Метод GetSiteDirectory

```
public string GetSiteDirectory(int site_id, bool isLive)
```

Возвращает физический путь корневой директории страниц сайта. Значение параметра `isLive` определяет требуемый режим работы сайта.

Метод GetContentUploadUrl

```
public string GetContentUploadUrl(int site_id, string content_name)
```

Возвращает URL Библиотеки контента для указанного контента.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Метод GetContentLibraryDirectory

```
public string GetContentLibraryDirectory(int site_id, int content_id)
```

Возвращает физический путь к Библиотеке контента для указанного контента.

Метод GetDirectoryForFileAttribute

```
public string GetDirectoryForFileAttribute(int attrId)
```

Возвращает физический путь к корневой директории указанного поля.

Примечание: учитывает настройки поля «Использовать библиотеку сайта» и «Подпапка для файлов».

Метод GetUrlForFileAttribute

```
public string GetUrlForFileAttribute(int fieldId)
public string GetUrlForFileAttribute(int fieldId, bool asShortAsPossible)
```

Возвращает URI корневой директории поля.

Параметры:

Название	Описание
fieldId	Идентификатор поля.
asShortAsPossible	Указатель, требуется ли вернуть результат в виде URL. Значение по умолчанию – `true`. **Примечание:** если для сайта активировано свойство «Использовать абсолютный виртуальный путь» к результату добавляется значение свойства «Префикс виртуального пути».

Примечание: учитывает настройки поля «Использовать библиотеку сайта» и «Подпапка для файлов», а также директории для динамических изображений.

Методы для работы с уведомлениями

Метод SendNotification

```
public void SendNotification(int content_item_id, string notification_on)
```

Инициирует механизм формирования уведомления для указанной статьи.

Примечание: если для сайта не задан флаг «Собирать страницы для предварительного просмотра и уведомлений в Основном режиме», то на работу метода влияет параметр `IsStage` (определяет, для какого режима должны собираться уведомления).

Вспомогательные методы

Примечание: если для экземпляра класса включено кэширование, то, если не указано иное, результаты всех методов кэшируются на время, заданное в конфигурационном файле веб-сайта.

Метод FieldID

```
public int FieldID(int site_id, string content_name, string field_name)
```

Возвращает идентификатор указанного поля.

Параметры:

Название	Описание
site_id	Идентификатор сайта
content_name	Имя контента. **Примечание:** поддерживается имя контента в формате *SiteName*.*ContentName*.
field_name	Имя поля

Метод FieldName

```
public string FieldName(int site_id, string content_name, string field_name)
```

Возвращает внутреннее имя поля.

Примечание: используется при формировании ключей коллекции `Values` для метода `AddFormToContent`.

Параметры:

Название	Описание
site_id	Идентификатор сайта
content_name	Имя контента. **Примечание:** поддерживается имя контента в формате *SiteName*.*ContentName*.
field_name	Имя поля

Метод FormatField

```
public string FormatField(string key, int site_id)
public string FormatField(string key, int site_id, isLive)
```

Выполняет замену заполнителей на URI. Если параметр `isLive` не задан, то используется инвертированное значение свойства `IsStage`.

Замена осуществляется на следующие URI из следующих параметров:

- `GetImagesUploadUrl`,
- `GetSiteUrl` или `GetSiteUrlRel` (в зависимости от значения параметра `UseAbsoluteSiteUrl`).

Примечание: следует применять метод при чтении данных статей во всех случаях, кроме использования классов LINQ to SQL и метода `Field`, в которых имеется идентичная функциональная возможность.

Метод GetSiteId

```
public int GetSiteId(string name)
```

Выводит идентификатор сайта по указанному имени сайта.

Метод GetSiteIdByContentId

```
public Int32 GetSiteIdByContentId(int contentId)
```

Выводит идентификатор сайта по указанному идентификатору контента.

Метод GetSiteName

```
public string GetSiteName(int site_id)
```

Выводит имя сайта по указанному идентификатору сайта.

Метод GetContentId

```
public int GetContentId(int siteId, string contentName)
```

Выводит идентификатор контента по указанным идентификатору сайта и имени контента.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Метод GetContentIdForAttribute

```
public int GetContentIdForAttribute(int id)
```

Выводит идентификатор контента по указанному идентификатору поля.

Метод GetContentIdForItem

```
public int GetContentIdForItem(int ItemID)
```

Выводит идентификатор контента по указанному идентификатору статьи.

Метод GetContentName

```
public string GetContentName(int contentId)
```

Выводит имя контента по указанному идентификатору контента.

Метод GetContentFieldValue

```
public string GetContentFieldValue(int itemID, string fieldName)
```

Выводит значение поля в виде строки.

Параметры:

Название	Описание
itemID	Идентификатор статьи
fieldName	Имя поля

В качестве результата возвращается пустая строка, если:

1) статья не существует,
2) поле не существует,
3) поле содержит значение `NULL`.

Примечание: на результат влияет значение свойства `IsStage`. При `IsStage = true` возвращаются данные текущей статьи, иначе – данные опубликованной.

Внешние транзакции

Поддерживается возможность выполнить несколько операций в БД (например, несколько различных вызовов метода `MassUpdate` для разных контентов) в рамках единой транзакции.

Для этого следует:

1) создать SQL-соединение с БД;

2) открыть созданное соединение;
3) начать транзакцию на открытом соединении;
4) создать экземпляр `DBConnector`;
   Экземпляру в качестве параметров должны быть переданы данные по:
   a. открытому соединению,
   b. транзакции.
5) выполнить необходимые операции на созданном соединении.

Пример:

```
            using (var conn = new SqlConnection(Global.ConnectionString))
            {
                conn.Open();
                var tr = conn.BeginTransaction();
                var localCnn = new DBConnector(conn, tr);

                try
                {
                    localCnn.MassUpdate(BaseContentId, values,
Global.LastModifiedId);
                    localCnn.MassUpdate(ContentId, values2, Global.LastModifiedId);
                    tr.Commit();
                }
                catch (Exception)
                {
                    tr.Rollback();
                }
            }
```

5.4.2. Класс QScreen

Метод CheckCustomTabAuthentication

```
public static bool CheckCustomTabAuthentication()
```

Используется для аутентификации при использовании пользовательских действий. В случае успешной аутентификации возвращает `true`.

Следует использовать для аутентификации при каждой перезагрузке страницы для пользовательского действия.

Внимание: на веб-сайте должны быть включены сессии. Если это требование невыполнимо, то следует использовать метод `AuthenticateForCustomTab`.

Метод AuthenticateForCustomTab

```
public static int AuthenticateForCustomTab()
```

Используется для аутентификации при использовании пользовательских действий. В случае успешной аутентификации возвращает идентификатор пользователя, иначе – `0`.

Следует использовать при первом вызове пользовательского действия. При перезагрузках страниц пользовательского действия метод использовать нельзя, так как аутентификация возможна только при первом вызове, поэтому результат аутентификации нужно сохранять. Если есть возможность хранить его в сессии, то следует использовать метод `CheckCustomTabAuthentication`.

5.4.3. Класс Permissions

Класс предоставляет возможности управления:

1) пользователями,
2) группами пользователей,
3) правами доступа.

Метод AddChildGroupToParentGroup

```
public void AddChildGroupToParentGroup(int parent_group_id, int child_group_id)
```

Задаёт для группы с идентификатором `child_group_id` родительскую группу с идентификатором `parent_group_id`.

Метод AddGroup

```
public int AddGroup(string name)
public int AddGroup(string name, bool AllowSharedOwnershipOfItem)
```

Создаёт новую группу. Результат содержит идентификатор группы.

Параметры:

Название	Описание
name	Имя для новой группы
`AllowSharedOwnershipOfItem`	Указатель, что допускается использование опции «Совместное использование статей» в свойствах группы. Значение по умолчанию – `false`.

Метод AddGroupToContentPermission

```
public void AddGroupToContentPermission(int groupId, int contentId, int permissionId)
public void AddGroupToContentPermission(int groupId, int contentId, int permissionId,
bool propagateToItems)
```

Задаёт или изменяет права доступа для группы на указанный контент.

Параметры:

Название	Описание
groupId	Идентификатор группы
contentId	Идентификатор контента
permissionId	Права доступа
propagateToItems	Указатель, что права доступа должны будут применяться к новым статьям, создаваемым в контенте

Метод AddGroupToItemPermission

```
public void AddGroupToItemPermission(int groupId, int itemId, int permissionId)
```

Задаёт или изменяет права доступа для группы на указанную статью.

Параметры:

Название	Описание
groupId	Идентификатор группы

itemId	Идентификатор статьи
permissionId	Права доступа

Метод AddUser

```
public int AddUser(string username, string password, string First_Name, string Last_Name, string Email)
```

Создаёт нового пользователя. Результат содержит идентификатор пользователя.

Примечание: пользователь, создаваемый таким образом, по умолчанию не имеет доступа в бекэнд (активно свойство «Заблокирован» (Disabled)).

Параметры:

Название	Описание
username	Имя пользователя
password	Пароль пользователя
First_Name	Имя персоны, для которой создаётся пользователь
Last_Name	Фамилия персоны, для которой создаётся пользователь
Email	Адрес электронной почты персоны, для которой создаётся пользователь

Метод AddUserToContentPermission

```
public void AddUserToContentPermission(int userId, int contentId, int permissionId)
public void AddUserToContentPermission(int userId, int contentId, int permissionId, bool propagateToItems)
```

Задаёт или изменяет права доступа для пользователя на указанный контент.

Параметры:

Название	Описание
userId	Идентификатор группы
contentId	Идентификатор контента
permissionId	Права доступа
propagateToItems	Указатель, что права доступа должны будут применяться к новым статьям, создаваемым в контенте

Метод AddUserToGroup

```
public void AddUserToGroup(int userId, int groupId)
```

Добавляет пользователя в указанную группу.

Метод AddUserToItemPermission

```
public void AddUserToItemPermission(int userId, int itemId, int permissionId)
```

Задаёт или изменяет права доступа для пользователя на указанную статью.

Метод AuthenticateUser

```
public int AuthenticateUser(string username, string password)
```

Производит попытку аутентификации пользователя. При успешной аутентификации возвращает идентификатор пользователя, при неуспешной – `0`.

Параметры:

Название	Описание
username	Имя пользователя
password	Пароль пользователя

Метод CopyUsersFromGroupToGroup

```
public void CopyUsersFromGroupToGroup(int fromGroupId, int toGroupId)
```

Копирует пользователей из группы `fromGroupId` в группу `toGroupId`.

Внимание: копируются только отсутствующие в группе `toGroupId` пользователи.

Метод GetAllGroups

```
public DataTable GetAllGroups()
```

Возвращает все группы пользователей. Возвращаемые поля:

1) `group_id`,
2) `group_name`,
3) `created`,
4) `modified`.

Метод GetAllGroupsForContentPermission

```
public DataTable GetAllGroupsForContentPermission(int contentId)
```

Возвращает данные по всем группам пользователей, для которых явно настроены права доступа на указанный контент. Возвращаемые поля:

1) `group_id`,
2) `group_name`,
3) `created`,
4) `modified`.

Метод GetAllGroupsForItemPermission

```
public DataTable GetAllGroupsForItemPermission(int itemId)
```

Возвращает данные по всем группам пользователей, для которых явно настроены права доступа на указанную статью. Возвращаемые поля:

1) `group_id`,
2) `group_name`,
3) `created`,
4) `modified`.

Метод GetAllUsersForItemPermission

```
public DataTable GetAllUsersForItemPermission(int itemId)
```

Возвращает данные всех пользователей, для которых явно настроены права доступа на указанную статью. Возвращаемые поля:

1) `user_id`,
2) `login`,
3) `password`,
4) `disabled`,
5) `first_name`,

6) `last_name`,
7) `email`,
8) `created`,
9) `modified`.

Метод GetAllUsersForContentPermission

```
public DataTable GetAllUsersForContentPermission(int contentId)
```

Возвращает данные всех пользователей, для которых явно настроены права доступа на заданный контент. Возвращаемые поля:

1) `user_id`,
2) `login`,
3) `password`,
4) `disabled`,
5) `first_name`,
6) `last_name`,
7) `email`,
8) `created`,
9) `modified`.

Метод GetChildParentGroups

```
public DataTable GetChildParentGroups()
```

Возвращает список групп с данными о родительской группе. Возвращаемые поля:

1) `child_group_id`,
2) `child_group_name`,
3) `parent_group_id`,
4) `parent_group_name`.

Метод GetGroupInfo

```
public DataTable GetGroupInfo(int group_id)
public DataTable GetGroupInfo(string group_name)
```

Возвращает данные об указанной группе по её идентификатору или имени. Возвращаемые поля:

1) `group_id`,
2) `group_name`,
3) `created`,
4) `modified`.

Метод GetPermissionLevels

```
public DataTable GetPermissionLevels()
```

Возвращает данные об уровнях доступа. Возвращаемые поля:

1) `permission_level_id`,
2) `permission_level`,
3) `permission_level_name`.

Метод GetRootGroupsForUser

```
public DataTable GetRootGroupsForUser(int user_id)
```

Для указанного пользователя возвращает список из `group_id` групп, в которые он входит.

Метод GetUserInfo

```
public DataTable GetUserInfo(int user_id)
public DataTable GetUserInfo(string login)
```

Возвращает данные по указанному пользователю. Возвращаемые поля:

1) `user_id`,
2) `login`,
3) `password`,
4) `disabled`,
5) `first_name`,
6) `last_name`,
7) `email`,
8) `created`,
9) `modified`.

Метод GetUsersForGroup

```
public DataTable GetUsersForGroup(int group_id)
```

Возвращает список пользователей, явно входящих в группу. Возвращаемые поля:

1) `user_id`,
2) `login`,
3) `password`,
4) `disabled`,
5) `first_name`,
6) `last_name`,
7) `email`,
8) `created`,
9) `modified`.

Метод MoveUsersFromGroupToGroup

```
public void MoveUsersFromGroupToGroup(int fromGroupId, int toGroupId)
```

Перемещает пользователей из группы `fromGroupId` в группу `toGroupId`.

Метод RemoveAllEntitiesFromContentPermission

```
public void RemoveAllEntitiesFromContentPermission(int contentId)
```

Удаляет все права доступа, явно определённые для указанного контента.

Метод RemoveAllEntitiesFromItemPermission

```
public void RemoveAllEntitiesFromItemPermission(int itemId)
```

Удаляет все права доступа, явно определённые для указанной статьи.

Метод RemoveAllGroupsFromContentPermission

```
public void RemoveAllGroupsFromContentPermission(int contentId)
```

Удаляет все права доступа для групп, явно определённые для указанного контента.

Метод RemoveAllGroupsFromItemPermission

```
public void RemoveAllGroupsFromItemPermission(int itemId)
```

Удаляет все права доступа для групп, явно определённые для указанной статьи.

Метод RemoveAllUsersFromContentPermission

```
public void RemoveAllUsersFromContentPermission(int contentId)
```

Удаляет все права доступа для пользователей, явно определённые для указанного контента.

Метод RemoveAllUsersFromItemPermission

```
public void RemoveAllUsersFromItemPermission(int itemId)
```

Удаляет все права доступа для пользователей, явно определённые для указанной статьи.

Метод RemoveChildGroupFromParentGroup

```
public void RemoveChildGroupFromParentGroup(int parent_group_id, int child_group_id)
```

Удаляет связь «Родитель-потомок» для заданных родительской и дочерней групп.

Метод RemoveGroup

```
public void RemoveGroup(int groupId)
```

Удаляет указанную группу.

Метод RemoveGroupFromContentPermission

```
public void RemoveGroupFromContentPermission(int groupId, int contentId)
```

Удаляет явно определённые права доступа для группы на указанный контент.

Метод RemoveGroupFromItemPermission

```
public void RemoveGroupFromItemPermission(int groupId, int itemId)
```

Удаляет явно определённые права доступа для группы на указанную статью.

Метод RemoveUser

```
bool RemoveUser(int userId)
```

Удаляет указанного пользователя.

Метод RemoveUserFromContentPermission

```
public void RemoveUserFromContentPermission(int userId, int contentId)
```

Удаляет явно определённые права доступа для пользователя на указанный контент.

Метод RemoveUserFromItemPermission

```
public void RemoveUserFromItemPermission(int userId, int itemId)
```

Удаляет явно определённые права доступа для пользователя на указанную статью.

Метод RemoveUserFromGroup

```
public void RemoveUserFromGroup(int userId, int groupId)
```

Удаляет пользователя `userId` из группы `groupId`.

Метод UpdateGroup

```
public void UpdateGroup(int groupId, string newName)
public void UpdateGroup(int groupId, string newName, bool
AllowSharedOwnershipOfItems)
```

Изменяет имя указанной группы.

Параметры:

Название	Описание
groupId	Идентификатор группы
newName	Новое имя группы

Метод UpdateGroupContentPermission

```
public void UpdateGroupContentPermission(int groupId, int contentId, int
permissionId)
public void UpdateGroupContentPermission(int groupId, int contentId, int
permissionId, bool propagateToItems)
```

Изменяет явно определённые права доступа для группы на указанный контент.

Метод UpdateGroupItemPermission

```
public void UpdateGroupItemPermission(int groupId, int itemId, int permissionId)
```

Изменяет явно определённые права доступа для группы на указанную статью.

Метод UpdateUser

```
public void UpdateUser(int userId, string newUserName, string newPassword, string
newFirst_Name, string newLast_Name, string newEmail)
```

Изменяет данные об указанном пользователе.

Параметры:

Название	Описание
newUserName	Новое имя пользователя
newPassword	Новый пароль пользователя
newFirst_Name	Новое имя персоны-владельца пользователя
newLast_Name	Новая фамилия персоны-владельца пользователя
newEmail	Новый адрес электронной почты персоны-владельца пользователя

Метод UpdateUserContentPermission

```
public void UpdateUserContentPermission(int userId, int contentId, int permissionId,
bool propagateToItems)
```

Изменяет явно определённые права доступа для пользователя на указанный контент.

Метод UpdateUserItemPermission

```
public void UpdateUserItemPermission(int userId, int itemId, int permissionId)
```

Обновляет явно определённые права доступа для пользователя на указанную статью.

5.4.4. Класс ContentItem

Внимание: для использования этого API необходим экземпляр класса `DBConnector`. В отличие от CRUD-методов класса `DBConnector`, методы класса `ContentItem`:

- не кэшируют данные статей;
- поддерживают отправку уведомлений;
- работают с текущими версиями статей, а не с опубликованными.

Свойства

Свойство FieldValues

```
public Dictionary<string, ContentItemValue> FieldValues {get; internal set; }
```

Хэш-таблица значений полей статьи.

Примечание: в отличие от хэш-таблицы `Values`, используемой в методах класса `DBConnector`, в данном случае в качестве ключей используются заданные в QP имена полей.

Внимание: данные регистрозависимы.

В качестве значения используется экземпляр класса `ContentItemValue`, в котором есть свойства, отвечающие за данные статей. Данные поля «Связь» типов M2M и M2O доступны через свойство `LinkedItems`, а всех полей остальных типов – через свойство `Data`:

```
public string Data { get; set; }
public HashSet<int> LinkedItems {get; internal set; }
```

При создании новой или чтения существующей статьи хэш-таблица с данными о всех полях и их значениях загружается автоматически. Для новой статьи значения по умолчанию не подгружаются, но при сохранении учитываются, если соответствующие поля пришли пустыми.

Свойство Id

```
public int Id { get; set; }
```

Идентификатор статьи. Для новой статьи значение равно `0`.

Примечание: для создания копии статьи в текущем контенте достаточно:

- прочитать статью методом `Read`,
- для `Id` задать значение `0`,
- сохранить статью методом `Save`.

Свойство Visible

```
public bool Visible { get; set; }
```

Флаг видимости статьи. Значение по умолчанию – `true`.

Свойство Archive

```
public bool Archive { get; set; }
```

Флаг, определяющий, находится ли статья в архиве. Значение по умолчанию – `false`.

Свойство DelayedSchedule

```
public bool DelayedSchedule { get; set; }
```

Флаг отложенной публикации. По умолчанию – `false`.

Свойство LastModifiedBy

```
public int LastModifiedBy { get; set; }
```

Идентификатор пользователя, последним изменявшим статью. Значение по умолчанию – `1`.

Примечание: для интеграции с пользовательскими действиями следует использовать метод `LoadLastModifiedFromCustomTab`.

Свойство StatusName

```
public string StatusName { get; set; }
```

Имя статуса. Значение по умолчанию зависит от использования Workflow:

- если не используется, то `Published`,
- если используется, то `None`.

Свойство ContentId

```
public int ContentId { get; set; }
```

Идентификатор контента.

Примечание: при изменении статья будет перемещена.

Свойство Created

```
public DateTime Created { get; internal set; }
```

Дата создания статьи.

Свойство Modified

```
public DateTime Modified { get; internal set; }
```

Дата последнего изменения статьи.

Свойство Splitted

```
public bool Splitted { get; internal set; }
```

Флаг расщепления статьи. Значение меняется при изменении статуса.

Методы

Метод New

```
public static ContentItem New(int contentId, DBConnector cnn)
```

Создаёт новую статью в контенте с указанным идентификатором.

Примечание: если контент отсутствует, то выводится исключение.

Метод Read

```
public static ContentItem Read(int id, DBConnector cnn)
```

Получает данные существующей статьи с указанным идентификатором статьи.

Примечание: если статья отсутствует, то выводится исключение.

Метод Remove

```
public static void Remove(int id, DBConnector cnn)
```

Удаляет статью с указанным идентификатором статьи.

Примечание: нет реакции на событие, когда статья отсутствует.

Метод Save

```
public void Save()
```

Сохраняет статью после создания или изменения.

Примечание: в ходе работы:

- вызывает метод `AddFormToContent` класса `DBConnector`,
- формирует уведомление.

Метод LoadLastModifiedFromCustomTab

```
public void LoadLastModifiedFromCustomTab()
```

Загружает свойство `LastModifiedBy` из сессионной переменной, которая устанавливается при аутентификации пользователя в пользовательской вкладке методом `CheckCustomTabAuthentication`. Если загрузка не выполнена, то значение свойства не меняется.

Пример:

```
ContentItem item = ContentItem.New(284, cnn);
item.FieldValues["Title"].Data = "Test";
item.FieldValues["Date"].Data = DateTime.Today.ToString();
item.FieldValues["Category"].LinkedItems.Add(1660);
item.LoadLastModifiedFromCustomTab();
item.Save();
```

5.4.5. Классы из пространства имён «Quantumart.QP8.BLL.Services.API»

Класс ContentService

Конструктор:

```
public ContentService(string connectionString, int userId)
```

Параметры:

Название	Описание
connectionString	Строка подключения.
userId	Идентификатор пользователя, от имени которого должны выполняться действия.

```
public Content Read(int id)
```

Выполняет чтение существующего контента.

```
public Content Save(Content content)
```

Выполняет сохранение нового или существующего контента.

```
public void Delete(int contentId)
```

Выполняет удаление существующего контента.

Класс FieldService

Конструктор:

```
public FieldService(string connectionString, int userId)
```

Параметры идентичны параметрам класса `ContentService`.

```
public Field Read(int id)
```

Выполняет чтение существующего поля.

```
public IEnumerable<Field> List(int contentId)
```

Получает список полей для контента.

```
public Field Save(Field field, bool explicitOrder)
```

Выполняет сохранение нового или существующего поля.

Параметр `explicitOrder` определяет, учитывается ли при сохранении значение поля `Order`. При значении `false` поле будет помещено на последнее место в контенте.

```
public void Delete(int id)
```

Удаляет существующее поле.

```
public IEnumerable<Field> ListRelated(int contentId)
```

Возвращает список полей «Связь», ссылающихся на текущий контент (типы O2M, M2M, M2O).

Пример использования для программного создания полей

```
FieldService service = new FieldService(connectionString, userId);
var fields = service.List(contentId);
var order = fields.Single(n => n.Name == "RegionValue1766").Order;

Field veField = fields.Single(n => n.Name == "RegionValue1822");
veField.Id = 0;
veField.Name = "RegionValue20000";
veField.FriendlyName = "Значение для тестового региона";
veField.Order = order;
service.Save(veField, true);
```

Класс ArticleService

Конструктор:

```
public ArticleService(string connectionString, int userId)
```

Параметры идентичны параметрам класса `ContentService`.

```
public void LoadStructureCache()
```

Загружает кэш структуры данных (контенты, поля, статусы, пользователи).

Загружается в `QPContext`.

Рекомендуется использовать для ускорения получения списков и элементов по идентификатору для других методов API.

Есть возможность кэшировать структуру во внешнем контексте:

```
public void LoadStructureCache(IContextStorage st)
```

Есть возможность сбросить данные, кэшируемые во внешнем контексте:

```
public void LoadStructureCache(IContextStorage st, bool resetExternal)
```

```
public static bool IsLive
```

Свойство, влияющее на получение списков статей. Определяет, какие версии статей мы хотим получать: опубликованные или нет.

Хранит данные в `QPContext`.

```
public Article New(int contentId)
```

Создаёт новую статью в указанном контенте для последующего сохранения.

В статье заполнены только значения по умолчанию.

```
public Article Read(int id, bool forceLoadFieldValues = false)
```

Выполняет чтение указанной статьи.

Установка флага `forceLoadFieldValues` принудительно загружает коллекцию `FieldValues` и агрегированные статьи. По умолчанию используется шаблон «ленивая загрузка».

```
public IEnumerable<Article> List(int contentId, int[] ids)
```

Возвращает список статей контента.

Можно ограничить набор статей путём указания списка их идентификаторов в параметре `ids`. Если передать `null`, то возвращаются все статьи контента.

```
public string GetRelatedItems(int fieldId, int? id)
```

Возвращает список идентификаторов связанных статей (в виде строки через запятую) для поля «Связь» типа M2O.

Идентификатор поля передаётся в параметре `fieldID`.

```
public string GetLinkedItems(int linkId, int id)
```

Возвращает список идентификаторов связанных статей (в виде строки через запятую) для поля «Связь» типа M2M.

Идентификатор связи (не поля) передается в параметре `linkId`.

```
public Dictionary<int, string> GetRelatedItemsMultiple(int fieldId, IEnumerable<int>
ids)
```

Возвращает коллекцию списков идентификаторов связанных статей (в виде строки через запятую) для поля «Связь» типа M2O по списку идентификаторов статей.

Идентификатор поля передается в параметре `fieldID`.

```
public Dictionary<int, string> GetLinkedItemsMultiple(int linkId, IEnumerable<int>
ids)
```

Возвращает коллекцию списков идентификаторов связанных статей (в виде строки через запятую) для поля «Связь» типа M2M по списку идентификаторов статей.

Идентификатор связи (не поля) передается в параметре `linkId`.

```
public Article CopyAndRead(Article article)
```

Копирует указанную статью и возвращает статью-результат.

```
public CopyResult Copy(Article article)
```

Копирует указанную статью и возвращает результат копирования.

```
public Article Save(Article article)
```

Сохраняет или изменяет указанную статью (в зависимости от её идентификатора).

```
public MessageResult Delete(int articleId)
```

Удаляет статью по указанному идентификатору.

При удалении производится проверка прав, отправка уведомлений и т.д.

```
public MessageResult Delete(int contentId, int[] articleIds)
```

Удаляет статьи указанного контента по списку идентификаторов.

При удалении производится проверка прав, отправка уведомлений и т.д.

```
public void SimpleDelete(int[] articleIds)
```

Удаляет статьи по списку идентификаторов без выполнения дополнительных действий.

```
public MessageResult SetArchiveFlag(int contentId, int[] articleIds, bool flag)
```

Добавляет в Архив статей или восстанавливает из Архива статей (в зависимости от параметра `flag`) статьи указанного контента по списку идентификаторов.

При восстановлении производится проверка прав, отправка уведомлений и т.д.

```
public void SimpleSetArchiveFlag(int[] articleIds, bool flag)
```

Добавляет в Архив статей или восстанавливает из Архива статей (в зависимости от параметра `flag`) по списку идентификаторов без выполнения дополнительных действий.

```
public MessageResult Publish(int contentId, int[] articleIds)
```

Публикует статьи указанного контента по списку идентификаторов.

При публикации производится проверка прав, отправка уведомлений, и т.д.

```
public void SimplePublish(int[] articleIds)
```

Публикует статьи по списку идентификаторов без выполнения дополнительных действий.

```
public string[] GetFieldValues(int[] ids, int contentId, int fieldId)
```

Получает данные из указанного контента с ограничением по списку идентификаторов статей и идентификатора поля.

```
public string[] GetFieldValues(int[] ids, int contentId, string fieldName)
```

Получает данные из указанного контента с ограничением по списку идентификаторов статей и имени поля.

Класс CustomActionService

Конструктор:

```
public CustomActionService(string connectionString, int userId)
```

Параметры идентичны параметрам класса `ContentService`.

```
public CustomAction ReadByCode(string code)
```

Выполняет чтение пользовательского действия по коду.

Класс DbService

Конструктор:

```
public DbService(string connectionString, int userId)
```

Параметры идентичны параметрам класса `ContentService`.

```
public Dictionary<string, string> GetAppSettings()
```

Выполняет чтение настроек веб-приложения (ключ-значение), заданных в бекэнде (пункт контекстного меню «Настройки» для корневого элемента в дереве сущностей, свойство «Настройки приложения» (Application Settings)).

5.4.6. Классы QPage и QUserControl

Основные свойства

Свойство page_id

```
public int page_id { get; set; }
```

Идентификатор страницы. Свойство устанавливается механизмом сборки на уровне страницы.

Свойство page_template_id

```
public int page_template_id { get; set; }
```

Идентификатор шаблона. Свойство устанавливается механизмом сборки на уровне страницы.

Свойство site_id

```
public int site_id { get; set; }
```

Идентификатор сайта. Свойство устанавливается механизмом сборки на уровне страницы.

Свойство upload_url

```
public string upload_url { get; set; }
```

URI Библиотеки сайта. Свойство устанавливается на уровне страницы при её инициализации.

Пример:

```
<img src="<%# upload_url%>/my_photo.jpg">
```

Свойство site_url

```
public string site_url { get; set; }
```

URI корневой директории веб-сайта. Свойство устанавливается на уровне страницы при её инициализации.

Примечание: значения различаются в Live- и Stage-режимах.

Свойство absolute_site_url

```
public string absolute_site_url { get; set; }
```

URL корневой директории веб-сайта. Свойство устанавливается на уровне страницы при её инициализации.

Примечание: значения различаются в Live- и Stage-режимах.

Свойство IsStage

```
public bool IsStage { get; private set; }
```

Показывает, в каком режиме (Live или Stage) была собрана страница или элемент управления. Свойство устанавливается механизмом сборки на уровне страницы.

Свойство Cnn

```
public DBConnector Cnn { get; private set; }
```

Текущий экземпляр класса `DBConnector`.

Обычно на страницах QP не требуется прямое обращение к этому свойству, а достаточно стандартной функциональности, которую предоставляют классы `QPage`, `QUserControl` и `QPublishControl`. Свойство устанавливается на уровне страницы при её инициализации.

Свойство Values

```
public Hashtable Values { get; set; }
```

Коллекция `Values`. Свойство устанавливается на уровне страницы при ее инициализации.

Внимание: прямое обращение к коллекции не рекомендуется. Если такой код встречается при сопровождении, его нужно заменить на вызов одного из методов для работы с Values.

Основные методы

Раздел описывает методы, доступные из страниц и элементов управления веб-сайтом, реализованным на объектах QP (в классах, унаследованных от `QPage` и `QUserControl`).

Примечание: если возможностей данного API не хватает, то следует использовать расширенный API, реализованный в классе `DBConnector`. Экземпляр данного класса доступен на странице и в элементах управления сайтом через свойство `Cnn`.

Методы для вызова объектов

Метод ShowObject

```
public void ShowObject(string object_name)
public void ShowObject(string object_name, object sender)
public void ShowObject(string object_name, object sender, object[] parameters)
```

Вызывает объект из Code Behind.

Параметры:

Название	Описание
object_name	Имя объекта. Поддерживаются вызовы в формате *TemplateName*.*ObjectName*.*FormatName* Примечание: задание *TemplateName* и *FormatName* не обязательно.
sender	Родительский элемент управления, в коллекцию `Controls` которого должен быть загружен объект. При отсутствии параметра вызываемый объект будет загружен в коллекцию `Controls` текущего объекта.
parameters	Параметры, которые будут переданы в конструктор соответствующего элемента управления (аналогично методу `LoadControl`).

При загрузке объекта используется модифицированная последовательность событий, чтобы работал механизм `Values`. Для того, чтобы генерировалась оригинальная последовательность событий ASP.NET, нужно использовать метод `ShowObjectSimple`.

Пример:

```
ShowObject("Main.Menu", this);
```

Метод ShowObjectSimple

Внимание: метод несовместим с механизмом `Values`.

```
public void ShowObjectSimple(string object_name)
public void ShowObjectSimple(string object_name, sender)
public void ShowObjectSimple(string object_name, sender, object[] parameters)
```

Аналог метода `ShowObject`. Отличие в том, что при вызове метода генерируется оригинальная последовательность событий ASP.NET, что позволяет корректно восстанавливать ViewState для серверных элементов управления.

Метод GetInternalCall

```
public string GetInternalCall(string object_name)
```

Возвращает относительный путь к элементу управления, который может быть использован для его явной загрузки. Поддерживаются вызовы в формате *TemplateName*.*ObjectName*.*FormatName*.

Примечание: задание *TemplateName* и *FormatName* не обязательно.

Методы для работы с коллекцией Values

Метод AddValue

```
public void AddValue(string key, object value)
```

Добавляет пару “ключ-значение” в коллекцию Values.

Примечание: получить значение по ключу можно с помощью методов:

- `DirtyValue`,
- `Value`,
- `StrValue`,
- `NumValue`.

Пример:

```
AddValue("Age", 35);
```

Метод Value

```
string Value(string key)
```

Возвращает значение из коллекции `Values` по заданному ключу.

Примечание: ключ не зависит от регистра.

Внимание: из результата удаляются апострофы с целью предотвращения атак с использованием SQL-инъекций. Для получения неизменённого значения следует использовать метод `DirtyValue`.

Метод DirtyValue

```
public string DirtyValue(string key)
```

Возвращает значение из коллекции `Values` по заданному ключу без изменений.

Примечание: ключ не зависит от регистра.

Метод NumValue

```
public long NumValue(string key)
```

Возвращает значение из коллекции `Values` по заданному ключу.

Внимание: результат преобразуется в число. Если преобразование неуспешно, то возвращается значение `0`. Таким образом, метод гарантированно возвращает число, что делает его безопасным с точки зрения атак с использованием SQL-инъекций. Рекомендуется для передачи числовых параметров в объект типа «Publishing Container».

Примечание: ключ не зависит от регистра.

Примечание: для безопасной передачи строковых параметров следует использовать метод `StrValue`.

Пример:

```
"[content_item_id] = " + NumValue("id")
```

Метод StrValue

```
public string StrValue(string key)
```

Возвращает значение из коллекции `Values` по заданному ключу.

Внимание: все апострофы в результирующей строке удваиваются, что предотвращает атаки с использованием SQL-инъекций. Рекомендуется для передачи строковых параметров в объект типа «Publishing Container».

Примечание: ключ не зависит от регистра.

Примечание: для безопасной передачи числовых параметров следует использовать метод `NumValue`.

Пример:

```
"[text] = '" + StrValue("text") + "'"
```

Методы для работы с полями

Метод Field

```
public string Field(DataRowView pDataItem, string key)
public string Field(DataRowView pDataItem, string key, string defaultValue)
public string Field(DataRow pDataItem, string key)
public string Field(DataRow pDataItem, string key, string defaultValue)
```

Возвращает значение поля статьи.

Внимание: для возвращаемого значения выполняется замена заполнителей. Если требуется выполнить только замену без чтения данных, то следует использовать метод `FormatField`.

Примечание: в режиме OnScreen метод генерирует ГПИ для изменения значения поля. Если это нежелательно, то следует использовать метод `FieldNS`.

Параметры:

Название	Описание
pDataItem	Строка контентной таблицы, соответствующая статье. **Примечание:** версия с `DataRowView` обычно применяется в Presentation, с `DataRow` – в Code Behind.
key	Имя поля.
defaultValue	Значение по умолчанию. Используется, если значение, полученное из таблицы – `NULL` или пустая строка.

Пример Presentation объекта «Publishing Container» с кодом по умолчанию:

```
<ItemTemplate>
...
<%# Field(((DataRowView)(Container.DataItem)), "Title")%>
...
</ItemTemplate>
```

Пример Code Behind объекта «Publishing Container» с кодом по умолчанию:

```
protected void OnItemCreated(Object sender, RepeaterItemEventArgs e) {
  if ((e.Item.ItemType == ListItemType.Item) | (e.Item.ItemType ==
ListItemType.AlternatingItem)) {
...
string title = Field(Data.Rows[e.Item.ItemIndex], "Title");
...
  }
}
```

Примечание: метод можно применять в любых объектах QP.

Метод FieldNS

```
public string FieldNS(DataRowView pDataItem, string key)
public string FieldNS(DataRowView pDataItem, string key, string defaultValue)
public string FieldNS(DataRow pDataItem, string key)
public string FieldNS(DataRow pDataItem, string key, string defaultValue)
```

Аналогичен методу `Field`. Отличие – в режиме OnScreen не генерируется код для изменения значения поля. Рекомендуется применять в случаях, когда вставка дополнительного кода может нарушить вёрстку веб-сайта.

Метод FormatField

```
public string FormatField(string key)
```

Выполняет замену заполнителей на URL. Замена осуществляется на следующие URL:

- `upload_url`,
- `site_url` или `absolute_site_url` (в зависимости от значения параметра `UseAbsoluteSiteUrl`).

Рекомендуется применять при чтении данных из статей во всех случаях, кроме использования классов LINQ to SQL и метода `Field` (уже обладают данной функциональностью).

Метод OnScreenFlyEdit

Внимание: режим OnScreen существует только в предыдущих версиях продукта.

```
public string OnScreenFlyEdit(string Value, int ItemID, string FieldName)
```

Позволяет работать в режиме OnScreen вне объекта «Publishing Container».

Параметры:

Название	Описание
Value	Текст, который должен быть отредактирован.
ItemID	Идентификатор статьи.
FieldName	Имя поля.

Метод OnScreen

Внимание: режим OnScreen существует только в предыдущих версиях продукта.

```
public string OnScreen(string Value, int ItemID)
```

Аналогичен методу `OnScreenFlyEdit`. Отличие – при использовании метода Разработчик может изменить статью только с переходом в свойства статьи, а не непосредственно в режиме OnScreen.

Методы для работы с виртуальными путями

Примечание: все методы неявно используют параметр `site_id` страницы, устанавливаемый при сборке.

Метод GetSiteUrl

```
public string GetSiteUrl()
```

Возвращает URL корневой директории страниц сайта.

Внимание: результат зависит от того, в каком режиме была собрана страница (Live или Stage).

Примечание: реализован с помощью метода GetSiteUrl класса DBConnector.

Метод GetActualSiteUrl

```
public string GetActualSiteUrl()
```

Возвращает URI корневой директории страниц сайта.

Внимание: результат зависит от того, в каком режиме была собрана страница.

Примечание: реализован с помощью метода GetActualSiteUrl класса DBConnector.

Метод GetContentUploadUrl

```
public string GetContentUploadUrl(string content_name)
```

Возвращает URI Библиотеки контента для указанного контента.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Примечание: реализован с помощью метода GetContentUploadUrl класса DBConnector.

Метод GetFieldUploadUrl

```
public string GetFieldUploadUrl(string contentName, string fieldName)
```

Возвращает URL корневой директории поля.

Учитывает настройки «Использовать библиотеку сайта» (Use Site Library) и «Подпапка для файлов» (File Subfolder) уровня поля, а также директории для динамических изображений.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Примечание: реализован с помощью метода `GetUrlForFileAttribute` класса `DBConnector`.

При использовании внутри объекта «Publishing Container» рекомендуется использовать версию метода из класса `QPublishControl`, в которой контент уже определён.

Методы для работы с контентами

Метод GetContentID

```
public int GetContentID(string content_name)
```

Возвращает идентификатор указанного контента.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Примечание: реализован с помощью метода `GetContentId` класса `DBConnector`.

Метод GetContentData

```
public DataTable GetContentData(string siteName, string contentName, string whereExpression, string orderExpression, long startRow, long pageSize, ref long totalRecords, byte useSchedule, string statusName, byte showSplittedArticle, byte includeArchive)
```

Возвращает данные статьи.

Примечание: реализован с помощью метода `GetContentData` класса `DBConnector`. Логика работы и назначение параметров совпадают.

Метод GetContentItemLinkIDs

```
public string GetContentItemLinkIDs(string linkFieldName, long itemID)
public string GetContentItemLinkIDs(string linkFieldName, string itemID)
```

В первом варианте использования возвращает значения указанного названия поля M2M (параметр `linkFieldName`) для указанного идентификатора статьи (параметр `itemID`) в виде списка идентификаторов связанных статей, разделённых запятыми.

Во втором варианте использования в качестве параметра `itemID` может быть передан список идентификаторов статей, разделённых запятыми.

Примечание: реализован с помощью метода `GetContentItemLinkIDs` класса `DBConnector`.

Пример:

```
GetContentItemLinkIDs("Authors", Field(Data.Rows[e.Item.ItemIndex], "content_item_id"));
```

Метод GetContentItemLinkQuery

```
public string GetContentItemLinkQuery(string linkFieldName, string itemID)
```

Аналог метода `GetContentItemLinkIDs`. В данном случае запрос не выполняется, а возвращается в виде текста.

Примечание: реализован с помощью метода `GetContentItemLinkQuery` класса `DBConnector`.

Методы для обработки форм

Методы предназначены для создания или изменения статей контента посредством HTML-форм, но могут также использоваться для изменения содержимого контентов.

Метод AddFormToContent

Метод используется для создания новой статьи:

```
public int AddFormToContent(string content_name, string status_name)
```

и изменения существующей:

```
public int AddFormToContent(string content_name, string status_name, int content_item_id)
```

Возвращает идентификатор статьи. Также этот ID записывается в коллекцию `Values` с ключом `new_content_item_id`.

Метод поддерживает отправку уведомлений. Если такое поведение нежелательно, то следует использовать метод `AddFormToContentWithoutNotification`.

В методе производятся следующие проверки на сохраняемые поля:

- тип поля («Числовой», «Дата»),

- обязательность,
- уникальность,
- допустимая длина и маска ввода (для поля типа «Строка»).

Дополнительно:

- при обновлении статьи создается её версия,
- вся работа с БД происходит в одной транзакции.

Параметры:

Название	Описание
content_name	Имя контента. **Примечание:** поддерживается динамическое изменение контента.
status_name	Имя статуса Workflow.
content_item_id	Идентификатор изменяемой статьи.

Перед вызовом метода значения полей статьи должны находиться в коллекции `Values`. Данные могут попасть в коллекцию:

1) автоматически при отправке на сервер HTML-формы (имена полей формы должны формироваться с помощью метода `FieldName`),
2) вручную с помощью метода `AddValue` (ключи также должны быть созданы с помощью `FieldName`).

Примечание: если в форме есть поля типа «Файл» (имена полей должны быть сформированы с помощью метода `FieldName`), то полученные файлы будут сохранены в Библиотеку контента. Если файл с таким именем уже существует, то файл будет сохранен под изменённым именем с числовым индексом (например, `image[1].jpg`). Также, при необходимости, будут сгенерированы значения для полей «Динамическое изображение».

Примечание: Поддерживается режим расщепления статей, включая поля «Связь» типов M2M и M2O. Значения таких полей должны представлять собой строку идентификаторов связанных статей, разделенных запятыми.

Примечание: реализован с помощью метода `AddFormToContent` класса `DBConnector`.

Пример:

```
AddValue(FieldName("Users", "Login"), Login);
AddValue(FieldName("Users", "Password"), Password);
AddFormToContent("Users",  "Published");
```

Метод AddFormToContentWithoutNotification

```
public int AddFormToContentWithoutNotification(string content_name, string
status_name)

public int AddFormToContentWithoutNotification(string content_name, string
status_name, int content_item_id)
```

Аналог метода `AddFormToContent`. Отличие в том, что нет поддержки уведомлений.

Метод UpdateContentItem

```
public void UpdateContentItem()
```

Метод аналогичен методу `AddFormToContent` для изменения существующей статьи. В данном случае идентификатор статьи передаётся через коллекцию `Values` (ключ `content_item_id`). Имя контента вычисляется по идентификатору статьи.

Метод поддерживает отправку уведомлений. Если такое поведение нежелательно, то следует использовать метод `UpdateContentItemWithoutNotification`.

Если в коллекции `Values` будут присутствовать не все поля, то значения обнулятся. В случае, если это нежелательно, то необходимо использовать перегруженную версию метода:

```
public void UpdateContentItem(bool updateEmpty, string statusName)
```

Параметры:

Название	Описание
statusName	Статус Workflow для статьи. **Примечание:** если статус менять не нужно, то следует передать пустую строку.
updateEmpty	При `updateEmpty = false` позволяет изменить данные только в требуемых полях статьи. **Примечание:** данный вариант предпочтительнее, чем несколько вызовов `UpdateContentItemField`.

Примечание: реализован с помощью метода `UpdateContentItem` класса `DBConnector`.

Пример:

```
AddValue("content_item_id", 12345);
AddValue(FieldName("Users", "Login"), Login);
AddValue(FieldName("Users", "Password"), Password);
UpdateContentItem();
```

Метод UpdateContentItemWithoutNotification

```
public void UpdateContentItemWithoutNotification()
```

Аналог метода `UpdateContentItem`. Отличие в том, что нет поддержки уведомлений.

Метод UpdateContentItemField

```
public void UpdateContentItemField(string content_name, string field_name, int
content_item_id)
```

Метод аналогичен методу `AddFormToContent` для изменения существующей статьи. Отличие в том, что метод предназначен для обновления только одного поля статьи.

Примечание: если необходимо обновление сразу нескольких определённых полей, то следует использовать метод `UpdateContentItem` с `UpdateEmpty = false`.

По умолчанию метод не поддерживает уведомления. При необходимости формирования уведомлений следует использовать перегруженную версию метода:

```
public void UpdateContentItemField(string content_name, string field_name, int
content_item_id, bool with_notification)
```

Пример:

```
AddValue(FieldName("Users", "Login"), "newLogin");
UpdateContentItemField("Users", "Login", Value("cid"));
```

Примечание: реализован с помощью метода `UpdateContentItemField` класса `DBConnector`.

Метод AddUpdateContentItemLink

```
public int AddUpdateContentItemLink(string LinkFieldName, int ItemID, string
LinkItems, string TargetLinkItems)
```

Добавляет или изменяет указанное поле “Связь” типа M2M (параметр `LinkFieldName`) для указанной статьи (параметр `ItemID`). Значение поля передается в параметре `LinkItems`, представляет собой строку идентификаторов связанных статей, разделённых запятыми. Параметр `TargetLinkItems` позволяет ограничить набор изменяемых статей (формат значения идентичен формату `LinkItems`).

Примечание: если изменение должно затрагивать все статьи, то в качестве значения `TargetLinkItems` нужно передать пустую строку.

При удачном завершении возвращает значение `1`, при неудаче возвращает `-1`.

Пример:

```
AddUpdateContentItemLink("Books", NumValue("AuthorID"),
Value("12345,45678,7890,23415,8907"), "");
```

Метод RemoveContentItem

```
public void RemoveContentItem(int content_item_id)
```

Удаляет статью с указанным идентификатором.

Метод поддерживает уведомления.

Примечание: реализован с помощью метода `DeleteContentItem` класса `DBConnector`.

Метод DeleteContentItem

```
public void DeleteContentItem()
```

Удаляет статью с идентификатором, который передается через коллекцию `Values` (ключ `content_item_id`).

Метод поддерживает уведомления.

Примечание: реализован с помощью метода `DeleteContentItem` класса `DBConnector`.

Вспомогательные методы

Метод FieldName

```
public string FieldName(string content_name, string field_name)
```

Возвращает внутреннее имя поля (`field_`*идентификатор поля*).

Примечание: обычно используется в HTML-формах для наименования полей совместно с методами `AddFormToContent` или `UpdateContentItem`.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Примечание: реализован с помощью метода `FieldName` класса `DBConnector`.

Метод FieldID

```
public int FieldID(string content_name, string field_name)
```

Возвращает идентификатор поля.

Примечание: поддерживается имя контента в формате *SiteName*.*ContentName*.

Примечание: реализован с помощью метода `FieldID` класса `DBConnector`.

Метод ReplaceHTML

```
public string ReplaceHTML(string str)
```

Заменяет символы < и > на безопасные XML-сущности `<` и `>`.

Примечание: используется в качестве защиты от XSS-атак.

Метод SendNotification

```
public void SendNotification(string notification_on, int content_item_id, string
notification_email)
```

Запускает механизм формирования уведомления об указанном типе события.

Параметры:

Название	Описание
notification_on	Тип события. **Внимание:** требуемый тип события должен быть задан в настройках уведомления. Допустимые значения: • `for_create`, • `for_modify`, • `for_remove`, • `for_status_changed`, • `for_frontend`.
content_item_id	Идентификатор статьи.
notification_email	Адрес электронной почты, на который требуется сформировать уведомление. **Внимание:** указанное значение перекрывает стандартные настройки уведомления. Если это не нужно, то следует передать пустую строку.

Примечание: автоматически вызывается методами:

- `AddFormToContent`,
- `UpdateContentItem`,
- `RemoveContentItem`.

Пример:

```
SendNotification("for_remove", 3657, "");
```

Примечание: реализован с помощью метода `SendNotification` класса `DBConnector`.

5.4.7. Класс QPublishControl

В разделе описаны дополнительные методы и свойства, доступные из кода объекта типа «Publishing Container». Кроме того, в таком объекте остаются доступными все методы и свойства класса `QUserControl` (например, методы для работы с полями).

Свойства

Свойство Data

```
public DataTable Data { get; set; }
```

Таблица результатов запроса, полученных в соответствии с настройками объекта «Publishing Container». По умолчанию генерируется код, привязывающий данную таблицу к объекту типа Repeater, но это не единственно возможный вариант и разработчик может использовать эту таблицу любым способом.

Свойство TotalRecords

```
public long TotalRecords { get; set; }
```

Возвращает общее число записей объекта «Publishing Container» с учётом фильтрации и деления на страницы (совпадает с `Data.Rows.Count`).

Свойство AbsoluteTotalRecords

```
public long AbsoluteTotalRecords { get; set; }
```

Возвращает общее число записей объекта «Publishing Container» с учетом фильтрации, но без учёта деления на страницы (т.е. может не совпадать с `Data.Rows.Count`).

Свойство ContentID

```
public long ContentID { get; set; }
```

Возвращает идентификатор текущего контента, к которому привязан объект «Publishing Container».

Свойство ContentName

```
public string ContentName { get; set; }
```

Возвращает имя текущего контента, к которому привязан объект «Publishing Container».

Свойство ContentUploadURL

```
public string ContentUploadURL { get; set; }
```

Возвращает путь к Библиотеке контента для текущего контента.

Примечание: обычно используется совместно с методом Field для полей типа «Изображение» или «Файл». Рекомендуется использовать более универсальный метод `GetFieldUploadUrl`.

Пример:

```
<img src="<%#ContentUploadURL%>/<%#Field(CType(Container.DataItem, DataRowView),
"Product Picture")%>">
```

Свойство RecordsPerPage

```
public long RecordsPerPage { get; set; }
```

Возвращает общее количество записей на странице в соответствии с настройками объекта «Publishing Container».

Методы

Метод GetFieldUploadUrl

```
public string GetFieldUploadUrl(string fieldName)
```

Возвращает путь к корневой директории указанного поля контента, с которым связан объект «Publishing Container».

Примечание: в базовом случае результат совпадает с `ContentUploadURL`. Отличия возможны в следующих случаях:

- использование Библиотеки сайта вместо Библиотеки контента,
- поле имеет тип «Динамическое изображение»,
- используется свойство «Подпапка для файлов».

5.5. Классы LINQ to SQL

Использование классов LINQ to SQL упрощает работу с содержимым БД за счёт того, что:

- доступ к полям контентов становится типизированным (с поддержкой IntelliSense в Visual Studio);
- LINQ to SQL предоставляет большое количество встроенных операций для работы с наборами данных. Единицами данных в этих наборах будут являться статьи.
- становится доступным поддержка декларативного связывания контентов с элементами управления (по `DataSourceID`) с помощью стандартного класса `LinqDataSource`;
- добавляется возможность изменения статей с помощью LINQ-классов.
- связанные таблицы становятся доступными, как свойства (включая поле «Связь» типа M2M).

5.5.1. Сборка контентов в классы LINQ to SQL

Внимание: процесс сборки с большой долей вероятности приведёт к перезагрузке веб-приложения, поэтому в production-окружении его следует применять с осторожностью.

Сборка контентов в классы LINQ to SQL может быть выполнена следующими способами:

Название	Описание
С использованием БД	Используются настройки LINQ to SQL (задаются в бекэнде в свойствах сайта, контента, поля).
С использованием пользовательского файла отображения	Файл может быть: • сгенерирован через бекэнд (по умолчанию), • составлен Разработчиком вручную.

Переключение режимов осуществляется опцией «Использовать прямое отображение из базы данных» в свойствах сайта.

Сборка выполняется с помощью опции «Собрать контенты» (Assemble Contents) из контекстного меню сайта. При этом в директории `App_Code` происходит полная перегенерация всех классов LINQ to SQL для сайта.

Опция «Импортировать файл отображения в базу данных» позволяет импортировать существующий пользовательский файл отображения в БД для последующей настройки через бекэнд и генерации файла отображения в автоматическом режиме. При успешном выполнении импорта:

- данная опция автоматически отключается,
- опция «Использовать прямое отображение из базы данных» автоматически включается.

Опция «Импортировать файл отображения в базу данных» также может быть полезна для первичного заполнения настроек LINQ to SQL для существующего сайта с большим количеством контентов. Для этого сначала генерируется пользовательский файл отображения по умолчанию (при отключённой опции «Использовать прямое отображение из базы данных»), а затем выполняется его импорт.

Генерация классов LINQ to SQL и вспомогательных файлов осуществляется в директориях `App_Data` и `App_Code` веб-сайта. Их расположение вычисляется по пути директории `bin` сайта (директории `App_Data` и `App_Code` находятся на том же уровне, что и `bin`).

Настройка отображения с использованием БД

Режим используется при включённой опции «Использовать прямое отображение из базы данных».

Название	Описание
Настройки уровня сайта	Задаются в свойствах сайта (см. раздел Параметры сборки в LINQ-классы (LINQ Assembling Parameters))
Настройки уровня контента	Задаются в свойствах контента (см. раздел Параметры отображения в LINQ-классы (LINQ Mapping Parameters))
Настройки уровня поля	Задаются в свойствах поля (см. раздел Параметры отображения в LINQ-классы (LINQ Mapping Parameters))

Использование пользовательского файла отображения

Общие сведения

Пользовательский файл отображения — это XML-файл, на основании которого осуществляется генерация классов LINQ to SQL. Файл должен иметь имя `Mapping.xml` и находиться в директории `App_Data` веб-сайта.

Примечание: генерация классов LINQ to SQL в этом режиме осуществляется только при отключённой опции «Использовать прямое отображение из базы данных» на уровне сайта.

В текущей версии продукта файл `Mapping.xml` не генерируется автоматически. Вместо этого генерируется пользовательский файл отображения по умолчанию с именем `DefaultMapping.xml`, на основе которого вручную должен быть составлен файл `Mapping.xml`. В простейшем случае достаточно переименовать `DefaultMapping.xml` в `Mapping.xml`.

Примечание: автоматическая генерация файла `Mapping.xml` отсутствует из-за возможности случайной генерации классов LINQ to SQL на ненастроенном сайте, что может приводить к его неработоспособности.

При ручном создании файла отображения в нём задаются только необходимые данные. Все дополнительные данные, которые нужны для генерации классов, получаются из БД. При этом классы и свойства генерируются только для тех контентов и полей, которые описаны в пользовательском файле отображения.

Автоматическая генерация файла

При автоматической генерации пользовательского файла отображения по умолчанию корректные идентификаторы .NET создаются по следующим правилам:

- удаляются пробелы и подчёркивания,
- для данных на русском языке выполняется транслитерация,
- идентификаторы ссылочных полей генерируются на основе идентификаторов связанных контентов,
- к повторяющимся значениям идентификаторов добавляется числовой суффикс.

Пример файла отображения

```
<?xml version="1.0" encoding="utf-8" ?>
<schema connectionStringName="qp_database" replaceUrls="true" useLongUrls="true">
  <content name="Event Category" mapped_name="Category"
plural_mapped_name="Categories">
    <attribute name="Title" />
  </content>
  <content name="Events" mapped_name="Event" plural_mapped_name="Events"
use_default_filtration="false">
    <attribute name="Title" />
```

```
    <attribute name="Category" />
    <attribute name="Category 2" mapped_name="SecondCategory"
mapped_back_name="EventsBySecondCategory" />
    <attribute name="Location" />
    <attribute name="Date" />
    <attribute name="Event Description" mapped_name="Description" />
    <attribute name="Many" />
    <attribute name="Related Event" mapped_name="RelatedEvent" />
    <attribute name="SubEvents" />
    <attribute name="InLive" />
    <attribute name="SubTitle" />
  </content>
  <content name="Event Location" mapped_name="Location"
plural_mapped_name="Locations">
    <attribute name="Title" />
  </content>
  <content name="Photos" mapped_name="Photo" plural_mapped_name="Photos">
    <attribute name="Name" />
    <attribute name="Description" />
    <attribute name="OwnerID" />
    <attribute name="Photo" mapped_name="Picture" />
  </content>
  <link id="17" mapped_name="RelatedEvent" plural_mapped_name="RelatedEvents"  />
</schema>
```

Элементы файла отображения

Элемент schema

Корневой элемент файла.

Примечание: отображение может быть настроено с использованием бекэнда (свойства сайта, раздел «Параметры сборки в LINQ-классы»). При ручной настройке отображения конкретный сайт указывать не требуется, так как его идентификатор будет передан в процессе сборки.

Название	Описание
connectionStringName	В бекэнде – поле «Имя строки подключения» (Connection string name).
replaceUrls	В бекэнде – опция «Заменять URL» (Replace URLs).
useLongUrls	В бекэнде – опция «Использовать длинные URL» (Use long URLs).
namespace	В бекэнде – поле «Пространство имён для генерируемых классов» (Namespace for generated classes).
class	В бекэнде – поле «Имя контекстного класса» (Context Class Name).

Элемент content

Соответствует контенту QP.

Примечание: отображение может быть настроено с использованием бекэнда (свойства контента, раздел «Параметры сборки в LINQ-классы»).

Примечание: Если какой-либо контент не задан в отображении, то генерация класса для него проводиться не будет (в бекэнде – выключена опция «Отображать как класс»).

Название	Описание
name	Имя контента. По этому полю осуществляется поиск соответствия между отображением и БД. Если название контента (поля) в БД является недопустимым идентификатором в .NET, то необходимо задать атрибуты `mapped_name` и `plural_mapped_name`.
mapped_name	В бекэнде – поле «Имя (единственное)».
plural_mapped_name	В бекэнде – поле «Имя (множественное)».
use_default_filtration	В бекэнде – опция «Использовать фильтрацию по умолчанию».

Элемент attribute

Соответствует полю контента QP.

Примечание: отображение может быть настроено с использованием бекэнда (в свойствах поля раздел «Параметры сборки в LINQ-классы»).

Примечание: если какое-либо поле не задано в отображении, то генерация свойства для него проводиться не будет (в бекэнде – выключена опция «Отображать как LINQ-свойство»).

Название	Описание
name	Имя поля в QP. По этому полю осуществляется поиск соответствия между отображением и БД. Если название контента (поля) в БД является недопустимым идентификатором в .NET, то необходимо задать атрибут `mapped_name`. Кроме того, изменение имени при отображении поля может понадобиться, чтобы исключить конфликт имён между полем и контентом.
mapped_name	В бекэнде – поле «Имя LINQ-свойства».
mapped_back_name	В бекэнде – поле «Имя обратного LINQ-свойства».

Элемент link

Соответствует связи в QP (поле «Связь» типа M2M).

Примечание: отображение может быть настроено с использованием бекэнда (свойства поля, раздел «Параметры сборки в LINQ-классы».

Примечание: Если какая-либо связь не задана в отображении и не используется полями «Связь» типа M2M, то генерация класса для него не выполняется (в бекэнде – выключена опция «Отображать узловую таблицу как класс»).

Примечание: Если связь не задана в отображении, но используется полями, то будет сгенерирован класс с именем по умолчанию.

Название	Описание
id	Идентификатор связи (таблица `CONTENT_TO_CONTENT`). По этому полю осуществляется поиск соответствия между отображением и БД.
mapped_name	В бекэнде – поле «Имя узлового LINQ-класса (единственное)».
plural_mapped_name	В бекэнде – поле «Имя узлового LINQ-класса (множественное)».

5.5.2. Использование контекстного класса

Под контекстным классом понимается класс, сгенерированный механизмом сборки в LINQ to SQL. Этот класс наследуется от `DataContext`. Имя класса может быть задано на уровне сайта или контента. В случае отсутствия значения в качестве имени используется значение `QPDataContext`.

Экземпляр по умолчанию

Если требуется работать только с одним контекстным классом, то можно использовать экземпляр контекстного класса по умолчанию, доступный как `LinqHelper.Context`. Он создаётся при первом запросе к нему, в случае вызова в веб-среде сохраняется на время выполнения текущего веб-запроса. В случае прочих приложений экземпляр сохраняется в статическом свойстве класса LinqHelper на всё время жизни приложения.

При создании независимого от БД кода следует иметь в виду, что экземпляр контекстного класса, доступный как `LinqHelper.Context`, по умолчанию использует настройки, с которыми он был сгенерирован. Сменить эти настройки можно до первого обращения к `LinqHelper.Context` через изменение статических свойств контекстного класса.

В веб-среде существует возможность одновременного использования нескольких экземпляров по умолчанию. Условия использования:

1) экземпляры должны находиться в разных пространствах имён;
2) должны различаться имена сайтов, с которыми работают экземпляры.

Явное инстанцирование контекстного класса

При необходимости работы с несколькими контекстными классами можно использовать явный вызов конструктора. Если контекстные классы получены в режиме генерации кода, независимого от БД, то нужно использовать один из статических методов создания:

```
static QPDataContext Create(string connection, string siteName, MappingSource mappingSource)
static QPDataContext Create(string siteName, MappingSource mappingSource)
static QPDataContext Create(string connection, string siteName)
static QPDataContext Create(string connection)
static QPDataContext Create()
```

Параметры:

Название	Описание
connection	Строка подключения к БД
siteName	Имя сайта
mappingSource	Источник данных для отображения на основе MAP-файла, полученного в результате сборки контентов в классы LINQ to SQL

Если какой-либо из параметров не задан, то вместо него используются соответствующие статические свойства контекстного класса, которые можно изменить перед вызовом метода создания.

Особенности использования

В обычном сценарии использования классов LINQ to SQL сущностные классы используют экземпляр контекстного по умолчанию (`LinqHelper.DataContext`) в тех случаях, когда им требуется внутри себя подгрузить какие-либо данные (например, для реализации полей «Связь» типа M2M). Однако в случае явного инстанцирования контекстного класса это не будет работать, так как внешний и внутренний контексты будут отличаться. Поэтому у каждого экземпляра сущностного класса имеется свойство `InternalDataContext`, которое позволяет явно задать внутренний контекст для подобных операций. По умолчанию свойство инициализируется значением `LinqHelper.DataContext`.

Необходимость задавать внутренний контекст возникает при явном инстанцировании контекстного класса и использовании следующих операций:

Название	Описание
Добавление новых статей	**Внимание:** необходимо задать контекст до сохранения статьи. Экземпляр контекстного класса можно передавать прямо в конструкторе сущностного класса.
Все операции с полями «Связь» типа M2M (свойствами-коллекциями внутри сущностных классов)	Контекст необходимо задать у экземпляра сущностного класса перед обращением к его полю «Связь» типа M2M
Использование дополнительных свойств полей-изображений	Контекст необходимо задать у экземпляра сущностного класса перед обращением к дополнительному полю

```
EventsArticle event = new EventsArticle(context);
```

```
var item = ctx.TestArticles.Where(n => n.Id == 3737).Single();
var newCat = ctx.TestCategoryArticles.Where(n => n.Id == 1856).Single();
item.InternalDataContext = ctx;
item.Categories.Add(newCat);
ctx.SubmitChanges();
```

Если необходимо применение контекста сразу для коллекции объектов, то возможно использование метода, расширяющего `IEnumerable<T>` (`ApplyContext(DataContext context)`), определённого для каждого сущностного класса.

```
foreach (var item in ctx.PhotosArticles.Where(n => n.Photo !=
null).ApplyContext(ctx))
{
    // use item.PhotoUrl;
}
```

Использование XmlMappingSource

Источник данных для отображения обычно получают с помощью статического метода `XmlMappingSource.FromXml`, в который передается строка с XML-содержимым отображения. Само XML-содержимое можно получить с помощью одного из экземплярных методов класса `DBConnector`:

- `GetMapFileContents`,
- `GetDefaultMapFileContents`.

Пример:

```
XmlMappingSource map =
XmlMappingSource.FromXml(Cnn.GetDefaultMapFileContents(Cnn.GetSiteId("Sandbox
Net")));
XmlMappingSource map2 =
XmlMappingSource.FromXml(Cnn.GetMapFileContents(Cnn.GetSiteId("Sandbox Net"),
"EventsDataContext.map"));
```

Статические свойства контекстного класса

Статические свойства позволяют настроить значения по умолчанию, которые потом будут использованы при инстанцировании контекстных классов, в том числе и для `LinqHelper.Context` перед первым обращением к нему.

Свойство DefaultConnectionString

Строка подключения к БД. Инициализируется строкой из раздела `connectionStrings` конфигурационного файла веб-сайта (`web.config`). Если значение не задано, то используется значение по умолчанию (`qp_database`).

Свойство DefaultSiteName

Имя сайта. Инициализируется именем сайта, для которого были сгенерированы классы LINQ to SQL.

Свойство DefaultXmlMappingSource

Инициализируется следующим вызовом:

```
XmlMappingSource.FromXml(Cnn.GetDefaultMapFileContents(Cnn.GetSiteId("Имя сайта")))
```

В качестве имени указывается имя сайта, для которого были сгенерированы классы LINQ to SQL.

Пример изменения статических свойств:

```
QPDataContext.DefaultXmlMappingSource =
XmlMappingSource.FromXml(Cnn.GetDefaultMapFileContents(Cnn.GetSiteId("Sandbox Net
7")));
QPDataContext.DefaultSiteName = "Sandbox Net 7";
```

5.5.3. Использование поля «Связь» типа M2M в классах LINQ to SQL

В классах LINQ to SQL, генерируемых QP, в отличие от стандартных классов LINQ to SQL, существует поддержка «Связь» типа M2M. Они доступны в двух интерфейсах:

- поле-коллекция в сущностном классе, которая состоит из экземпляров связанного сущностного класса;
- класс, представляющий узловую таблицу.

Первый интерфейс обычно используется для простых выборок, добавления, удаления связанных статей. Второй интерфейс – для сложных выборок, а именно для фильтрации по полю «Связь» типа M2M на уровне SQL.

При написании LINQ-запроса с фильтрацией по полю «Связь» типа M2M на уровне SQL следует использовать следующий алгоритм:

- начать с узловой сущности,
- выполнить фильтрацию по одной связанной сущности,
- сделать проекцию по второй связанной сущности.

Пример:

```
IQueryable<Article> art = ctx.ArticlesCategories.Where(n => n.Category.Title ==
"main").Select(m => m.Article);
```

5.5.4. Поддержка стандартного поведения объекта «Publishing Container»

Фильтрация статей по умолчанию

По умолчанию фильтрация статей выполняется следующим образом:

- в Live-режиме выводятся статьи, которые:

1) видимы (`VISIBLE = 1`, флаг управляется расписанием видимости статьи);
2) не находятся в Архиве статей (`ARCHIVE = 0`);
3) имеют максимальный статус Workflow (если `STATUS_TYPE_ID` = ID статуса `Published` текущего сайта (в большинстве Workflow) или если Workflow не назначен);

- В Stage-режиме применяются только первые два условия.

В текущей версии QP данное поведение для классов LINQ to SQL настраивается через включение опции «Использовать фильтрацию по умолчанию» в свойствах контента. При включённой опции LINQ-класс отображается не на таблицу `CONTENT_`*`NNN`*, а на представление `CONTENT_`*`NNN`*`_LIVE` (для Stage-режима используется аналогичное представление `CONTENT_`*`NNN`*`_STAGE` вместо `CONTENT_`*`NNN`*`_UNITED`).

Примечание: в предыдущих версиях продукта реализовывалось через вызов метода расширения `ForFrontEnd()`. В текущей версии метод оставлен для обратной совместимости, но в случае включённой опции никаких действий не выполняет.

Примечание: единственное отличие от объекта «Publishing Container» в том, что фильтрация по статусу всегда осуществляется по системному статусу `Published`. Таким образом не поддерживаются Workflow, у которых максимальный статус не равен `Published`.

Если генерируемые классы LINQ to SQL не используются непосредственно на веб-сайте из директории `App_Code`, а сначала включаются в проект в Visual Studio, то для организации работы веб-сайта в Live- и Stage-режимах целесообразно воспользоваться режимом генерации, независимым от БД (настраивается в свойствах сайта). В этом случае один и тот же код можно использовать и в Live-, и в Stage-режиме, отличие будет только в содержимом MAP-файла привязки. При такой организации кода для Stage-режима потребуется задать `isLive = false` в секции `appSettings` конфигурационного файла веб-сайта (`web.config`).

Примечание: для Live-режима параметр `isLive` должен быть либо не задан, либо равен `true`.

5.5.5. Поддержка служебных полей контента

В любом сгенерированном сущностном классе есть постоянный набор служебных полей:

Название	Тип	Описание
Id	int	Идентификатор статьи. Изменения данного поля не сохраняются в БД, значение выставляется сервером автоматически.
StatusTypeId	int	ID статуса статьи. Изменения данного поля сохраняются в БД.
StatusType	StatusType	Статус статьи. Изменения данного поля сохраняются в БД.
Visible	bool	Флаг, определяющий видимость статьи. Для возможности работы с разными значениями данного поля необходимо отключить фильтрацию по умолчанию на уровне контента. Изменения данного поля сохраняются в БД.
Archive	bool	Флаг, определяющий нахождение статьи в архиве статей. Для возможности работы с разными значениями данного поля необходимо отключить фильтрацию по умолчанию на уровне контента. Изменения данного поля сохраняются в БД.
Created	DateTime	Дата создания статьи. Изменения данного поля не сохраняются в БД, значение выставляется сервером автоматически.
Modified	DateTime	Дата последней модификации статьи. Изменения данного поля не сохраняются в БД, значение выставляется сервером автоматически.
LastModifiedBy	int	Пользователь, последним модифицировавший статью. Изменения данного поля не сохраняются в БД. Модификации

		статей из классов LINQ to SQL выполняются от имени пользователя по умолчанию (администратора).

5.5.6. Дополнительные свойства полей «Изображение» и «Динамическое изображение»

Дополнительные свойства позволяют более удобно работать с полями типа «Изображение» и «Динамическое изображение» без прямого обращения к QP8 API. Названия свойств генерируются из названия поля изображения путём добавления к ним следующих значений:

Название	Описание
*Url	Возвращает URL файла изображения.
*Path	Возвращает физический путь к файлу изображения. Может использоваться при сохранении.

5.5.7. Классы LINQ to SQL и кэширование

У класса `DBConnector` существует обобщённый экземплярный метод `GetCachedEntity`.

```
public T GetCachedEntity<T>(string key, Func<T> fillAction)
public T GetCachedEntity<T>(string key, Func<string, T> fillAction)
public T GetCachedEntity<T>(string key, double cacheInterval, Func<T> fillAction)
public T GetCachedEntity<T>(string key, double cacheInterval, Func<string, T>
fillAction)
```

Если требуется закэшировать коллекцию объектов, полученных с помощью LINQ to SQL, то её сначала нужно сделать вещественной с помощью любого из методов:

1) `ToList`,
2) `ToArray`,
3) `ToDictionary`.

Пример использования:

```
DBConnector cnn = new DBConnector();
List<Banner> banners = cnn.GetCachedEntity<List<Banner>>(pageAddress, 10,
GetPageBanners);
```

В примере `GetPageBanners` – метод, возвращающий `List<Banner>`, который будет вызван только при заполнении кэша. Метод может быть определён как без параметров, так и с единственным строковым параметром. Во втором случае в метод будет передано значение ключа (параметр `key`).

5.5.8. Использование классов LINQ to SQL при создании компонентов

Под компонентами в данном случае понимаются небольшие обладающие определённой функциональностью части, которые могут быть повторно использованы. Каждая из этих частей обычно использует в качестве источника данных несколько связанных контентов, а значит может быть построена на основе группы классов LINQ to SQL, которые генерируются для этих контентов. Поскольку классы LINQ to SQL, используемые в компонентах, должны быть переносимыми, их нужно создавать только при включённой в свойствах сайта опции «Выполнять генерацию, независимую от БД».

Для выделения нескольких сущностных классов в одну группу нужно у нескольких контентов задать одинаковое значение в поле «Имя дополнительного контекстного класса». В качестве значения нужно указывать полное имя класса с учетом пространств имён. Таким образом, один и тот же контент может участвовать в двух контекстных классах:

1) основном, задаваемом на уровне сайта;

2) дополнительном, задаваемом на уровне контентов.

Примечание: для предотвращения конфликта имён пространства имён для всех генерируемых контекстных классов (как основного, так и дополнительных) должны различаться.

При использовании компонента, использующего классы LINQ to SQL, для сайта необходимо:

Название	Описание
Воссоздать структуру контентов	Также должны быть восстановлены данные об отображении в классы LINQ to SQL и значения свойств.
Сгенерировать .map-файл привязки классов	Так как классы генерировать не надо, то в свойствах сайта можно установить опцию «Генерировать только .map-файл».

5.5.9. Основные ошибки использования классов LINQ to SQL

- Многократное выполнение одних и тех же запросов без необходимости.
- Отсутствие понимания, что выполняется на веб-сервере, а что – на сервере СУБД.
- Получение от SQL-сервера лишних данных (аналог в SQL – `select *`), когда можно обойтись ограниченным набором полей с помощью `select new`.
- В конце цепочки операций LINQ в большинстве случаев желательно явно овеществлять результат (например, с помощью `ToList()`). Результат должен быть не `IQueryable<T>`, а `IEnumerable<T>` или `T`, иначе последующий `foreach` может привести к новому SQL-запросу на каждой итерации.

5.5.10. Пример использования

Добавление статьи:

```
NewsArticle art = new NewsArticle();
 art.Title = "some title";
art.Text = "some text";
LinqHelper.Context.NewsArticles.InsertOnSubmit(art);
LinqHelper.Context.SubmitChanges();
```

5.5.11. Настройка LINQ to SQL

Внимание: программа SQLMetal не входит в дистрибутив продукта.

В конфигурационный файл QP нужно добавить параметр `SqlMetalPath`, в качестве значения задать путь к программе SQLMetal (отвечает за генерацию SQL-классов). Программа устанавливается вместе с Visual Studio, путь к ней может выглядеть так: `C:\Program Files\Microsoft SDKs\Windows\v*n.nn*\bin\sqlmetal.exe`. Также программа доступна к составе Windows SDK.

Примечание: если в конфигурационном файле QP отсутствует параметр `SqlMetalPath`, то при генерации данных используется T4-шаблон.

Лог ошибок утилиты SQLMetal пишется в файл `sqlmetal.log` в директории `App_Data`. Ошибки обычно связаны с конфликтами имён.

Пользователю ОС, под которым запущен `Application Pool` (обычно это `NETWORK SERVICE`) необходимо дать право доступа «Modify» на директории:

1) `App_Code` (требуется для возможности генерации файлов классов),
2) `App_Data` (для генерации промежуточных файлов отображений).

В конфигурационный файл веб-сайта необходимо добавить ссылки на необходимые сборки:

```
<system.web>
      <compilation>
```

```
            <assemblies>
...
                    <add assembly="System.Data.Linq, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089" />
                    <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089" />
                    <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089" />
            </assemblies>
...
      </compilation>
</system.web>
...
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
        <providerOption name="CompilerVersion" value="v3.5" />
        <providerOption name="WarnAsError" value="false" />
      </compiler>
    </compilers>
  </system.codedom>
```

5.6. Использование Entity Framework

В QP поддерживается работа с контентами с использованием Entity Framework (версии 6.0 и выше). Реализованы доступы на чтение и запись.

Для работы с Entity Framework требуется установить следующие NuGet-пакеты:

1) `QP8.EntityFramework6`,
2) `Quantumart` (запись, работа с мета-данными).

5.6.1. Особенности генерации EF

- Выборочная загрузка полей.
- Поддерживаются асинхронные методы чтения.

  Примечание: поддержка асинхронной записи отсутствует в текущей версии продукта.

- Поддержка локализации.
- Быстрая (по сравнению с LINQ) запись за счёт группового обновления данных.
- Использование отдельных развязочных таблиц для связей M2M (вместо `Views`, обращающихся к единой таблице).
- Поддерживаются обратные (в том числе несимметричные) связи M2M для сценариев расщепления статей.

- Для связей можно не создавать в модели обратные свойства (например, на сущность «Регион» могут ссылаться несколько контентов, а «Регион» может не ссылаться ни на кого).
- В модель добавлены системные сущности:
  - `User`,
  - `UserGroup`,
  - `StatusType`.
- Возможна одновременная работа в режимах Live, Stage.
- Для пагинации используется синтаксис SQL2012 (OFFSET, а не row number).

5.6.2. Принципы работы

- Генерация всех необходимых классов и интерфейсов осуществляется в Visual Studio автоматически с помощью T4-шаблона.
- Используется подход POCO.
- Классы используют EF 6.
- Сгенерированное сопоставление (mapping) использует EF Fluent API.
- Процесс генерации настраивается с помощью отдельного файла конфигурации.

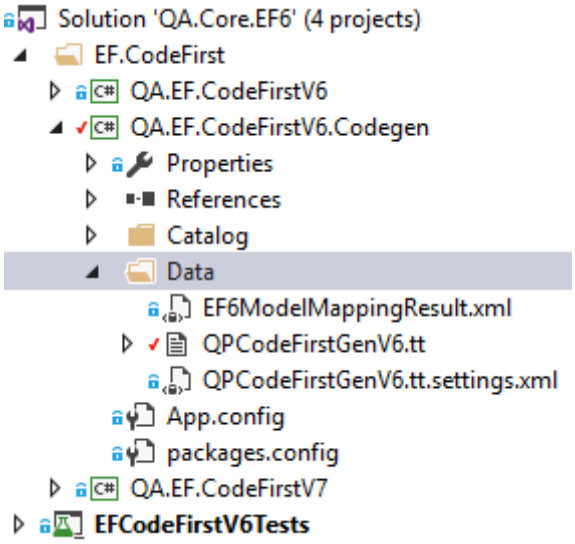

Рисунок 5.6.2-1. Структура решения в Visual Studio.

```
#region Equipment mappings
 modelBuilder.Entity<Equipment >()
     .ToTable(GetTableName("452"))
     .Property(x => x.Id)
     .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity)
     .HasColumnName("CONTENT_ITEM_ID");

modelBuilder.Entity<Equipment>()
        .HasMany<MarketingRegion>(p => p.Regions)
        .WithMany(r => r.BackwardForRegions12)
    .Map(rp =>
    {
        rp.MapLeftKey("id");
        rp.MapRightKey("linked_id");
        rp.ToTable(GetLinkTableName("103"));
    });
```

Рисунок 5.6.2-2. Пример содержимого для файла конфигурации.

5.6.3. Выборочная загрузка полей

Некоторые поля можно выделять в отдельные сущности.

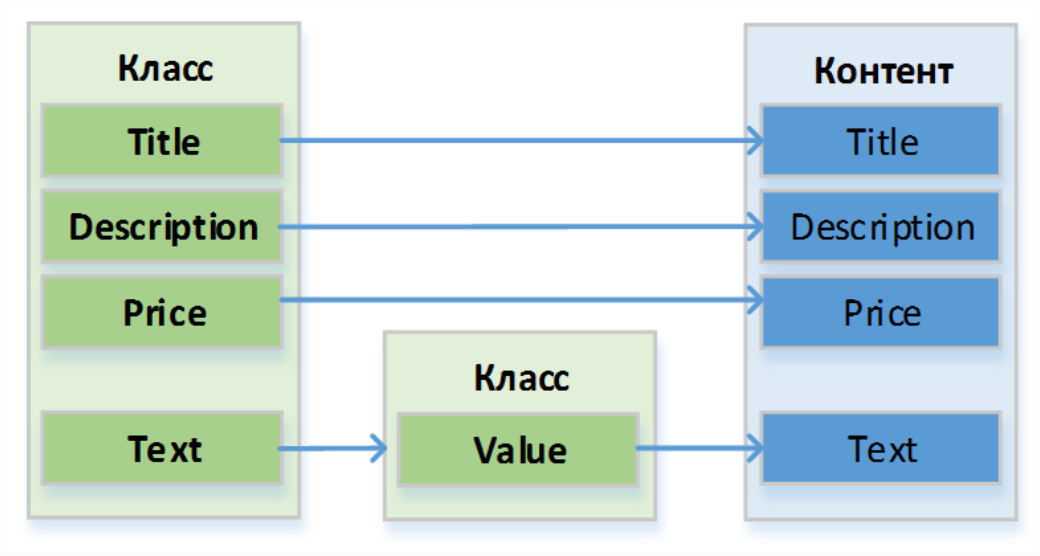

Рисунок 5.6.3-1. Выделение поля в отдельную сущность.

Загрузка значений может быть «ленивой» или явной с использованием `Include<T,TP>(…)`:

```
var included = m.Settings
    .Include(x => x.Text)
    .Where(z => z.Text.Value.Length > 10)
    .FirstOrDefault();
Assert.IsNotNull(included.Text);
Assert.IsTrue(included.Text.Value.Length > 10);
```

5.6.4. Подходы к локализации

Селективное сопоставление

- Используется конвенция именования полей.
- В класс не попадают поля других языков.
- Региональный стандарт (англ. «locale») указывается при создании экземпляра контекста.
- Сопоставление с полем требуемого регионального стандарта осуществляется в среде выполнения (англ. «runtime»).
- Подходит для сценариев чтения.
- Поддерживается добавление нового регионального стандарта без повторной компиляции.

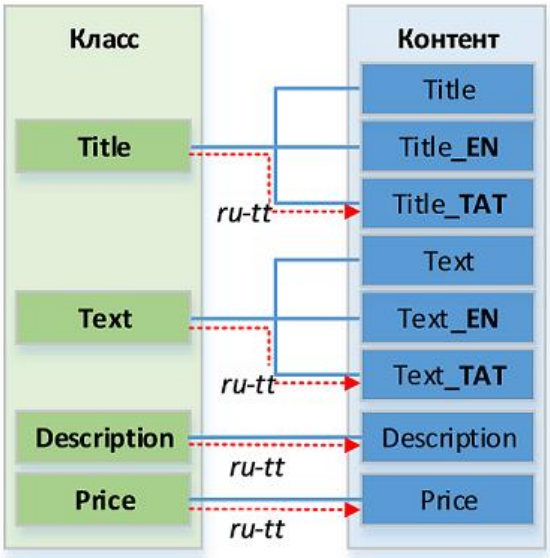

Рисунок 5.6.4-1. Сопоставление с полем требуемого регионального стандарта.

Установка нужного значения во время материализации сущностей

В модель включаются поля из других используемых региональных стандартов:

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
...
  <Localization>
    <UseSelectiveMappings>true</UseSelectiveMappings>
    <Pattern>{fieldName}_{cultureAlias}</Pattern>
    ...
    <CaseSensitive>true</CaseSensitive>
    <CultureMappings>
      <!--default-->
      <Map cultureAlias="" to="ru-ru" />
      <!--US english-->
      <Map cultureAlias="EN" to="en-us" />
      <Map cultureAlias="ENU" to="en-us" />
      <Map cultureAlias="TAT" to="ru-tt" />
    </CultureMappings>
  </Localization>
</settings>
```

5.6.5. Обратные свойства для поля «Связь» типа M2M

В случае использования LINQ to SQL возможно возникновение ситуации с некорректное чтением при расщеплении сущности. Например, если экземпляр «Регион1» сущности «Регион» расщеплён и привязан к экземпляру «Тариф1» сущности «Тариф»:

- в режиме Stage у загруженного из БД «Тариф1» не будет указана связь с «Регион1»,
- в режиме Live все связи будут присутствовать.

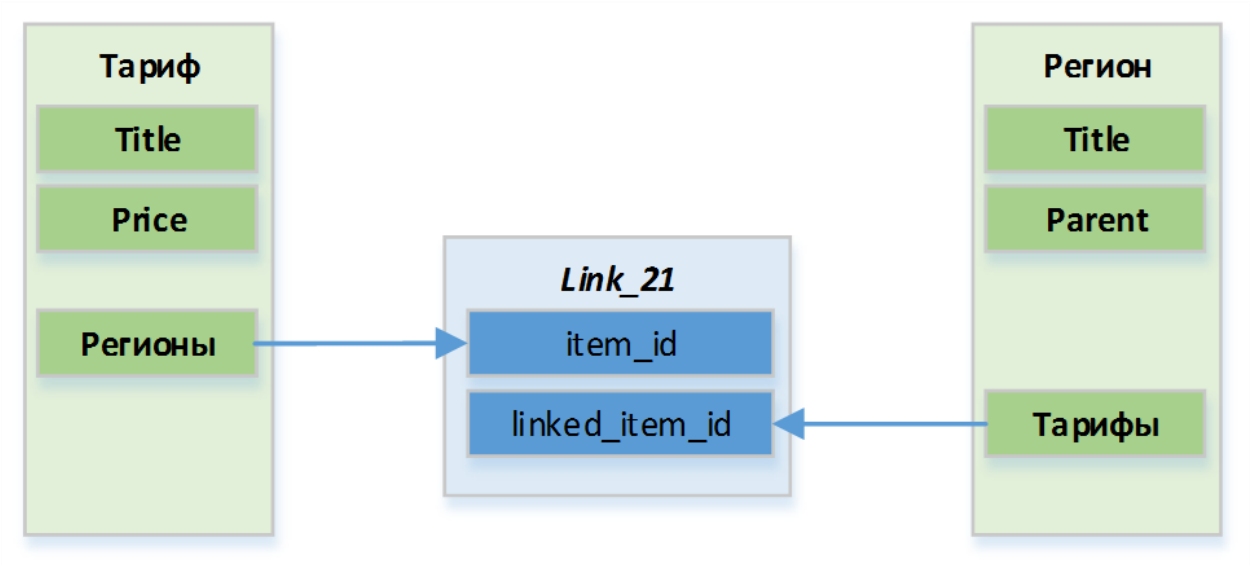

Рисунок 5.6.5-1. Связь сущностей (прямое представление).

Эта проблема решается добавлением дополнительного представления (англ. «View») для обратного поля. Шаблон EF позволяет генерировать сопоставление с использованием обратного представления.

По умолчанию обратные представления не используются. Для включения опции в свойствах поля «Связь» типа M2M требуется активировать свойство «Использовать отдельные обратные представления» (Use separate reverse views).

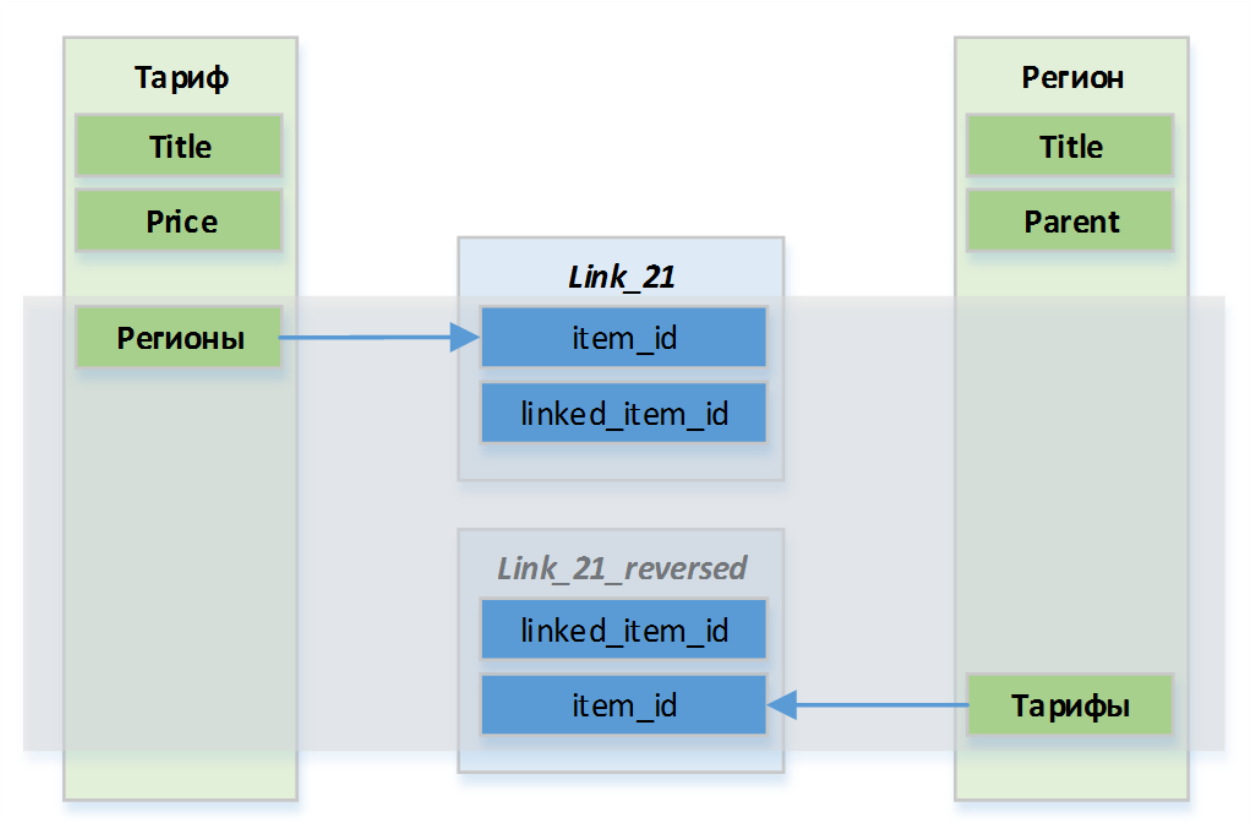

Рисунок 5.6.5-2. Прямое и обратное представления.

5.6.6. Порядок использования

Получение файлов сопоставления и шаблона

В результате сборки контентов (пункт «Собрать контенты» (Assemble contents) в контекстном меню сайта) QP8 генерирует файл с описанием структуры данных контентов выбранного сайта. Формат имени файла – *Имя контекстного класса*`MappingResult.xml`. Полученный файл следует добавить в проект в Visual Studio.

Примечание: в качестве имени контекстного класса по умолчанию используется значение `QPDataContext` (см. свойство «Имя контекстного класса» в разделе Параметры сборки в LINQ-классы (LINQ Assembling Parameters)).

Также в проект требуется добавить T4-шаблон `QPCodeFirstGenV6.tt`.

Примечание: файл шаблона содержится в пакете `QP8.EntityFramework6`.

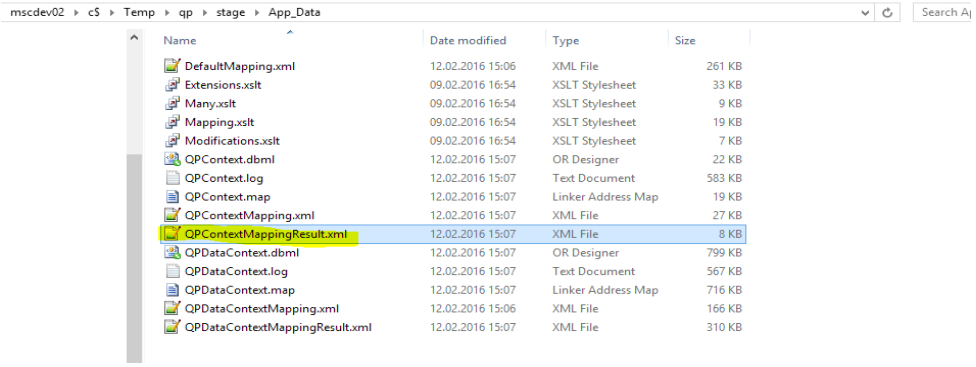

Рисунок 5.6.6-1. Файл с описанием структуры данных в директории QP.

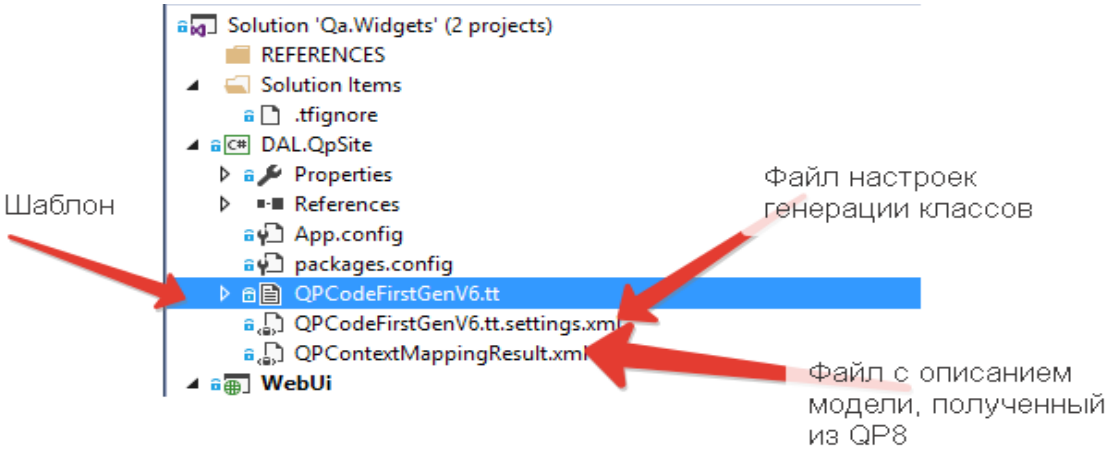

Рисунок 5.6.6-2. Используемые EF файлы в решении в Visual Studio.

Подключение файла с описанием структуры данных

В файле настроек в качестве значения параметра `QPContextMappingResultPath` требуется указать имя или путь до файла с описанием структуры данных, полученного из QP.

```
<settings>
  <QPContextMappingResultPath>EF6ModelMappingResult.xml</QPContextMappingResultPath>
  <GenerateModel>true</GenerateModel>
...
  <GenerateInterface>true</GenerateInterface>
...
  <UseContextNameAsConnectionString>true</UseContextNameAsConnectionString>
  <UseReversedAssociations>true</UseReversedAssociations>
  <ProxyCreationEnabled>false</ProxyCreationEnabled>
  <LazyLoadingEnabled>true</LazyLoadingEnabled>
  <Localization>
    ...
</settings>
```

Рисунок 5.6.6-3. Пример содержимого файла настроек.

Способы сопоставления данных

При использовании EF для работы с данными используются модели. В БД, содержащей данные QP, для запросов к таблицам с данными используются идентификаторы содержащих данные контентов. Когда в EF требуется получить список определённых объектов из QP, возникает задача по имени типа модели определить идентификатор контента. Это нужно, чтобы узнать название таблицы БД, к которой необходимо выполнить запрос для получения данных. Задача решается за счёт использования сопоставления имён типов модели и идентификаторов контентов. По этим данным EF формирует запросы к БД.

Способ использования сопоставления определяется в методе `GetDataContext`. В результате возвращается `DataContext`.

```
GetDataContext(ContentAccess access, Mapping mapping)
```

Существуют следующие способы сопоставления:

Название	Описание
Static Mapping	`DataContext` статический, создаётся на этапе автоматической генерации кода в проекте. В методе использовать `CreateWithStaticMapping(access)`. Применяется по умолчанию.
Database Mapping	`DataContext` динамический. В методе использовать `CreateWithDatabaseMapping(access,` *`DefaultSiteName`*`)`, где *`DefaultSiteName`* – имя сайта, содержащего необходимые данные.
File Mapping	`DataContext` динамический. В методе использовать `CreateWithFileMapping(access, GetPath(`*`DefaultMappingResult`*`))`, где *`DefaultMappingResult`* – путь до файла с описанием структуры данных из QP.

Static Mapping

Для способа Static Mapping применяется сопоставление, автоматически сгенерированное на основании исходных данных, жёстко заданное в коде проекта.

Способ подходит для случаев, когда Система на постоянной основе работает с одной БД в качестве источника данных.

В случае изменения БД потребуется повторная генерация кода проекта с использованием файла со структурой данных из QP, работающего с новой БД. Это необходимо для получения новых значений идентификаторов из QP.

Database Mapping

Для способа Database Mapping данные по идентификаторам при создании `DataContext` запрашиваются из БД, указанной в Connection String проекта. Получение данных осуществляется в среде выполнения. В случае изменения БД достаточно указать данные по новой БД в Connection String.

File Mapping

Для способа File Mapping данные по идентификаторам при создании `DataContext` запрашиваются из файла со структурой данных из QP, указанного в `CreateWithFileMapping(access, GetPath(`*DefaultMappingResult*`))` в качестве значения *DefaultMappingResult*. Получение данных осуществляется в среде выполнения. В случае изменения БД достаточно указать путь до файла со структурой данных из QP, работающего с новой БД.

5.7. Использование Entity Framework Core (отличия от Entity Framework)

В QP поддерживается работа с контентами с использованием Entity Framework Core (версии 2.2.4 и выше). Реализованы доступы на чтение и запись.

Для работы с Entity Framework Core требуется установить следующие NuGet-пакеты:

- `QP8.EntityFrameworkCore`
- `Quantumart.AspNetCore` (запись, работа с мета-данными).
- `Npgsql.EntityFrameworkCore.PostgreSQL` (если требуется работа с PostgreSQL)

5.7.1. Особенности генерации EF Core

Особенностями генерации EF Core можно выделить:

- Поддержка работы с PostgreSQL
- Для связей M2M генерируются отдельные классы, так как EF Core не поддерживает связи M2M
- Шаблоны для генерации и некоторые файлы *.cs обновляются (копируются) из папки NuGet пакета при каждой сборке проекта. Это необходимо для корректного обновления при обновлении версии пакета.
- Файлы классов данных EF core генерируются при каждой сборке проекта

5.7.2. Принципы работы

Генерация всех необходимых классов и интерфейсов осуществляется в Visual Studio автоматически с помощью T4-шаблона.

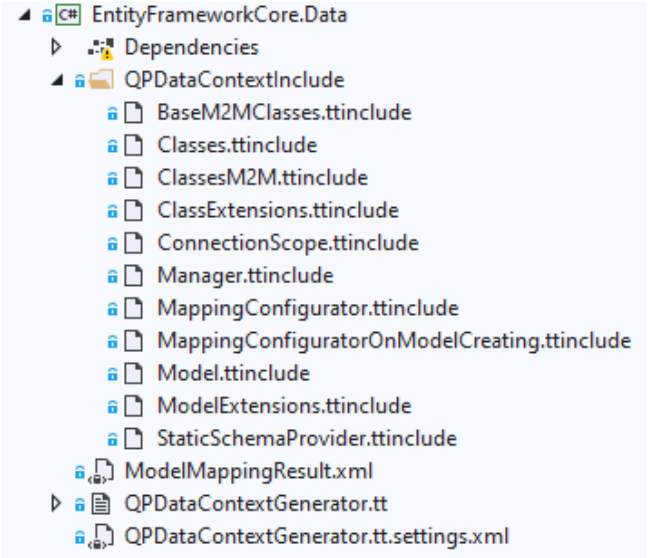

Рисунок 5.7.2-1 Структура решения в Visual Studio

5.6.3 Порядок использования

1. Получение файлов сопоставления и шаблона

В результате сборки контентов (пункт «Собрать контенты» (Assemble contents) в контекстном меню сайта) QP8 генерирует файл с описанием структуры данных контентов выбранного сайта. Формат имени файла – *Имя контекстного класса*`MappingResult.xml`. Полученный файл следует добавить в проекте в Visual Studio.

2. Подключение файла с описанием структуры данных

В файле настроек в качестве значения параметра `QPContextMappingResultPath` требуется указать имя или путь до файла с описанием структуры данных, полученного из QP.

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <!--Путь к файлу с описанием модели-->
  <QPContextMappingResultPath>ModelMappingResult.xml</QPContextMappingResultPath>
  <GenerateModel>true</GenerateModel>
  <GenerateClasses>true</GenerateClasses>
  <GenerateExtensions>true</GenerateExtensions>
  <GenerateInterface>true</GenerateInterface>
  <GenerateMappings>true</GenerateMappings>
  <GenerateMappingInterface>true</GenerateMappingInterface>

  <!--использовать имя класса модели в качестве имени строки подключения-->
  <UseContextNameAsConnectionString>true</UseContextNameAsConnectionString>
```

*Рисунок 5.7.2-2 При*мер содержимого файла настроек

Сгенерированные файлы *.cs с классами группируются под файлом QPDataContextGenerator.tt

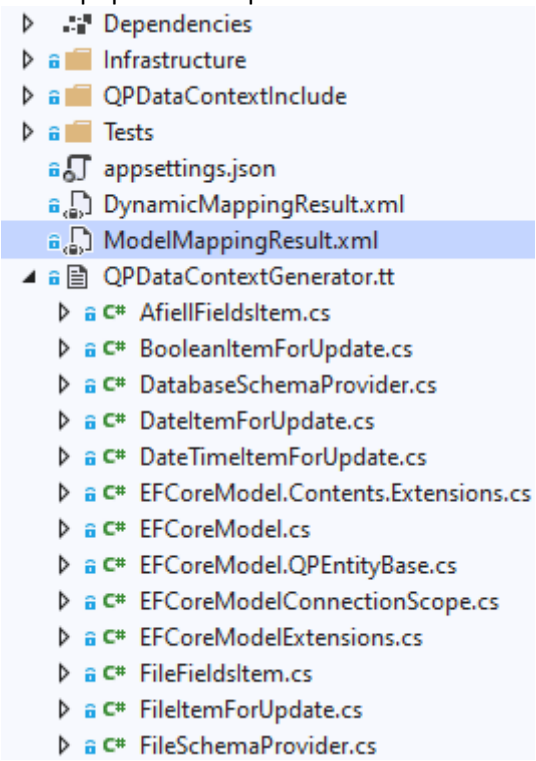

Рисунок 5.7.2-3 Пример структуры классов

Примечание: При возникновении проблемы генерации T4: «T4 Template Could not load file or assembly ‘System.Runtime, Version = 4.2.0.0’ необходимо отредактировать файл конфигурации Visual Studio согласно https://stackoverflow.com/questions/51550265/t4-template-could-not-load-file-or-assembly-system-runtime-version-4-2-0-0

5.8. JS-интеграция для веб-приложений пользовательских действий

Если стороннее веб-приложение запущено через пользовательское действие внутри бекэнда, то есть возможность взаимодействовать с бекэндом через JavaScript, даже когда веб-приложение и бекэнд работают на различных доменах. Двусторонний обмен данными реализуется библиотекой PMRPC (подробнее по ссылке) с помощью механизма HTML5 `postMessage`.

Для получения возможности интеграции требуется установить NuGet-пакет `QP8BackendApi.Interaction` (см. Установка NuGet-пакетов). Также возможно подключение пакета через npm: `npm i @quantumart/qp8backendapi-interaction`

Пример подключения:

```
import { SomeMethod } from "@quantumart/qp8backendapi-interaction";
```

Внимание: PMRPC в этот npm пакет не входит, его необходимо импортировать и подключать отдельно.

Таким образом, полное подключение может иметь следующий вид:

```
import "~wwwroot/js/pmrpc";

import { SomeMethod } from "@quantumart/qp8backendapi-interaction";
```

Для интеграции стороннего веб-приложения написанного на ASP.NET Core с QP, также необходимо подключить pmrpc. Это может быть сделано в .cshtml файле:

```
<script src="~/Scripts/pmrpc.js"></script>
```

В классе `Quantumart.QP8.Interaction` поддерживается ряд интеграционных методов, определённых, как статические.

5.8.1. Метод checkHost

Метод осуществляет проверку что веб-приложение выполняется внутри бекэнда.

Сигнатура:

```
function (hostUID, destination, callback)
```

Проверка, что веб-приложение выполняется внутри бекэнда.

Параметры:

Название	Описание
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`.
callback	Функция, в которую возвращается результат проверки. Определяет параметр `args`, в котором возвращаются: • `success` – результат проверки (Boolean), • `hostVersion` – версия бекэнда в случае успешной проверки (String), • `error` – текст ошибки в случае неуспешной проверки (String).

5.8.2. Метод executeBackendAction

Сигнатура:

```
function (executeOptions, hostUID, destination)
```

Выполнение действия в бекэнде.

Поддерживаются как интерфейсные, так и неинтерфейсные действия. Также есть возможность вызова пользовательских и многошаговых действий.

Параметры:

Название	Описание
executeOptions	Экземпляр `ExecuteActionOptions` (или объект с аналогичной структурой).
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`.

Класс ExecuteActionOptions

Экземпляр данного класса, передаваемый при вызове `executeBackendAction`, позволяет определить, какое действие должно быть выполнено и каким образом.

Возможности:

- определить, как будет вызываться действие:
  - в текущей вкладке,
  - в новой вкладке,
  - в окне бекэнда;
- вызвать форму создания новой сущности с предопределёнными значениями;
- Вызвать форму изменения сущности;
- Скрыть часть полей при создании/изменении сущности;
- Скрыть определённые кнопки в ГПИ бекэнда;
- Вызвать другое пользовательское действие, связанное с текущим веб-приложением, и передать туда дополнительные данные.

Структура:

Название	Тип	Описание
actionCode	String	Код вызываемого действия.
entityTypeCode	String	Код типа сущности, к которому относится вызываемое действие.
parentEntityId	Number	Идентификатор родительской сущности.
entityId	Number	Идентификатор сущности.
actionUID	String	Уникальный идентификатор действия. Задаётся веб-приложением. Использовать в случае, когда осуществляется управление несколькими интерфейсными действиями и в дальнейшем, получая от них `callback`, требуется возможность различать их.
callerCallback	Function	Ссылка на метод-обработчик в веб-приложении. Обычно определяется через `BackendEventObserver`.
changeCurrentTab	Boolean	Указатель, выполнять ли интерфейсное действие со сменой содержимого текущей вкладки ГПИ бекэнда. Значение по умолчанию – `false` (или открывать новую вкладку).
isWindow	Boolean	Указатель, выполнять ли интерфейсное действие в окне бекэнда. Значение по умолчанию – `false` (открывать во вкладке).

options		Экземпляр `ArticleFormState` (или объект с аналогичной структурой): • `initFieldValues` – значения для инициализации полей. Тип – `Array`. Элементом является экземпляр `ArticleFormState.InitFieldValue`, (или объект с аналогичной структурой): ○ `fieldName` – имя поля; ○ `value` – значение (зависит от типа). • `disabledFields` – идентификаторы полей, которые должны быть в состоянии `disable` (массив имён полей); • `hideFields` – идентификаторы полей, которые должны быть скрыты (массив имён полей); • `disabledActionCodes` – массив `Action Code`, для которых кнопки в ГПИ бекэнда должны быть скрыты. • `additionalParams` – дополнительные параметры для выполнения пользовательского действия, которые должны быть в него переданы через `QueryString`.

Класс ArticleFormState

Класс `ArticleFormState` содержит параметры для инициализации формы статьи и имеет следующую структуру:

Название	Тип	Описание
initFieldValues	Array	Содержит значения для инициализации полей
disabledFields	Array of strings	Содержит Id полей которые должны быть показаны, но доступны в режиме read-only (массив имен полей)
hideFields	Array of strings	Содержит Id полей, которые должны быть скрыты (массив имён полей)
disabledActionCodes	Number	Содержит массив `Action Code`, для которых кнопки в ГПИ бекэнда скрыты
additionalParams	Object	Содержит дополнительные параметры для выполнения пользовательского действия, которые должны быть в него переданы через `QueryString`

5.8.3. Метод closeBackendHost

Сигнатура:

```
function (actionUID, hostUID, destination)
```

Закрытие открытого ранее (с помощью `executeBackendAction`) интерфейсного действия.

Параметры:

Название	Описание
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.

actionUID	Уникальный идентификатор действия, ранее заданный веб-приложением. Тип значения – `String`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`.

5.8.4. Метод openSelectWindow

Открытие всплывающего окна для выбора значения с последующим возвратом результата выбора в веб-приложение.

Сигнатура:

```
function (openSelectWindowOptions, hostUID, destination)
```

Параметры:

Название	Описание
openSelectWindowOptions	Экземпляр класса `OpenSelectWindowOptions` (или объект с аналогичной структурой), определяющий действие выбора.
hostUID	Уникальный идентификатор вкладки текущего пользовательского действия. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`.

Класс OpenSelectWindowOptions

Класс `OpenSelectWindowOptions` имеет следующую структуру:

Название	Тип	Описание
selectActionCode	String	Код вызываемого действия выбора.
entityTypeCode	String	Код типа сущности, к которой относится вызываемое действие.
parentEntityId	Number	Идентификатор родительской сущности.
isMultiple	Boolean	Указатель, является ли действие выбора множественным Значение по умолчанию – `false`.
selectedEntityIDs	Array	ID сущностей, выбранные заранее. Элементы массива имеют тип `Number`.
selectWindowUID	String	Уникальный идентификатор окна выбора. Задаётся веб-приложением.
callerCallback	Function	Ссылка на метод-обработчик в веб-приложении. Обычно определяется через `BackendEventObserver`.

5.8.5. Метод BackendEventObserver

Необходимо использовать экземпляр класса, чтобы через `callback` получать результат взаимодействия с бекэндом.

Сигнатура:

```
function (callbackProcName, callback)
```

Регистрирует `callback` с именем `callbackProcName` в библиотеке PMRPC.

Метод `callback` должен иметь следующую сигнатуру:

```
function (eventType, args)
```

Параметры:

Название	Описание

eventType	Причина вызова `callback`. Принимает одно из возможных значений, определённых в `BackendEventObserver.EventType`: • `1` (вызванный хост был отсоединён); В этом случае в `args.reason` передаётся причина отсоединения (одно из двух значений, определённых в `BackendEventObserver.HostUnbindingReason`). • `2` (вызванное действие было выполнено); • `3` (сущности были выбраны); • `4` (окно выбора было закрыто).
args	Передаются дополнительные данные, специфичные для каждого действия. Например, при выборе сущностей передаются: • `selectWindowUID` – идентификатор окна, в котором произошёл выбор; • `selectedEntityIDs` – массив идентификаторов выбранных сущностей.

5.8.6. Метод openFileLibrary

Открытие всплывающего окна библиотеки файлов с последующим возвратом результата выбора в веб-приложение.

Сигнатура:

```
function (openFileLibraryOptions, hostUID, destination)
```

Параметры:

Название	Описание
openFileLibraryOptions	Экземпляр класса `openFileLibraryOptions` (или объект с аналогичной структурой), определяющий действие выбора.
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`

Класс OpenFileLibraryOptions

Класс `OpenFileLibraryOptions` содержит параметры сообщения на открытие окна библиотеки файлов и имеет следующую структуру:

Название	Тип	Описание
isImage	Boolean	Указывает на фильтрацию по типу файла – только изображения. Значение по умолчанию `false`.
useSiteLibrary	Boolean	Указывает использование библиотеки сайта или библиотеку контента. Значение по умолчанию `false`.
subFolder	String	Указывает подкаталог в каталоге библиотеки
libraryEntityId	Number	Значение `Id` контента при использовании библиотеки контента, либо `Id` сайта для библиотеки сайта
libraryParentEntityId	Number	Значение `Id` сайта для библиотеки контента, либо значение 0 для библиотеки сайта
selectWindowUID	String	Значние `Id` для идентификации окна со списком. Задаётся веб-приложением

callerCallback	Function	Ссылка на метод-обработчик в веб-приложении. Обычно определяется через `BackendEventObserver`.

5.8.7. Метод sendNotification

Метод осуществляет отправку Push-уведомления средствами QP, так как многие браузеры блокируют отправку через IFRAME.

Сигнатура:

```
function (sendNotificationOptions, hostUID, destination)
```

Параметры:

Название	Описание
sendNotificationOptions	Экземпляр класса `sendNotificationOptions` (или объект с аналогичной структурой), определяющий действие выбора.
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`

Класс SendNotificationOptions

Класс `SendNotificationOptions` содержит параметры сообщения на отправку уведомления и имеет следующую структуру:

Название	Тип	Описание
title	String	Заголовок сообщения уведомления
icon	String	Необязательная иконка уведомления
body	String	Необязательное тело сообщения

5.8.8. Метод previewImage

Метод осуществляет предпросмотр изображения, содержащегося в поле статьи.

Сигнатура:

```
function (previewImageOptions, hostUID, destination)
```

Параметры:

Название	Описание
previewImageOptions	Экземпляр класса `previewImageOptions` (или объект с аналогичной структурой), определяющий действие выбора.
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`

Класс PreviewImageOptions

Класс `PreviewImageOptions` содержит параметры просмотра изображения и имеет следующую структуру:

Название	Тип	Описание
entityId	Number	Значение Id сущности

fieldId	Number	Значение Id поля
fileName	String	Имя файла

5.8.9. Метод downloadFile

Метод позволяет скачать файл, содержащийся в поле статьи.

Сигнатура:

```
function (downloadFileOptions, hostUID, destination)
```

Параметры:

Название	Описание
downloadFileOptions	Экземпляр класса `downloadFileOptions` (или объект с аналогичной структурой), определяющий действие выбора.
hostUID	Уникальный идентификатор текущей вкладки ГПИ. Генерируется бекэндом, передаётся в пользовательское действие в виде одноимённого параметра `QueryString`.
destination	Окно, содержащее основное приложение бекэнда. Обычно нужно передавать `window.parent`

Класс DownloadFileOptions

Класс `DownloadFileOptions` содержит параметры скачивания файла и имеет следующую структуру:

Название	Тип	Описание
entityId	Number	Значение Id сущности
fieldId	Number	Значение Id поля
fileName	String	Имя файла

5.9. Кастомизация форм QP

В QP скрипты задаются в двух уровнях: глобальное поле на уровне сайта и поле на уровне контента. На уровне сайта поле глобальное и применяется ко всем формам, а на локальном уровне контента применяется только на уровне текущего контента. Скрипт позволяет кастомизировать форму под необходимые требования.

5.9.1. Базовые возможности

Существует возможность добавления собственных функциональных возможностей в форму создания и изменения статей. Примеры возможных вариантов:

- скрытие или показ полей в зависимости от значения других полей,
- использование клиентской валидации,
- добавление пользовательских кнопок с требуемым поведением.

В свойствах сайта для подключения собственных скриптов используется свойство «Скрипт для формы контента». Заданные скрипты будут добавлены в форму для текущего сайта.

Скрипты

Скрипт для формы контента: Скрыть

```
1 QP_CURRENT_CONTEXT.loadScript("/plugins/[redacted]dpc.js");
2 QP_CURRENT_CONTEXT.loadScript("/plugins/[redacted]'translit.js");
```

Рисунок 5.9.1-1. Скрипт для формы контента.

В свойствах контента используется свойство «Скрипт для формы». Скрипт используется в форме для текущего контента.

Клиентские скрипты

Скрипт для формы: Скрыть

```
1 QP_CURRENT_CONTEXT.typeResolverInputName = 'field_1233';
2
3 QP_CURRENT_CONTEXT.typeFilters = {
4       "field_1295" : "",
5       "field_1363" : "",
6       "field_1365" : "",
7       "field_1667" : ""
8  };
```

Рисунок 5.9.1-2. Скрипт для формы.

Параметр `QP_CURRENT_CONTEXT` заменяется на значение для текущего контекста (компонента, создаваемого для вкладки или окна с редактором).

Методы и поля объекта контекста

```
setGlobal(key, value)
```

Установка глобальной переменной (уровня бекэнда).

Полезно при задании переменной во внешнем скрипте и чтении из скрипта, определяемого в бекэнде.

```
getGlobal(key)
```

Чтение глобальной переменной (уровня бекэнда).

`loadScript(url, key)`

Загрузка внешнего скрипта с проверкой на повторную загрузку по имени или по ключу.

Примечание: параметр `key` необязателен.

`addCustomButton(settings)`

Добавление пользовательской кнопки для поля с указанным именем.

Примечание: кнопка добавляется справа от поля.

Передаваемый объект поддерживает следующие поля:

Название	Описание
name	Имя поля формы, к которому добавляется кнопка.
title	Имя кнопки. Используется в ГПИ.
suffix	Суффикс для формирования идентификатора кнопки на основании идентификатора поля.
class	CSS-класс кнопки. **Примечание:** необязательный параметр. **Примечание:** поскольку `class` является зарезервированным словом в JavaScript, его нужно задавать в кавычках или апострофах, иначе возможна некорректная работа (например, в IE8).
url	URL пиктограммы для кнопки. **Примечание:** необязательный параметр.
onClick	Обработчик события `click`. Назначается на кнопку с помощью стандартных средств jQuery. В качестве дополнительных параметров обработчика в него передаются: • `settings` – исходные настройки кнопки; • `$input` – jQuery-обёртка элемента ГПИ, к которому привязана кнопка; • `$form` – jQuery-обёртка формы.

`addCustomLinkButton(settings)`

Добавление пользовательской кнопки для поля с указанным именем.

Примечание: кнопка LinkButton добавляется над полем.

Назначается на кнопку с помощью стандартных средств jQuery. В качестве дополнительных параметров обработчика в него передаются те же поля, что и в методе `addCustomButton`.

`initHandler`

Поле, задающее пользовательский обработчик события `init`. Срабатывает как для основной формы, так и при инициализации полей, относящихся к классификатору.

В качестве параметров метода передаются:

Название	Описание
editor	Родительский компонент `BackendEntityEditor`.
$elem	Обёртка jQuery для родительского элемента (формы или контейнера для классификатора).

`disposeHandler`

Поле, задающее пользовательский обработчик события `dispose`.

Передаваемые параметры идентичны параметрам метода `initHandler`.

`beforeSubmitHandler`

Поле, задающее пользовательский обработчик события `beforeSubmit`. В качестве параметров метода передаётся параметр `editor` – родительский компонент `BackendEntityEditor`.

`fieldValueChangedHandler`

Поле, задающее пользовательский обработчик события `fieldValueChanged`. Срабатывает в тех же случаях, что и в механизме автоматического сохранения форм.

В качестве параметров метода передаются:

Название	Описание
editor	Родительский компонент `BackendEntityEditor`.
data	Объект для автоматического сохранения. Содержит поля: • `fieldName` – имя поля, • `value` – значение поля.

5.9.2. Translit.js

Скрипт используется для транслитерации полей формы. Поставляется вместе с QP, может быть подключен в свойствах сайта, подробнее в разделе Базовые возможности.

Метод	Описание
`addTransliterateButton(srcInputName, dstInputName)`	При выполнении добавляет кнопку `Transliterate` к `dstInputName`, при нажатии на которую берет текст из `srcInputName`, транслитерирует и вставляет содержимое в `dstInputName`

5.9.3. Dynamic.js

Скрипт используется для добавления динамического поведения полям формы. Поставляется вместе с QP, может быть подключен в свойствах сайта, подробнее в разделе Базовые возможности.

Добавление обработчиков

Название	Описание
`addInitHandler(callback)`	Добавляет обработчик на событие инициализации формы
`addValueChangedHandler(callback)`	Добавляет обработчик на событие изменения значения поля формы

Резолверы

Примечание: под резолвером здесь и далее понимается элемент формы, который позволяет вычислить состояние, влияющее на другие элементов формы. При этом данные, определяющие состояние, могут как быть доступными в рамках текущей формы, так и вычисляться из связанных данных, уже охраненных в БД ранее.

Для включения функциональности динамических резолверов необходимо вызвать метод `initDynamicResolvers()`.

При использовании `widgets.js` данный метод вызывается в `initWidgetSystem`.

Используемые свойства

Название	Описание
`fieldsToHide`	Массив полей, которые должны быть скрыты. Пример:

	`QP_CURRENT_CONTEXT.fieldsToHide = ['OldSiteId', 'OldCorpSiteId', 'OldPointId', 'OldCorpPointId']`
`resolverNames`	Массив имен резолверов, объявляемых в рамках редактора. Если не задано, то считается состоящим из одного элемента с именем `Type`. **Внимание:** Эти имена используются в остальных свойствах как часть имени и обозначаются в дальнейшем как `nnn`. Пример: `QP_CURRENT_CONTEXT.resolverNames = ["Type", "Product", "BaseParameter"];`
`parentNnnFieldName`	Для резолвера `Nnn` задается имя поля родительской статьи. Если для резолвера достаточно информации в рамках текущего редактора, то данное свойство задавать не требуется. Пример: `QP_CURRENT_CONTEXT.parentTypeFieldName = "Type";`
`parentNnnClassifierFieldName`	Для резолвера `Nnn` задается имя классификатора родительской статьи. Пример: `'parent' + resolverName + 'ClassifierFieldName';`
`parentByClassifierNnnFieldNames`	Для резолвера `Nnn` задаются имя итоговых полей родительской статьи, если они находится в контентах-расширениях. При этом задается коллекция ключ-значение, где ключом является числовое значение классификатора, а значением – имя поля из соответствующего контента-расширения.
`nnnResolverInputName`	Имя поля для резолвера `Nnn` (обратите внимание на Camel-нотацию). Пример: `QP_CURRENT_CONTEXT.productResolverInputName = 'Product';`

`nnnFilters`	Коллекция, в котором ключам является имя свойство, а значением тип фильтрации. При передаче в качестве значения пустой строки, производится фильтрация по M2M. При непустом значении будет выполнена фильтрация по O2M по указанному в значении имени поля.
	
	Пример:
	`QP_CURRENT_CONTEXT.typeFilters = {`
	`"BaseParameter" : "",`
	`"Modifiers" : "",`
	`"ProductGroup" : "",`
	`"Group" : "",`

	`};`

Логика фильтрации:

1) Для каждого резолвера из `resolverNames` вычисляется его значение в соответствии со свойствами `nnnResolverInputName`, `parentNnnFieldName`, `parentByClassifierNnnFieldNames`, `parentNnnClassifierFieldName`.
   a. Берется значение поля с именем, заданным в свойстве `nnnResolverInputName`. Если не заданы другие свойства, относящиеся к резолверу `nnn`, то это значение используется в качестве результата вычисления, иначе вычисление продолжается, используя результат данного шага.
   b. Если задано свойство `parentNnnFieldName`, но не заданы свойства `parentByClassifierNnnFieldNames`, `parentNnnClassifierFieldName`, то в родительском контенте ищется значение поля, указанного в `parentNnnFieldName` для `Id` статьи, заданного результатом предыдущего шага. Если в `parentNnnFieldName` передано числовое значение, то это оно используется как `Id` связи, и ищутся все соответствующие M2M для `Id` статьи, заданного результатом предыдущего шага. В итоге если возвращаются либо значения поля родительской статьи, либо список `id` статей, либо если задано свойство `parentNnnClassifierFieldName`, то вычисления данного пункта пропускаются и с результатом пункта a переходим к пункту c.
   c. Если задано свойство `parentNnnClassifierFieldName`, то вычисляется значение родительского классификатора и ID статьи-расширения (предполагается фиксированное название аггрегированного поля - `Parent`). Если не задано `parentByClassifierNnnFieldNames`, то в статье-расширении ищется поле `parentNnnFieldName`, иначе из коллекции `parentByClassifierNnnFieldNames`, определяется имя поля, соответствующее текущему классификатору. В любом случае в итоге мы получаем значение поля из статьи расширения, которое возвращается как результат вычисления.
2) Если резолвер относится к предопределенному типу `Type`, то осуществляется дополнительный маппинг значения резолвера с использованием глобального объекта `typesByProduct`, иначе данный шаг пропускается.
3) Для резолвера определяем его набор фильтров из свойства `nnnFilters`
4) Для каждого фильтра из набора выполняем метод `setFilter`, передавая следующие параметры:

Название	Описание
`inputName`	Имя фильтра из набора
`value`	Значение резолвера
`fieldname`	Значение фильтра из набора
`$form`	Обёртка jQuery для родительского элемента (формы или контейнера для классификатора)

Простые методы

Во всех методах поддерживаются как внутренние имена полей (формата `field_nnn`), так и пользовательские (имена полей в контенте). В качестве `editor` можно передавать как экземпляр класса `BackendEntityEditor`, так и обертку jQuery для родительского элемента (формы или контейнера для классификатора)

Метод	Описание

changeFieldState(editor, data, $rootElem)	Основной метод для задания пользовательской логики обработки формы, предоставляемый `dynamic.js`. Он вызывается при изменении значения полей формы (в этом случае в параметре `data` передается информация об измененном поле, `$rootElem - null`), либо при первичной инициализации (в поле `data` передается `null`, `$rootElem` – обёртка jQuery для родительского элемента (формы или контейнера для классификатора)).
getInput(editor, inputName)	Получение элемента формы с именем `inputName`
getRow(editor, inputName)	Получение строки документа, в которой находится элемент формы с именем `inputName`
getText(editor, inputName)	Получение заголовка для значения элемента формы с именем `inputName` (для O2M)
getIds(editor, inputName)	Получение значения элемента формы с именем `inputName` (список ID для M2M)
getValue(editor, inputName)	Получение значения элемента формы с именем `inputName` (скалярного)
setValue(editor, inputName, value)	Установка значения `value` в элемент формы с именем `inputName`
applyFilter(inputName, filter, editor)	Применение фильтра `filter` к списочному полю формы с именем `inputName`
setFilter(inputName, value, fieldName, editor)	Формирование фильтра для списочного элемента формы с `inputName` на основе значения value и имени поля `fieldname`. При пустом имени поля формируется фильтр по M2M.
toggleField(editor, inputName, state, preserveHidden)	Изменить состояние видимости поля `inputName` в соответствии с параметром `state`. По умолчанию – обнуляет скрываемое поле. Это можно изменить, передав `preserveHidden – true`)
wrapFields(editor, inputNames, title)	Оборачивает набор полей `inputNames` редактора `editor` в `FieldSet` с заголовком `title` Пример `QP_CURRENT_CONTEXT.wrapFields(editor, ["BaseParameter", "Zone", "Direction", "BaseParameterModifiers"], "Тарифное направление");`

5.10. Использование объектов QP

Внимание: данный подход является устаревшим.

В шаблон полностью вносится вёрстка для страницы (всё содержимое в тэге `<html>`).

1. Осуществляется разбиение вёрстки на блоки. Блоки оформляются в виде объектов шаблона и заменяются в теле шаблона вызовами требуемых объектов.
2. Подобная декомпозиция применяется к созданным объектам шаблона до нужного уровня. Нужный уровень определяется наличием динамического содержимого или необходимостью переопределения.

3. Если объект имеет динамическое содержимое, то создаётся объект типа «Publishing Container», связанный с контентом для данного содержимого.
4. При создании других страниц, основанных на том же шаблоне, можно как создавать новые объекты страницы, так и создавать их, переопределяя требуемые объекты шаблона.

6. Пример создания Системы

6.1. Постановка задачи. Определение требований

В примере рассматривается создание для веб-сайта одной страницы, содержащей одну зону для размещения виджета.

Для создания веб-сайта следует использовать QP и Виджетную платформу.

Дополнительная разметка для страницы не требуется.

Разработка выполняется в Visual Studio.

6.2. Проектирование

6.2.1. Определение групп пользователей

Доступ к странице не ограничивается по группам пользователей. Таким образом, не потребуется выполнения работ с пользователями и группами пользователей для веб-сайта.

6.2.2. Проектирование структуры данных

Для выполнения задачи не понадобится разработка дополнительных виджетов.

В бекэнде понадобится создать статьи в контентах «ItemDefinition» и «AbstractItem», также сгенерировать классы LINQ to SQL.

6.2.3. Проектирование ГПИ

Внимание: для использования на странице ГПИ Виджетной платформы в разметку следует добавить код для вызова ГПИ.

Для страницы достаточно минимально необходимой разметки, содержащей одну зону для размещения виджета.

6.3. Реализация

6.3.1. Подготовка структуры данных

Примечание: общие сведения о процедуре создания страницы приведены в разделе Процедура создания и добавления нового типа страницы в Систему.

Создание типа страницы

В контенте «ItemDefinition» создана статья со следующим содержимым:

Название поля	Значение поля
Название (Title)	Стартовая страница
Идентификатор (Name)	start_page
Description	Корневая страница
Страница (IsPage)	Флаг установлен
Контент (PreferredContentId)	`0` (контент-расширение не требуется для решения задачи)
FullName (FriendlyDescription)	Значение не задано (не требуется для решения задачи)
Категория (CategoryName)	

IconUrl	

После создания статью следует опубликовать.

Создание экземпляра страницы требуемого типа

В контенте «AbstractItem» создана статья, для которой в качестве значения поля `Тип страницы/виджета` (Discriminator) задана ранее созданная статья «Стартовая страница».

После создания статью следует опубликовать.

Примечание: создание статьи можно осуществить как с помощью ГПИ QP, так и ГПИ Виджетной платформы (пользовательское действие «Manage pages»).

Подготовка данных в QP для разработки в Visual Studio

Для сайта, в котором размещены статьи, сгенерированы требуемые для разработки данные (классы LINQ to SQL и/или сопоставление для Entity Framework). Полученные файлы скопированы в решение в Visual Studio, в проекте из которого далее должно быть создано веб-приложение.

6.3.2. Разработка кода веб-приложения

Для данной задачи создание модели не требуется.

Примечание: файл с кодом модели требуется размещать в проекте в директории `Models`.

Файл с кодом представления требуется размещать в проекте в директории `Views`. Для страницы создана вложенная директория `StartPage`, в ней размещён файл `Index.cshtml` со следующим кодом:

```
@{
    ViewBag.Title = Model.Title;
}
<div class="content">
    @Html.Raw(Model.Text)
</div>
```

Файл с кодом контроллера требуется размещать в проекте в директории `Controllers`. Создан файл `StartPageController.cs` со следующим кодом:

```
using System;
using System.Web.Mvc;
using QA.Core.Engine;
using QA.Core.Engine.Web.Mvc;
using System.Threading.Tasks;
using Main.PageModel.Pages;

namespace Main.WebUi.Controllers
{
    [Controls(typeof(Main.PageModel.Pages.StartPage))]
    public partial class StartPageController :
ContentController<Main.PageModel.Pages.StartPage>
```

```
    {
        [NonAction]
        public override ActionResult Index()
        {
            throw new InvalidOperationException();
        }

        public async Task<ActionResult> Index(string id)
        {
            await Task.Yield();
            var mode = CurrentItem.Mode;

            if (mode == StartPageMode.Content)
            {
                return View(CurrentItem);
            }
            else
            {
                // в этом режиме стартовая страница должна наследовать
функциональность страницы-перенаправления
                throw new NotImplementedException("The mode Redirect is not
implemented yet.");
            }
        }

    }
}
```

6.3.3. Создание ГПИ

Файл с кодом разметки требуется размещать в проекте в директории `Views`. Создана вложенная директория `Shared`, в ней размещён файл `_Layout.cshtml` со следующим кодом:

```
@using QA.Engine.Extensions.Html
@using QA.Infrastructure.Web.Mvc.Extensions
@{
    var zoneState = ZoneExtensions.GetZoneState(Context);
}
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
    <title>@ViewBag.Title</title>
    @Scripts.Render(BundleConfig.JQuery)
    @*это нужно только для onscreen*@
    @if (zoneState != ZoneState.Disabled)
    {
        @Styles.Render(BundleConfig.JQueryUIStyles)
        @Scripts.Render(BundleConfig.JQueryUI)
    }
    @RenderSection("headScripts", required: false)
</head>
<body>
    @{Html.ControlPanel().Render();}
    @*скрипты для панели, скрипты для Onscreen*@
    @if (zoneState != ZoneState.Disabled)
    {
        @Scripts.Render(BundleConfig.OnScreen);
        @Styles.Render(BundleConfig.OnScreenCSS);
        <script type="text/javascript">
            // workaroud
            var ContentIdToEdit = 537;
        </script>
    }

    <div class="g-page-wrapper">
        <div class="footer__to-bottom-wrap">
            <div class="footer__to-bottom-content">
                <div class="b-wrapper">
                    <H2>@ViewBag.Title</H2>
                    @{ Html.WidgetZone("ZoneName").Render(); }
                    @RenderBody()
                </div>
            </div>
        </div>
    </div>
    @Scripts.Render("~/bundles/bootstrap")
    @Scripts.Render(BundleConfig.QAScripts)
    @RenderSection("scripts", required: false)
</body>
```

```
</html>
```

6.3.4. Размещение виджета

В бекэнде с использованием ПУ виджетной платформы (пользовательское действие «Manage widgets») на страницу добавлен виджет типа «HTML-виджет» и задано его содержимое.

В результате при обращении к странице через веб-браузер выводится содержимое, заданное при добавлении виджета на страницу.