ООО «КВАНТУМ АРТ»

115184, Москва, Озерковский переулок, д. 12

тел. (495) 783-65-74

QP8.ProductCatalog

Руководство пользователя

Москва
2025

Назначение документа

Документ содержит описание Системы управления структурированными данными «Продуктовый каталог». Система является расширением CMS QP8.

История изменений

Версия	Дата	Автор	Описание
2.0	27.03.2025	Молькова М.Е.	Добавлено описание интеграции с Camunda
1.9	06.06.2024	Селю П.Н.	Добавлено описание интеграции с MinIO
1.8	01.03.2024	Селю П.Н.	Добавлено описание установки демо-сайта
1.7	07.12.2023	Молькова М.Е.	Обновлено: • Методы поиска • Параметры и режимы поиска • Механизм переиндексации
1.6	02.06.2023	Тимофеев Е.А.	Добавлено: • Инструкция по установке на Linux • Инструкция по установке с использованием Docker • Инструкция по установке с использованием Kubernetes
1.5	08.09.2020	Филинова А.С.	Добавлено: • Принцип работы редактора описаний • Настройки конфигурационных файлов
1.4	21.03.2019	Комаров В.В.	Добавлено: • Параметры take/skip для пагинации, пример использования • Добавлено описание получения данных по справочникам • Настройка пользовательских методов
1.3	05.02.2019	Корякин С.А.	Добавлено: • Руководство по установке
1.2	08.02.2018	Советкали Б.С.	Обновлено: • Защита от ошибочных публикаций • Добавлено описание параметра, отвечающего за сохранение истории продукта (см. Источники данных функционала)
1.1	24.01.2018	Советкали Б.С.	Добавлено: • Защита от ошибочных публикаций • Версии продуктов • Состояние витрины на указанную дату Обновлено: • Добавлено описание поля date в контенте Индексы Elastic

			• Добавлено описание поля «Дата» функционала «HighloadFront»
1.0	20.11.2017	Советкали Б.С.	Первичное техническое описание

Оглавление

1. ОБОЗНАЧЕНИЯ 7

2. ОПРЕДЕЛЕНИЯ И ТЕРМИНЫ, ИСПОЛЬЗУЕМЫЕ В ДОКУМЕНТЕ 8

2.1. Общепринятые термины 8
2.2. Термины Продуктового каталога 8
2.3. Термины QP 8
2.4. Роли пользователей Продуктового каталога 9

3. ВВЕДЕНИЕ 10

4. ВХОД В DPC 11

4.1. Средствами веб-браузера 11
4.2. Вход в CMS средствами программы входа в операционную систему Windows WinLogon 11

5. ВЫХОД ИЗ DPC 12

6. АРХИТЕКТУРНАЯ СХЕМА DPC 13

6.1. Backend Server 13
6.2. Internal DPC Services 13
6.3. DB server 14
6.4. Elasticsearch Cluster 14
6.5. External DPC Services 14
6.6. DPC Consumers 14
6.7. DPC Integrations 14

7. ОСНОВНЫЕ КОМПОНЕНТЫ DPC 15

7.1. Административный модуль (DPC.Admin) 15
7.1.1. Продукты 15
7.1.2. Пользовательские действия 29
7.1.3. Описание пользовательских действий 31
7.1.4. Пользовательская валидация 52
7.1.5. Настройки конфигурационного файла 54
7.2. Сервис выполнения отложенных задач (DPC.ActionsService) 56
7.2.1. Мои задачи 57
7.2.2. Все задачи 61
7.2.3. Фильтрация задач 61
7.2.4. Расписание задач 62
7.2.5. Настройки конфигурационного файла 63
7.3. Сервис отправки продуктов на витрины (DPC.NotificationSender) 66
7.3.1. ГПИ службы. Функциональные возможности ГПИ 68
7.3.2. Управление Каналами 69
7.3.3. Настройка канала публикации 70
7.3.4. Запросы к Каналу 73
7.3.5. Лог работы службы 73
7.3.6. Очередь публикации 73
7.3.7. Защита от ошибочных публикаций 74
7.3.8. Настройки конфигурационного файла 74
7.4. DPC Web API (DPC.Api) 76
7.4.1. Описание API 76

7.4.2. Настройки конфигурационного файла ... 96
7.5. ELASTIC API – ВИТРИНА ДЛЯ ЗАПИСИ В ELASTIC (DPC.SYNC) ... 99
7.5.1. Общие сведения ... 99
7.5.2. Интерфейс индексации ... 99
7.5.3. Состояние витрины на указанную дату ... 100
7.5.4. Настройки конфигурационного файла ... 100
7.6. ELASTIC API – ВЫСОКОПРОИЗВОДИТЕЛЬНАЯ ВИТРИНА (DPC.SEARCH) ... 104
7.6.1. Общие сведения ... 104
7.6.2. Методы поиска ... 105
7.6.3. Параметры и режимы поиска ... 111
7.6.4. Аутентификация ... 122
7.6.5. Лимиты запросов ... 122
7.6.6. Кэширование ... 123
7.6.7. Быстродействие и отказоустойчивость ... 123
7.6.8. Настройки конфигурационного файла ... 124
7.7. РЕФЕРЕНСНАЯ ВИТРИНА (DPC.FRONT) ... 127
7.7.1. Настройки конфигурационного файла ... 130

8. ИНТЕГРАЦИЯ С CAMUNDA ... 132

8.1. ОБЩИЕ СВЕДЕНИЯ ... 132
8.2. ПОДДЕРЖИВАЕМЫЕ ЗАДАЧИ ... 132
8.2.1. Отправка продукта на Stage ... 132
8.2.2. Публикация продукта на Live ... 133
8.2.3. Проверка публикации на высокопроизводительной витрине ... 134
8.2.4. Проверка публикации на референсной витрине ... 137

9. ВИРТУАЛЬНЫЕ КОНТЕНТЫ ... 141

9.1. ВЕРСИИ ПРОДУКТОВ ... 141
9.1.1. Источники данных функционала ... 141
9.1.2. Первоначальная синхронизация истории продуктов и референсной витрины ... 141
9.1.3. Фильтрация версий продукта ... 141
9.1.4. Структура данных контентов ... 142
9.1.5. Версии локализованного продукта ... 144
9.2. НЕПРАВИЛЬНО УДАЛЕННЫЕ ПРОДУКТЫ ... 144
9.3. ПРОДУКТЫ ТОЛЬКО НА STAGE ... 144

10. ЛОКАЛИЗАЦИЯ DPC ... 145

10.1. НАСТРОЙКА ЛОКАЛИЗАЦИИ ... 146

11. КОНФИГУРАЦИЯ QP ... 147

12. СТРУКТУРА КОНТЕНТОВ ... 148

12.1. ИНДЕКСЫ ELASTIC ... 148
12.2. КАНАЛЫ ... 148
12.3. ЛИМИТЫ HIGHLOAD API ... 149
12.4. МЕТОДЫ HIGHLOAD API ... 149
12.4.1. Дополнительные методы (Highload API Methods) ... 150
12.5. ОПИСАНИЯ КАРТОЧЕК ПРОДУКТОВ ... 150
12.6. ОПИСАНИЯ ПРОДУКТОВ ... 150
12.7. ПОЛЬЗОВАТЕЛИ HIGHLOAD API ... 150
12.8. СЕРВИСЫ ... 151

12.9. Форматеры 151
12.10. Локализация 152
12.11. Мэппинг локализации 152
12.12. Языки 152

13. СТРУКТУРА ДАННЫХ ТАБЛИЦ 153

13.1. Таблица Messages БД dpc_notifications 153

14. РУКОВОДСТВО ПО УСТАНОВКЕ QP8.PRODUCTCATALOG 154

14.1. Установка на Windows 154
14.1.1. Требования 154
14.1.2. Необходимые права 154
14.1.3. Автоматическая установка 154
14.1.4. Ручная установка 156
14.2. Установка на Linux 157
14.2.1. Требования 157
14.2.2. Установка БД 157
14.2.3. Установка на Linux (с использованием Docker) 158
14.2.4. Установка на Linux (без использования Docker) 159
14.2.5. Установка на Linux (с использованием Kubernetes) 169
14.2.6. Настройка nginx 169
14.2.7. Заполнение индексов ElasticSearch / OpenSearch 170
14.3. Установка демо-сайта на Linux 171
14.3.1. Установка демо-сайта (с использованием Docker) 171
14.3.2. Установка демо-сайта (без использования Docker) 171
14.3.3. Установка демо-сайта (с использованием Kubernetes) 173
14.3.4. Настройка nginx 173

1. 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 `http://quantumart.ru/`.
Код	Пример кода.	`public DataTable Data { get; set; }`
Переменная	Используется для указания переменного значения.	Формат URL: *Базовый URI*/*Псевдоним объекта*
Требует дополнения	TBD (to be determined). Указывает, что необходима доработка текста – проверка корректности утверждения, детализация, правка после внесения изменений в документ и т.п.	Система работает с одной БД.
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.
<значение>	Значение между символом < и > необязательно для указания	Выберите категорию <и подкатегорию>
термин	Термины, на которые необходимо обратить внимание	*Integer* – числовой тип данных

2. Определения и термины, используемые в документе

2.1. Общепринятые термины

В таблице 1 приведены определения, используемые в руководстве.

Таблица 1. Определения, используемые в Руководстве

Определение	Пояснение
Elastic	Поисковая система, поддерживающая полнотекстовый поиск. В текущем документе соответствует как Elasticsearch, так и OpenSearch
БД	База данных
ГПИ	Графический пользовательский интерфейс
Информационная Система (далее **Система**)	Автоматизированный программно-аппаратный комплекс, предназначенный для хранения, обработки и выдачи данных
ОС	Операционная система

2.2. Термины Продуктового каталога

В таблице 2 приведены термины Продуктового каталога.

Таблица 2. Термины Продуктового каталога

Определение	Пояснение
QP8.Framework (далее **QP**)	Программный продукт, предназначенный для разработки программной части Систем. Например, Продуктовый каталог
QP8.ProductCatalog (Digital Product Catalog; Продуктовый каталог) (далее **DPC**)	Система управления структурированными данными, которые хранятся в QP8
Продукт	Это совокупность статей одного или нескольких контентов и их связей вложенности

2.3. Термины QP

В таблице 3 приведены термины QP.

Таблица 3. Термины QP

Определение	Пояснение
Customer code	Код клиента. Уникальный параметр (тенант), определяющий БД, с которой взаимодействует бэкенд QP. Выбирается пользователем при входе в Систему.
Бэкенд	Копия QP. Бэкенд обладает ГПИ для работы с содержимым БД Системы
Витрина	Удалённая Система, обладающая возможностью автоматически получать уведомления об изменении данных о продуктах от Системы DPC
Контент	Раздел сайта
Мультитенантность	Архитектура программного обеспечения, в которой один экземпляр приложения обслуживает много клиентов (тенантов).

Поле	Атрибут контента. С использованием полей формируется структура данных для контента
Пользовательское действие	Дополнительная функциональная возможность для бэкенда, добавленная Разработчиком в Систему
Сайт	Набор данных в бэкенде. Допускается создание нескольких сайтов. Содержимое каждого сайта определяется созданными в нём контентами
Статья	Элемент контента. Статья содержит данные, заданные в поля контента

2.4. Роли пользователей Продуктового каталога

В таблице 4 приведены определения ролей пользователей в Продуктовом каталоге.

Таблица 4. Определение ролей пользователей в Продуктов каталоге

Роль	Определение
Администратор	Пользователь с правами на внесение любых изменений в Систему, которые можно выполнить с использованием бэкенда QP либо отдельной административной панели управления
Контент-менеджер	Пользователь с ограниченными правами на изменение содержимого Системы с использованием бэкенда QP либо отдельной административной панели управления
Пользователь	Персона, осуществляющая взаимодействие с Системой посредством интерфейсов, предоставляемых Системой
Разработчик	Пользователь с правами на внесение любых изменений в Систему (в том числе в содержимое скриптов, структуру БД)

3. Введение

Продуктовый каталог (DPC) является решением для управления данными сложной структуры, хранящихся в QP8. В терминах каталога, Сайт также является внешней системой по отношению к нему.

Предполагается, что продукт состоит из совокупности статей одного или нескольких контентов произвольной вложенности. Система позволяет для каждого типа продукта указать, из каких контентов и их связей он состоит. И для каждой из связей настроить поведение для загрузки, клонирования и удаления.

Таким образом, система управления структурой продуктов не зависит от конкретной структуры контентов и полей. DPC может быть настроен на уже существующих контентах.

DPC располагает встроенным редактором продуктов, который гибко настраивается для каждого продукта с помощью Xml.

Продуктовый каталог отправляет изменившиеся продукты по принципу PUSH-уведомлений всем подписавшимся на данный тип продукта витринам. Витрины могут подписываться как на опубликованные продукты, так и на версии, предназначенные для stage-сайтов. Кроме того, в коробку входит высокопроизводительный REST WEB API, позволяющий:

- Получать продукты по различным условиям фильтрации
- Получать продукты с использованием полнотекстового поиска
- Получить только часть полей продуктов, указав их в запросе
- Настраивать права и квоты для использующих его приложений

Также в состав DPC входит WEB API, предназначенный для управления продуктами. С помощью него можно:

- Выполнять CRUD-операции над продуктами
- Выполнять различные действия (публикация, постобработка продуктов)
- Получать неопубликованные версии продуктов в различных форматах (xml, json)

Продуктовый каталог поддерживает фоновые задачи и обладает интерфейсом, рассчитанным на управление таким продуктами повышенной сложности, как Тарифы, Услуги и подобные им. Для интеграции с внешними системами (например, обновление цен в каталоге, полученных из биллинга) при необходимости будут доработаны коннекторы к внешним системам.

Кроме того, в состав продуктового каталога входит генератор PDF, с помощью которого можно отдавать внешним витринам или пользователям сгенерированный прайс-листы или любые другие документы, в которые входят данные продуктов. Шаблон для документа можно указывать как формате MS Word, так и с применением шаблонизатора Razor (Html+CSharp).

4. Вход в DPC

4.1. Средствами веб-браузера

Для входа в DPC необходимо:

1. Ввести в адресную строку браузера ссылку на веб-ресурс Системы. Откроется экранная форма авторизации пользователя в Системе (рис. 4.1);

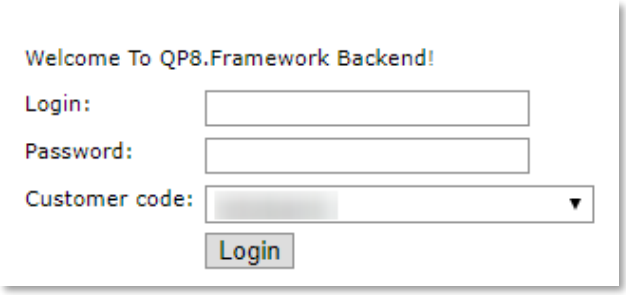


Рисунок 4.1. Вход в DPC

2. Заполнить поля:
   2.1. Login;
   2.2. Password;
   2.3. Customer Code.
3. Нажать кнопку «Login».

После прохождения идентификации и аутентификации пользователь допускается к работе в Системе.

4.2. Вход в CMS средствами программы входа в операционную систему Windows WinLogon

Вход через *WinLogon* – это вход в учетную запись Windows, на которой настроен доступ к бэкенду через программу входа Windows WinLogon.

Для входа через WinLogon необходимо выполнить следующие действия:

1. Ввести учетные данные пользователя для входа в Windows;
2. Запустить браузер;
3. В адресной строке браузера ввести ссылку на веб-ресурс Системы;
4. Выбрать Customer code (рис. 4.2 п. 1);
5. Кликнуть по кнопке Login (рис. 4.2 п. 2).

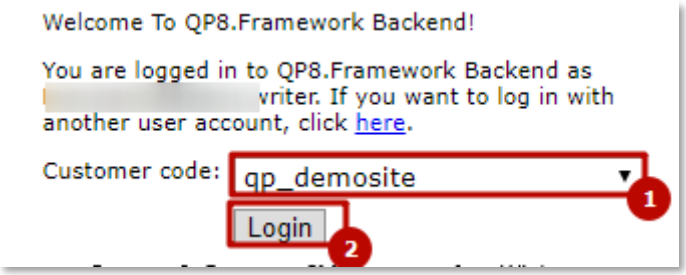


Рисунок 4.2. Вход в DPC средствами WinLogon
(п.1 – выбор Customer Code, п.2 – клик по кнопке)

После прохождения идентификации и аутентификации пользователь допускается к работе в Системе.

5. Выход из DPC

Для завершения работы с Системой необходимо нажать кнопку **«Выход» (Exit)**, расположенную в правом верхнем углу каждой экранной формы (рис. 5.1).

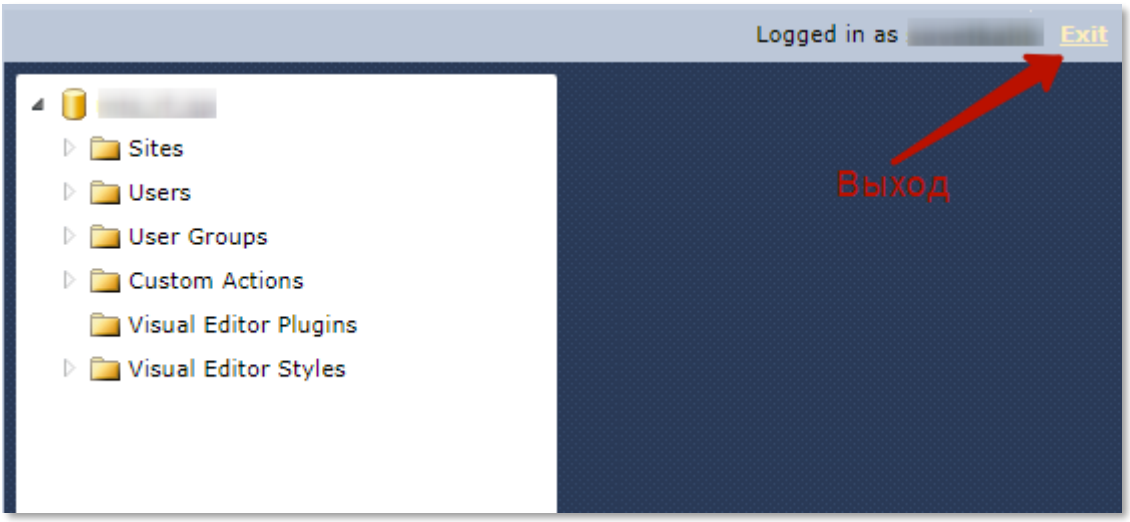


Рисунок 5.1. Выход из DPC

Система отобразит страницу авторизации пользователя.

6. Архитектурная схема DPC

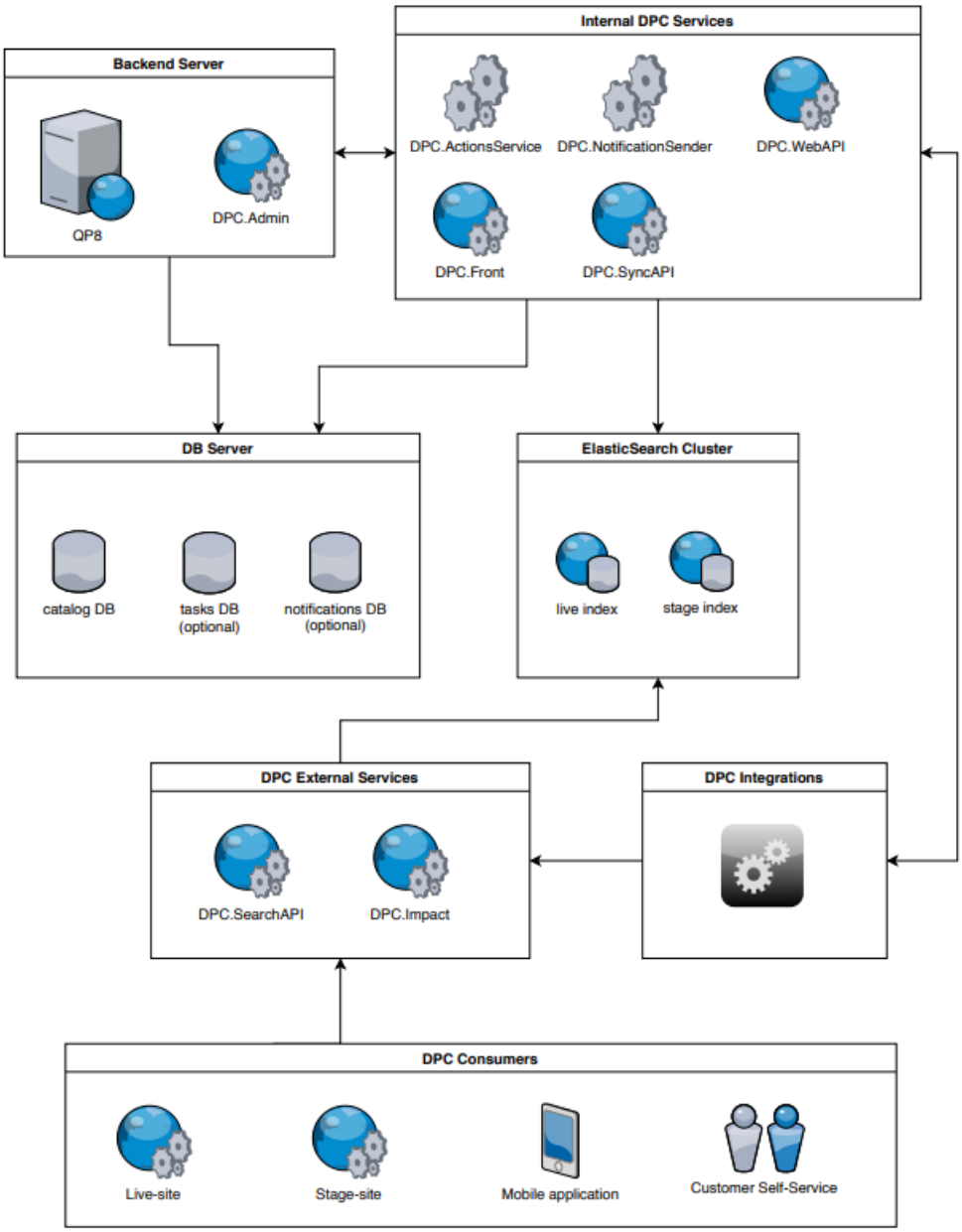


Рисунок 6.1. Архитектурная схема DPC

6.1. Backend Server

- QP8 – базовая конфигурация QP.
- DPC.Admin – административный модуль DPC, который разворачивается на одном сервере с QP8 и подключается в виде набора пользовательских действий. Предоставляет UI для взаимодействия с внутренними компонентами DPC.

6.2. Internal DPC Services

Набор внутренних сервисов DPC. Обычно разворачиваются на отдельном сервере, выделенном для сервисов, но могут быть развернуты и на бэкенд-сервере. В целом они обеспечивают работу с продуктами: сборку, публикацию, изменение, доставку потребителям (как внутренним, так и внешним):

- DPC.ActionsService – win-сервис выполнения задач DPC;
- DPC.NotificationSender – win-сервис отправки продуктов на витрины (как внутренние, так и внешние). Используется для интеграций (получение измененных продуктов);
- DPC.WebAPI (DPC.Api) – веб-приложение для CRUD-операций с продуктами. Используется для интеграций (модификация существующих продуктов по информации из внешнего источника);
- DPC.Front – веб-приложение, обеспечивающее сохранение продуктов во внутреннюю (референсную) витрину. Может использоваться для интеграций (источник данных для первичного заполнения);
- DPC.SyncAPI (DPC.Sync) – веб-приложение, обеспечивающее сохранение продуктов в Elastic.

6.3. DB server

Хранит БД для DPC. Один экземпляр DPC может работать как со многими базами каталогов, так и с одной в зависимости от конфигурации. Если DPC работает с одной БД каталога, то дополнительно могут быть вынесены в отдельные БД данные для задач DPC и нотификаций DPC.

6.4. Elasticsearch Cluster

Хранит информацию по опубликованным продуктам. Минимальный набор индексов для каждого каталога – 2: для хранения live- и stage-продуктов.

6.5. External DPC Services

Внешние сервисы DPC. Обычно разворачиваются в отдельном отказоустойчивом кластере. Читают информацию из Elastic и предоставляют ее потребителям.

- DPC.SearchAPI (DPC.Search) – веб-приложение, реализующее API для получения информации по продуктам из Elastic;
- DPC.Impact – веб-приложение, реализующее API для получения информации о влиянии продуктов друг на друга (калькуляторы). Данный сервис не входит в основной дистрибутив каталога и устанавливается дополнительно.

6.6. DPC Consumers

Основные потребители информации DPC:

- live-сайт,
- stage-сайт,
- мобильное приложение,
- личный кабинет.

6.7. DPC Integrations

Возможные внешние интеграции DPC. Они могут взаимодействовать с внутренними компонентами:

- получать информацию об изменениях продуктов,
- осуществлять первичную заливку данных о продуктах,
- выполнять CRUD-операции с продуктами.

Также они могут взаимодействовать с внешними компонентами DPC, читая информацию о опубликованных продуктах.

7. Основные компоненты DPC

DPC состоит из следующих компонентов:

- DPC Admin;
- Сервис выполнения отложенных задач ActionService;
- Сервис отправки продуктов на витрины NotificationSender;
- Web API;
- Elastic API (Sync API, Search API);
- Генератор PDF;
- Высокопроизводительная витрина.

7.1. Административный модуль (DPC.Admin)

DPC.Admin – веб-приложение с пользовательскими действиями и API для валидации.

Пользовательские действия – это веб-приложения или методы для работы с данными DPC, которые находятся по указанному URL.

Пользовательские действия делятся на 2 вида:

- **Интерфейсные** – веб-приложения, во время их выполнения ГПИ не доступен. Характерной чертой таких действий является отображение в ГПИ CMS всплывающих окон или новых вкладок;
- **Неинтерфейсные** – методы или веб-приложения, если существует предварительный запрос, вызов которых не блокирует работу с CMS, т.е. пользователь может вызвать действие и выполнять иные операции.

Пользовательские действия находятся в группе «Пользовательские действия» (Custom Actions).

7.1.1. Продукты

Описание продуктов

Описание продуктов – схема продукта, которая описывает как собирать продукт и какие данные в него входят.

Контент (рис. 7.1 п.2), который содержит описание продуктов находится в группе «Служебные» (рис. 7.1 п.1).

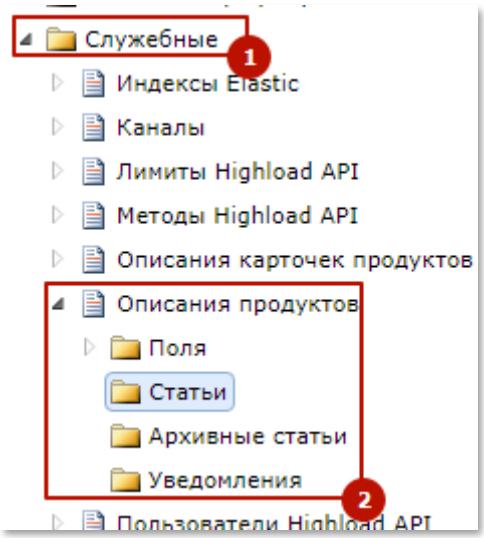


Рисунок 7.1. Описание продукта
(п.1 – Группа контентов, п.2 – контент «Описания продуктов»)

Статьи контента содержат описание продукта, которое находится в поле `XmlDefinition` в виде XML-разметки (рис.7.2).

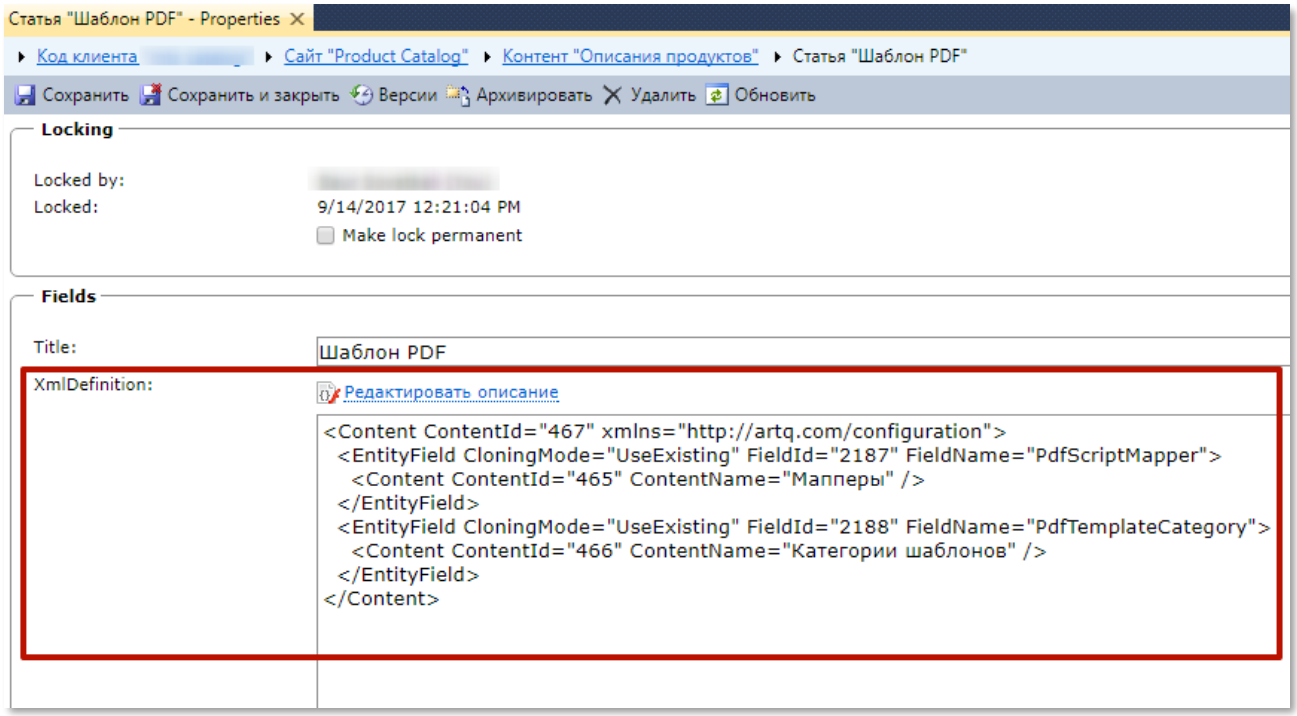


Рисунок 7.2. Описание продукта в поле XmlDefinition

Описание содержит информацию, из каких контентов и связей состоит продукт, XML-разметка полностью соответствует структуре продукта.

Корневым элементом XML-разметки является тег `Content`, задающий контент, статьи которого являются продуктами. Дочерние элементы – это поля контента. Поля, являющиеся связями, могут содержать тег `Content`.

Описание допускает неограниченную вложенность и содержание циклов. В одной БД допускается неограниченное количество описаний.

Система управления продуктами поддерживает операции каскадного удаления, архивации, клонирования продукта и его связей. Выполнение этих действий могут обрабатываться дочерние статьи. Например, при клонировании продукта его параметры также следует клонировать, но статьи, являющиеся словарями (регионы, группы, и т. д.) клонировать не следует. Настройки поведения при выполнении подобных операций указываются в описании продукта (см. Редактировать описание).

Стандартная структура продукта представлена в листинге 1.

Листинг 1. Стандартная структура продукта

```
- Content
    - PlainField
    - EntityField
        - Content
            - …
    - BackwardRelationField
        - Content
            - …
    - ExtensionField
        - Content
        - Content
        - …
```

Фрагмент описания продукта в формате XML-разметки приведен в листинге 2.

Листинг 2. Фрагмент описания продукта в формате XML-разметки

```
<Content ContentId="340" ContentName="Группы продуктов">
      <PlainField ShowInList="True" FieldName="Title" FieldId="1329"/>
      <EntityField CloningMode="UseExisting"  FieldName="Texts" FieldId="1334" >
            <Content ContentName="Дополнительные тексты групп продуктов"
ContentId="341" />
      </EntityField>
      <PlainField ShowInList="True" FieldName="Parent" FieldId="1337"/>
      <EntityField CloningMode="UseExisting"  FieldName="Modifiers" FieldId="1353" >
            <Content ContentName="Модификаторы групп продуктов"  ContentId="371" />
      </EntityField>
</Content>
```

Поля `FieldName` и `ContentName` необязательные, носят информативный характер.

Описание атрибутов XML-тегов приведено в разделе Редактор описаний.

Дочерние элементы тега `Content` указывают, какие из полей являются частью продукта. Можно включить или исключить из описания скалярные поля (текстовые, числовые, и т. д.). Для полей связей можно указать, загружать ли связанные статьи, а также определить политику клонирования и удаления (см. Редактор описаний).

Редактор описаний

Пользовательское действие выполняется вызовом метода `DefinitionEditor`.

Пользовательское действие вызывает редактор описания продукта. Редактор Definitions находится в контенте `Описания продуктов`.

Для открытия редактора необходимо перейти в форму редактирования статьи контента (форма редактирования описана в приложении А) и кликнуть по псевдоссылке «Редактировать описание».

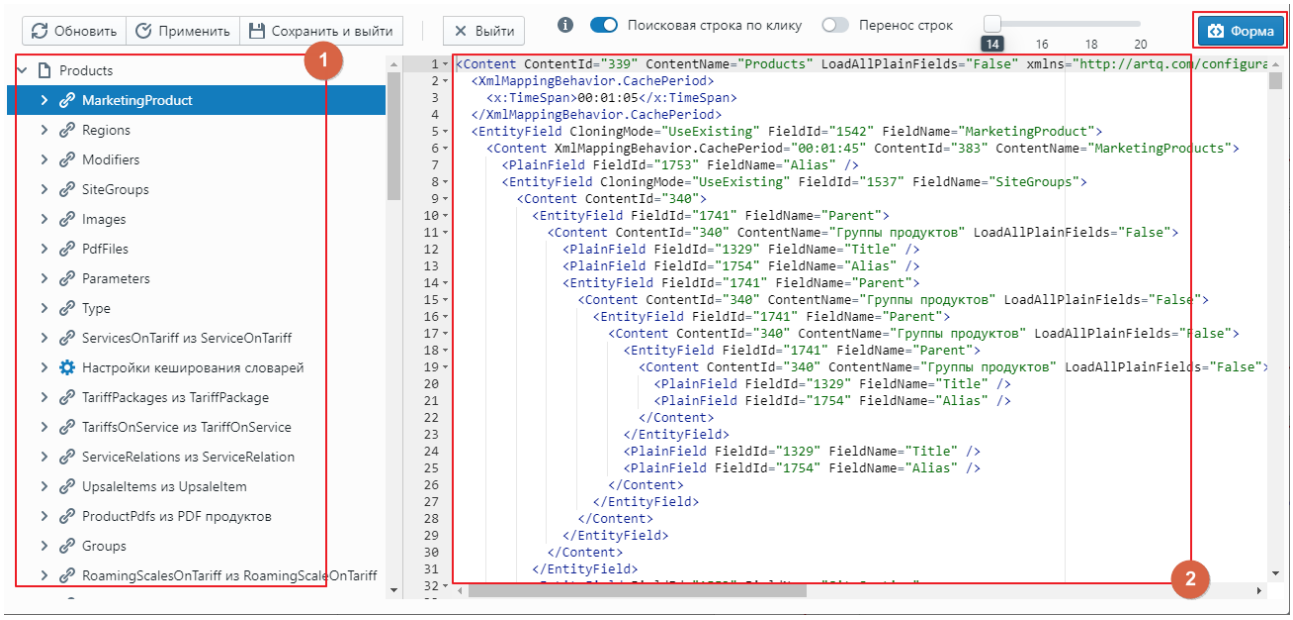
Внимание! Клик по псевдоссылке «Редактировать описание» открывает в текущей вкладке веб-браузера визуальный редактор. Для возврата к форме редактирования статьи необходимо нажать по пиктограмме «X» в правом верхнем углу окна.

Внимание! На текущий момент существует ограничение, при первом входе в редактор описаний открывается пустое окно. Для решения проблемы необходимо авторизоваться в любом пользовательском действии, например, открыть «Каналы». Произойдет авторизация и после этого повторить открытие редактора описаний.

Визуальный редактор разделен на 3 области:

1. Дерево описания продукта (рис. 7.3 п. 1);
2. Свойство узла продукта (рис.7.3 п. 2);
3. Редактор XML-разметки (рис. 7.3 п. 3).

При выделении элемента в дереве становится активной кнопка «Форма», при нажатии на которую происходит переключение с редактора кода на форму свойств узла, соответственно повторное нажатие возвращает окно редактора.



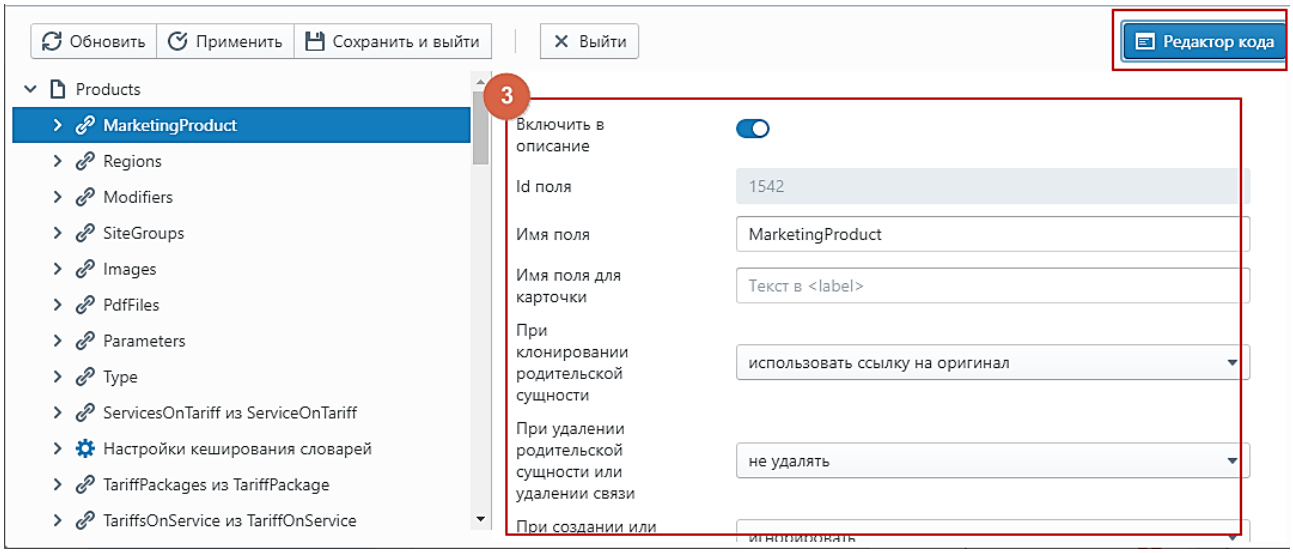


Рисунок 7.3. Визуальный редактор описания продукта
(п.1 – Дерево свойств продукта, п.2 – XML-редактор продукта, п.3 – Свойства элемента дерева)

Описание продукта состоит из XML-разметки. Редактор позволяет создать или редактировать XML-разметку описания напрямую в окне XML-редактора, либо внести правки при помощи формы.

Для задания свойств элементам дерева необходимо выбрать его в дереве описания продукта (рис. 7.3 п.1) и указать свойства в соответствующем поле (рис. 7.3 п.3).

Клик по элементу дерева выводит свойства узла, который можно редактировать, если у пользователя есть права на редактирование. После редактирования свойств элемента дерева необходимо кликнуть по кнопке применить, чтобы сохранить изменения (рис. 7.3). Подробнее про узлы см. разделы ниже.

Также доступен поиск в редакторе кода с помощью регулярных выражений (рис. 7.4).

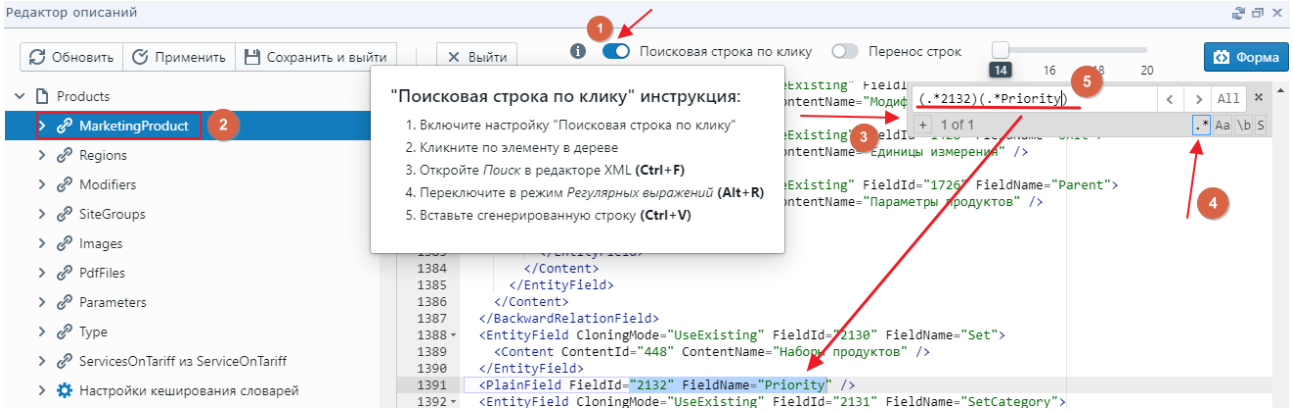


Рисунок 7.4. Алгоритм поиска строки по клику

Внимание! Для сохранения внесенных изменений в редакторе описаний необходимо нажать на кнопку «Сохранить» на панели XML-редактора, после этого необходимо закрыть окно редактора и сохранить изменения в статье продукта.

Каждому контролу формы соответствует элемент или атрибут XML. Пример представлен на рис. ниже.5.

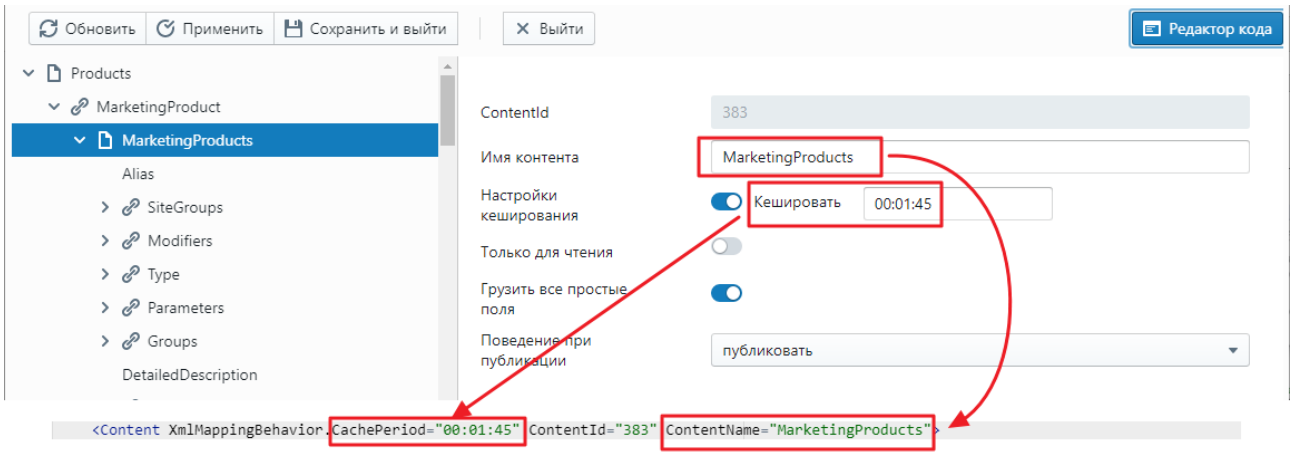


Рисунок 7.5. Отображение изменений свойств элемента в XML-структуре.

Примечание: поле ввода XML-разметки соответствует описано в подразделе «Текстовое поле» раздела «Типы полей» приложения А.

Доступно изменение размера областей визуального редактора.

Для обновления содержимого окна необходимо кликнуть по пиктограмме ( ). Для того, чтобы уменьшить/развернуть окно необходимо кликнуть по пиктограмме (/).

Content

Для узла дерева уровня Content возможно задать поля, представленные на рис. 7.6 и в таблице 5.

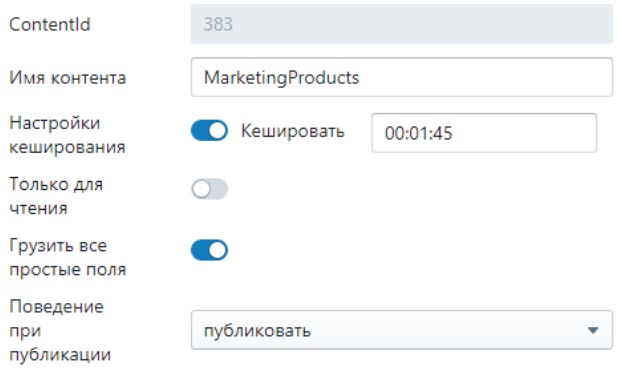


Рисунок 7.6. Форма для элемента Content

Таблица 5.Соответствие формы и XML для элемента Content

Content Корневой элемент		
Свойство в редакторе описаний	**Элементы/ Атрибуты**	**Описание**
Id поля (ContentId)	ContentId	Идентификатор контента. **Важно!** Id поля не редактируется через форму Возможное значение: числовое Пример XML-разметки: `ContentId="524"`
Настройки кеширования (Cache Settings)	CachePeriod	Интервал кеширования Возможное значение: строковое представление TimeSpan Пример XML-разметки: `CachePeriod="01:45:00"`
Имя контента (Content Name)	ContentName	Необязательное поле. Содержит имя контента Возможное значение: строковое Пример XML-разметки: `ContentName="Вопросы и ответы - Типы"`
Только для чтения (Read-only)	IsReadOnly	Режим чтения Возможные значения: Бинарное - true. В XML: `IsReadOnly="True"` - false. В XML: `IsReadOnly="False"`(по умолчанию)
Грузить все простые поля (Load all plain fields)	LoadAllPlainFields	Признак того, загружать все скалярные поля или нет Возможные значения: Бинарное - true. В XML: `LoadAllPlainFields="True"` (по умолчанию) - false. В XML: `LoadAllPlainFields="False"`
Поведение при публикации (Publish Behaviour)	PublishingMode	Поведение при публикации продукта Возможные значения:

		- Публиковать (по умолчанию) (Publish). В XML: `PublishingMode="Publish"` - Не публиковать (Don't publish). В XML: `PublishingMode="SkipRecursive"`

EntityField

Для узла дерева, являющегося связью, возможно задать поля, представленные на рис. 7.7 и в таблице 6.

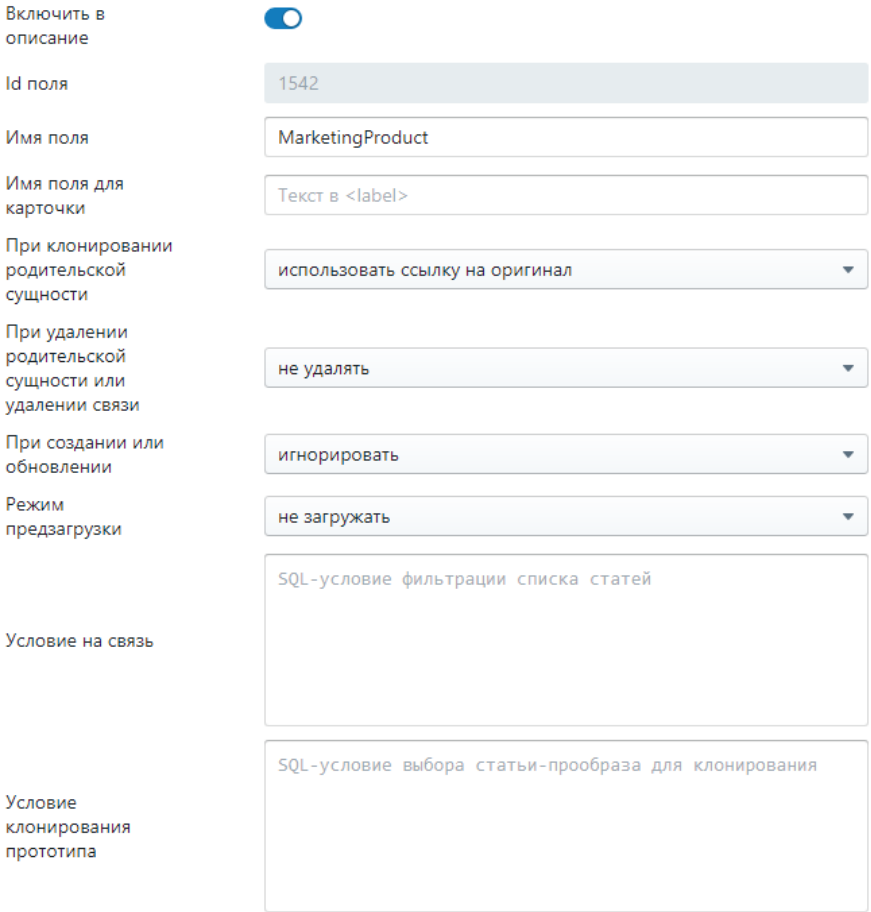


Рисунок 7.7. Форма для элемента EntityField

Таблица 6. Соответствие формы и XML для элемента EntityField

EntityField Связь с контентами		
Свойство в редакторе описаний	**Элементы/ Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта Пример XML-разметки: `<EntityField CloningMode="UseExisting" FieldId="2658" FieldName="Parent">`
Id поля (Field Id)	FieldId	Идентификатор поля **Важно!** Id поля не редактируется через форму Возможное значение: числовое. Пример XML-разметки: `FieldId="2675"`
Имя поля (Field Name)	FieldName	Имя поля Возможное значение: строковое Пример XML-разметки: `FieldName="QuestionTypes"`
Имя поля для карточки (Field Name for Card)	FieldTitle	Имя поля для карточки Возможное значение: строковое Пример XML-разметки: `FieldTitle="Main"`
При клонировании родительской сущности (Cloning Mode)	CloningMode	Действие над полем при клонировании продукта Возможные значения: - Использовать ссылку на оригинал (Use existing reference). В XML: `CloningMode="UseExisting"`; - Устанавливать пустую ссылку (по умолчанию) (Set null reference). В XML: `CloningMode="SetNull"`; - Копировать сущность (Clone entity). В XML: `CloningMode="Copy"`;
При удалении родительской сущности или удалении связи (Deleting Mode)	DeletingMode	Действие над полем при удалении продукта Возможные значения: - Удалять (Remove). В XML: `DeletingMode="Delete"` - Не удалять (по умолчанию) (Don’t remove). В XML: `DeletingMode="Keep"`
При создании или обновлении (Updating Mode)	UpdatingMode	Действие над полем при обновлении продукта Возможные значения: - Обновлять/создавать (Update or create). В XML: `UpdatingMode="Update"` - Игнорировать (по умолчанию) (Ignore). В XML: `UpdatingMode="Ignore"`
Режим предзагрузки (Preloading Mode)	PreloadingMode	Действие над полем при загрузке Возможные значения: - Не загружать (по умолчанию) (Don’t load). В XML: `PreloadingMode="None"` - Загружать сразу (Eager loading). В XML: `PreloadingMode="Eager"` - Загружать отложенно (Lazy loading). В XML: `PreloadingMode="Lazy"`
Условие на связь (Relation Condition)	RelationCondition	Условие на связь Возможное значение: строковое Пример XML-разметки: `RelationCondition="c.ProductType = 427"`

Условие клонирования прототипа (Clone Prototype Condition)	ClonePrototypeCondition	Условие клонирования прототипа Возможное значение: строковое Пример XML-разметки: `ClonePrototypeCondition="c.ProductType = 427 "`
Ссылается на контент (Relates to content)	Атрибуты ContentId и ContentName из дочернего элемента Content	Указывает на связь с полем уровня Контент, доступно в режиме read-only Пример XML-разметки: `<Content ContentId="383" ContentName="MarketingProducts">`

BackwardRelationField

Для узла, являющегося обратной связью с контентами, можно задать поля, представленные на рис. 7.*8. Форма для элемента BackwardRelationField* и в таблице 7.

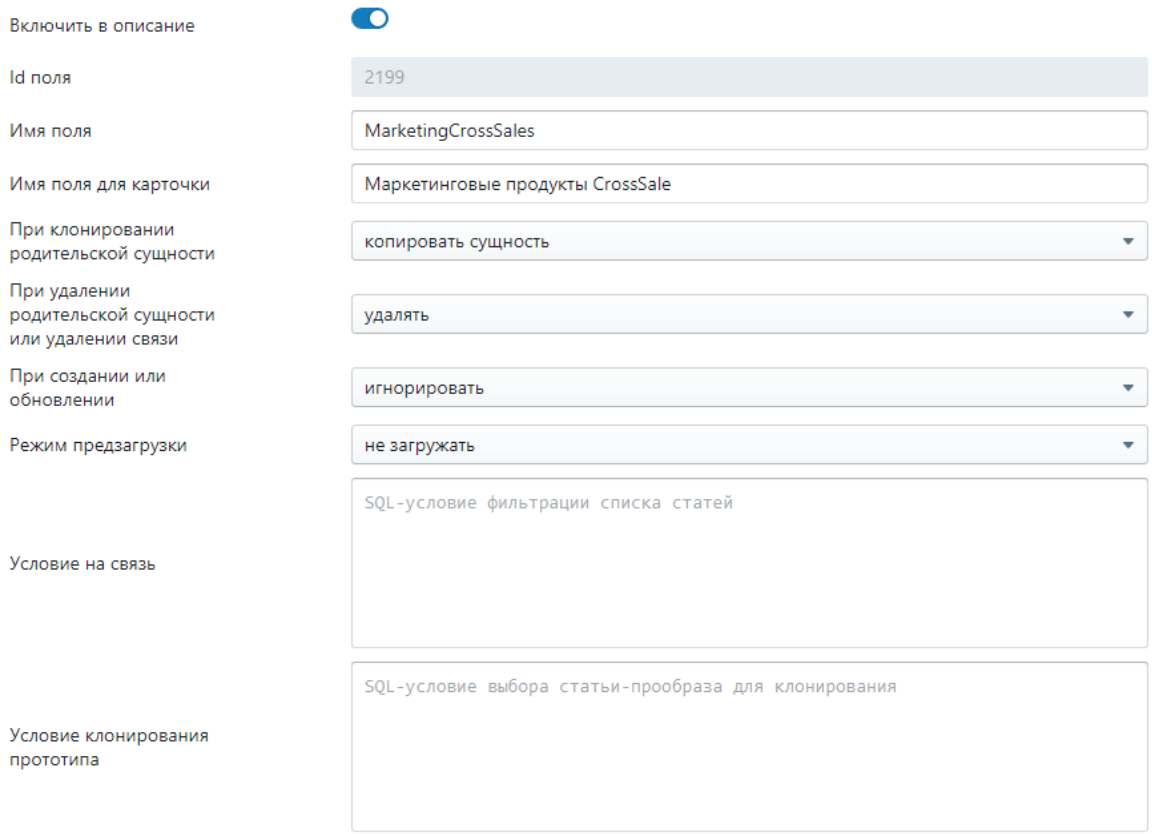


Рисунок 7.8. Форма для элемента BackwardRelationField

Таблица 7. Соответствие формы и XML для элемента BackwardRelationField

BackwardRelationField Обратная связь с контентами		
Свойство в редакторе описаний	**Элементы/ Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта Пример XML-разметки: `<BackwardRelationField CloningMode="Copy" DeletingMode="Delete" DisplayName="Маркетинговые продукты`

		`CrossSale" FieldId="2199" FieldName="MarketingCrossSales" FieldTitle="Маркетинговые продукты CrossSale" RelationGroupName="MarketingProduct">`
Id поля (Field Id)	FieldId	Идентификатор поля **Важно!** Id поля не редактируется через форму Возможное значение: числовое Пример XML-разметки: `FieldId="2199"`
Имя поля (Field Name)	FieldName	Имя поля Возможное значение: строковое Пример XML-разметки: `FieldName="MarketingCrossSales"`
Имя поля для карточки (Field Name for Card)	FieldTitle	Задает имя поля для карточки Возможное значение: строковое Пример XML-разметки: `FieldTitle="Маркетинговые продукты CrossSale"`
	DisplayName	Возможное значение: строковое Пример XML-разметки: `DisplayName="Маркетинговые продукты CrossSale"`
При клонировании родительской сущности (Cloning Mode)	CloningMode	Действие над полем при клонировании продукта Возможные значения: - Использовать ссылку на оригинал (Use existing reference). В XML: `CloningMode="UseExisting"`; - Устанавливать пустую ссылку (по умолчанию) (Set null reference). В XML: `CloningMode="SetNull"`; - Копировать сущность (Clone entity). В XML: `CloningMode="Copy"`;
При удалении родительской сущности или удалении связи (Deleting Mode)	DeletingMode	Действие над полем при удалении продукта Возможные значения: - Удалять (Remove). В XML: `DeletingMode="Delete"` - Не удалять (по умолчанию) (Don’t remove). В XML: `DeletingMode="Keep"`
При создании или обновлении (Updating Mode)	UpdatingMode	Действие над полем при обновлении продукта Возможные значения: - Обновлять/создавать (Update or create). В XML: `UpdatingMode="Update"` - Игнорировать (по умолчанию) (Ignore). В XML: `UpdatingMode="Ignore"`
	RelationGroupName	Имя группы связей Возможное значение: строковое Пример XML-разметки: `RelationGroupName="MarketingProduct"`
Ссылается на контент (Relates to content)	Атрибуты ContentId и ContentName из дочернего элемента Content	Указывает на связь с полем уровня Контент, доступно в режиме read-only Пример XML-разметки: `<Content ContentId="404" ContentName="ServiceOnTariff">`

ExtensionField

Для уровня, который указывает расширение, можно задать поля, представленные на рис. 7.9 и в таблице 8.

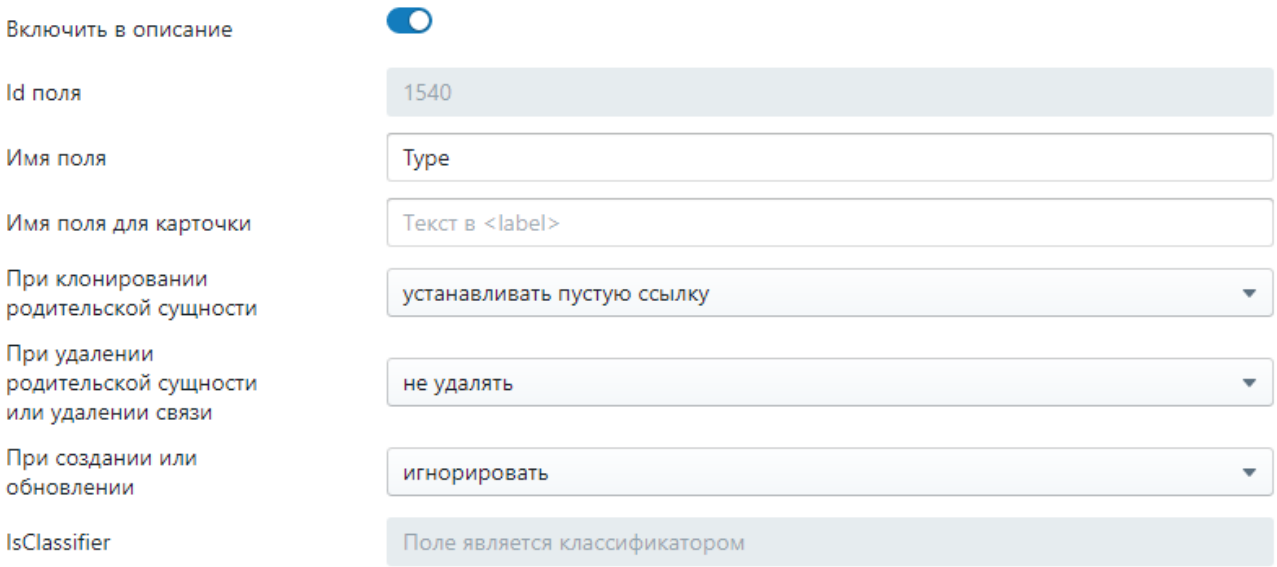


Рисунок 7.9. Форма для элемента ExtensionField

Таблица 8. Соответствие формы и XML для элемента ExtensionField

ExtensionField Поле для указания расширения		
Свойство в редакторе описаний	**Элементы/ Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта Пример XML-разметки: `<ExtensionField CloningMode="UseExisting" FieldId="1417" FieldName="Type">`
Id поля (Field Id)	FieldId	Идентификатор поля **Важно!** Id поля не редактируется через форму Возможное значение: числовое Пример XML-разметки: `FieldId="1540"`
Имя поля (Field Name)	FieldName	Имя поля Возможное значение: строковое Пример XML-разметки: `FieldName="Type"`
При клонировании родительской сущности (Cloning Mode)	CloningMode	Действие над полем при клонировании продукта Возможные значения: - Использовать ссылку на оригинал (Use existing reference). В XML: `CloningMode="UseExisting"`; - Устанавливать пустую ссылку (по умолчанию) (Set null reference). В XML: `CloningMode="SetNull"`; - Копировать сущность (Clone entity). В XML: `CloningMode="Copy"`;
При удалении родительской	DeletingMode	Действие над полем при удалении продукта Возможные значения:

сущности или удалении связи (Deleting Mode)		- Удалять (Remove). В XML: `DeletingMode="Delete"` - Не удалять (по умолчанию) (Don't remove). В XML: `DeletingMode="Keep"`
При создании или обновлении (Updating Mode)	UpdatingMode	Действие над полем при обновлении продукта Возможные значения: - Обновлять/создавать (Update or create). В XML: `UpdatingMode="Update"` - Игнорировать (по умолчанию) (Ignore). В XML: `UpdatingMode="Ignore"`

PlainField

Для уровня простых полей можно задать поля, представленные на рис. 7.10 и в табл. 9.

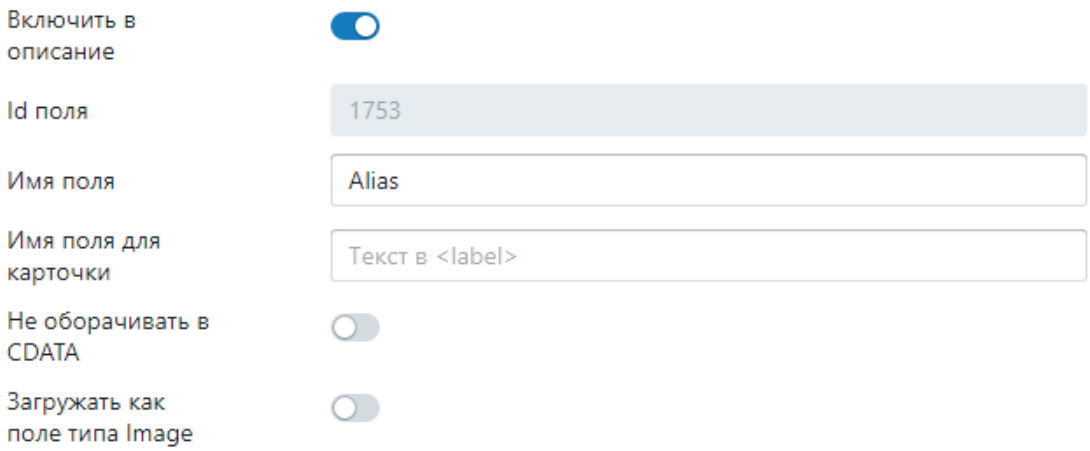


Рисунок 7.10. Форма для элемента PlainField

Таблица 9. Соответствие формы и XML для элемента PlainField

PlainField Обычное поле. Используется, например, для указания приоритета сортировки		
Свойство в редакторе описаний	**Элементы/Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта Пример XML-разметки: `<PlainField FieldId="2644" FieldName="Alias" />`
Id поля (Field Id)	FieldId	Идентификатор поля **Важно!** Id поля не редактируется через форму Возможное значение: числовое Пример XML-разметки: `FieldId="2640"`
Имя поля (Field Name)	FieldName	Название поля Возможное значение: строковое Пример XML-разметки: `FieldName="Alias"`
Имя поля для карточки (Field Name for Card)	FieldTitle	Имя поля для карточки Возможное значение: строковое Пример XML-разметки: `FieldTitle="Main"`
Не оборачивать в CDATA (Don't wrap in CDATA)	CustomProperties с ключом RenderTextFieldAsXml	По умолчанию поля типа Визуальный редактор (Visual Editor) и Текстовое окно (Textbox) оборачиваются в CDATA. При включении данной

		опции содержимое будет вставлено как есть. **Внимание!** Может быть получен невалидный XML
		Пример XML-разметки:
		`<PlainField.CustomProperties>`
		`<x:Boolean x:Key="RenderTextFieldAsXml">True</x:Boolean>`
		`</PlainField.CustomProperties>`

Загружать как поле типа Image (Load as Image)	CustomProperties с ключом RenderFileFieldAsImage	По умолчанию поле типа Файл (File) рендерится вместе с размером файла. Для того, чтобы рендерить его как поле типа Изображение (Image), содержащее только ссылку, можно использовать данную опцию
		Пример XML-разметки:
		`<PlainField.CustomProperties>`
		`<x:Boolean x:Key="RenderFileFieldAsImage">True</x:Boolean>`
		`</PlainField.CustomProperties>`

VirtualField

Для поля из виртуального контента задаются свойства, представленные ниже (рис. 7.*11* и табл. 10).

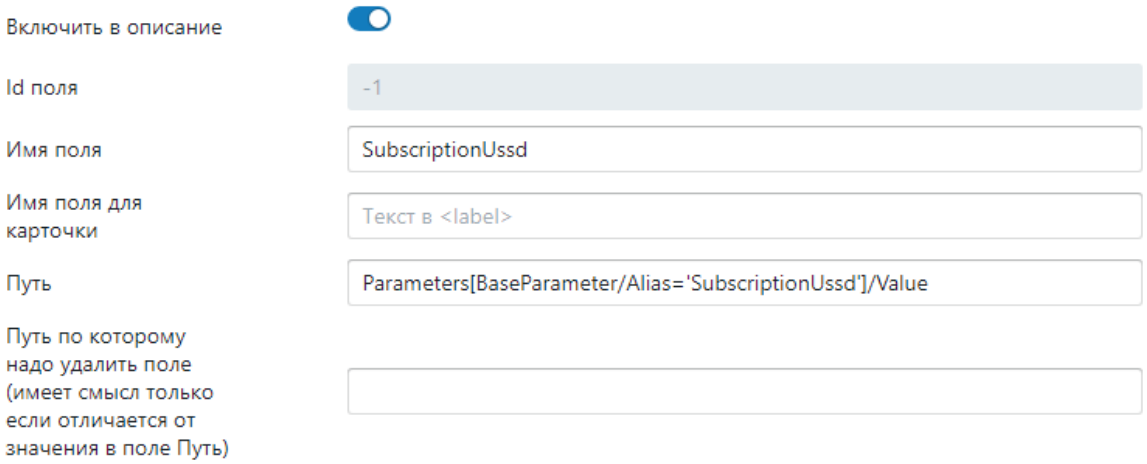


Рисунок 7.11. Форма для элемента VirtualField

Таблица 10. Соответствие формы и XML для элемента VirtualField

VirtualField Поле, значение которого загружается в текущую структуру из виртуального контента		
Свойство в редакторе описаний	**Элементы/Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта Пример XML-разметки: `<VirtualField FieldName="SubscriptionUssd" Path="Parameters[BaseParameter/Alias='SubscriptionUssd']/Value" />`
Id поля (Id Field)	IdField	По умолчанию значение «-1». **Важно!** Id поля не редактируется через форму Возможное значение в XML: числовое

Имя поля (Field Name)	FieldName	Название поля Возможное значение: строковое Пример XML-разметки: `FieldName="SubscriptionUssd"`
Имя поля для карточки (Field Name for Card)	FieldTitle	Имя поля для карточки Возможное значение: строковое Пример XML-разметки: `FieldTitle="Main"`
	ObjectToRemovePath	Необязательное свойство. Задает путь к узлу в структуре продукта, который надо удалить после создания виртуального поля Возможное значение: строковое Пример XML-разметки: `ObjectToRemovePath="Parameters"`
Путь (Path)	Path	Путь, откуда загружается поле. Данные по этому пути автоматически копируются в текущее поле Возможное значение: строковое XPath-выражение. Пример: `"Parameters[BaseParameter/Alias='WebSiteLink']/Value"`, из поля `Parameters` будут выбраны статьи, у которых в поле `Alias` поля `BaseParameter` имеет значение `WebSiteLink`, из отфильтрованных таким образом статей вернется поле `Value`

Dictionaries

Для узла настройки кеширования можно задать следущие поля (рис. 7.12 и табл. 11).

Рисунок 7.12. Форма для элемента Dictionaries

Таблица 11. Соответствие формы и XML для элемента Dictionaries

Dictionaries		
Свойство в редакторе описаний	**Элементы/Атрибуты**	**Описание**
Включить в описание (Include in definition)	Include in definition	Установка чекбокса добавляет поле в описание продукта
Период кеширования по умолчанию (Default cache period)	DefaultCachePeriod	Период кеширования по умолчанию Возможно значение: строковое представление TimeSpan Пример XML-разметки: `DefaultCachePeriod="00:10:00"`

Карточка продукта

Статьи контента «Описания карточек продуктов» позволяют управлять выводом (рендерингом) данных о продукте при вызове пользовательского действия «Продукты». Соответствующее веб-приложение принимает на вход XML-разметку карточки продукта и выводит элементы управления и назначает им функциональностью.

Описание структуры данных контента «Описания карточек продуктов» приведено в подразделе Описания карточек продуктов.

7.1.2. Пользовательские действия

Создание и настройка пользовательского действия

Пользовательское действие разрабатывается разработчиками в виде веб-приложения. Для того, чтобы действие было доступно пользователям DPC его необходимо создать и разместить на сервере.

Пользовательские действия создаются в группе «Пользовательские действия» (Custom Action) и хранятся в БД QP (рис. 7.13).

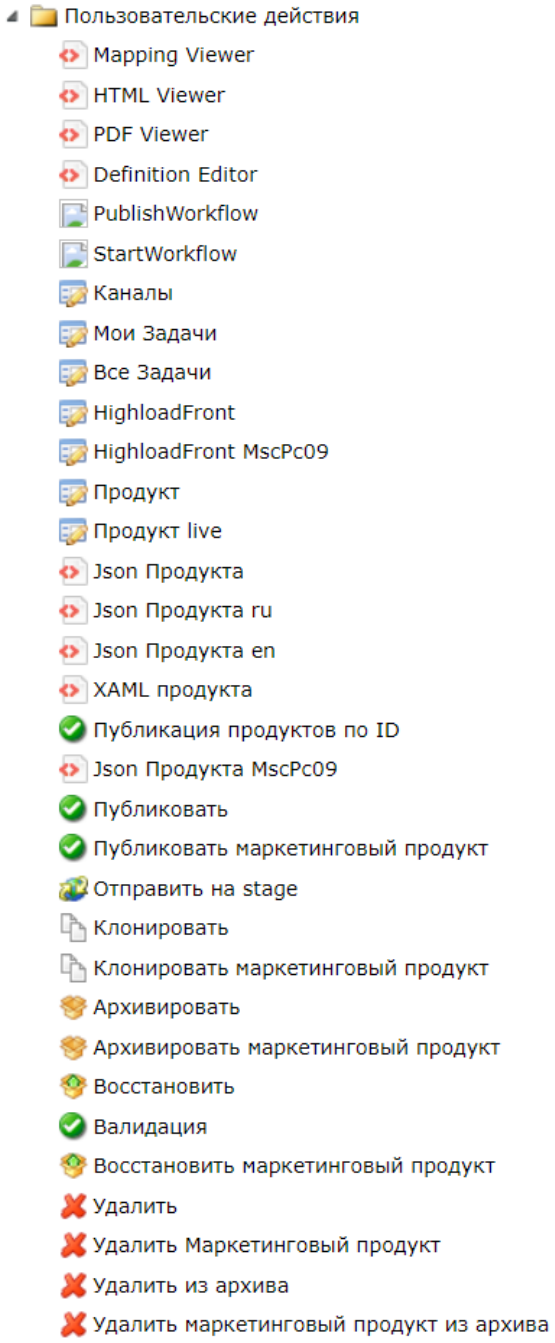


Рисунок 7.13. Группа «Customer Actions»

Для создания действия необходимо кликнуть правой кнопкой мыши по группе и пункт «Новое пользовательское действие». В появившейся вкладке необходимо указать следующую информацию:

- В поле `Name` имя действия, которое будет выводиться в ГПИ;
- В поле `Description` описание действия;
- В выпадающем списке `Action Type` выбрать тип действия;
- В выпадающем списке `Entity Type` указать сущность, с которой работает пользовательское действие;
- Выбрать радиокнопку `Site selection mode`, которая отвечает за отображение пользовательского действия на Сайтах:
  - если установлена `Show on all sites except selected`, то действие будет отображаться на всех Сайтах, кроме выбранного;
  - если установлена `Hide on all sites except selected`, то действие не будет отображаться на всех Сайтах, кроме выбранного.
- В поле `Selected sites` указать Сайт(-ы), на котором(-ых) доступно пользовательское действие;
- Выбрать радиокнопку `Content selection mode`, которая отвечает за отображение пользовательского действия в контентах:
  - если установлена `Show in all contents except selected`, то действие будет отображаться во всех контентах, кроме выбранного;
  - если установлена `Hide in all contents except selected`, то действие будет отображаться во всех контентах, кроме выбранного.
- В поле `Selected contents` указать контент(-ы), в котором(-ых) доступно пользовательское действие;
- В поле `URL`, указывается ссылка на веб-приложение или метод. URL имеет следующую форму: `http://dpchost/action/Action?param`, где:
  - `dpchost` - хост веб приложения DPC;
  - `Action` - название обработчика пользовательского действия.
  - `Param` - дополнительные параметры. Параметры могут быть общие для всех обработчиков, а могут быть индивидуальные. Подробная информация о дополнительных параметрах описана в разделе «Дополнительные параметры пользовательского действия».
- В поле `Icon URL`, указывается ссылка на пиктограмму пользовательского действия;
- В поле `Order` указывается число, по которому сортируются пользовательские действия. Чем меньше число, тем выше/левее действие располагается в контекстном меню/на панели инструментов DPC;
- Указать флагом в каком действии доступно настраиваемое пользовательское действие;
- Кликнуть по кнопке «Сохранить».

Пользовательские действия преимущественно используется для обработки статей продуктов DPC.

Примечание: дополнительная настройка «Фраза подтверждения» (Confirm Phrase) позволяет задать сообщение, которое выводится при подтверждении выполнения действия. Тип вводимых данных – строка. Однако при необходимости использовать в сообщении содержимое поля «Title» пользователь может добавить в текст элемент `{0}` (Рисунок 7.14).

Фраза подтверждения: Уверены, что хотите восстановить из архива {0}?

Рисунок 7.14. Фраза подтверждения

Дополнительные параметры пользовательского действия

Дополнительные параметры задаются в следующем формате:

```
http://dpchost/action/Action?param
```

где:

`param` – дополнительные параметры передаваемые приложению. Один из таких параметров `Adapter`.

`Adapter` – параметр `Adapter`, отвечает за способ обработки пользовательского действия. Адаптер распределяет обработку данных между немедленной обработкой и отложенной обработкой по расписанию. Если адаптер не подключен, то продукт обрабатывается сразу.

Описание значений, принимаемых параметром `Adapter`, приведены в таблице 12.

Таблица 12. Описание принимаемых значений параметром Adapter

Название адаптера	Описание
`TaskActionAdapter`	Адаптер добавляет запрос на обработку в очередь. Очередь обрабатывает отдельный сервис, при обработке сообщения выводится соответствующее сообщение. Обработка продуктов не блокирует бэкенд.
`CountActionAdapter`	Адаптер устанавливает условие: если количество продуктов для обработки меньше или равно значению параметра `Count`, то продукты будут обработаны сразу. Иначе все продукты будут добавлены в очередь на обработку
`TimebasedActionAdapter`	Адаптер ставит задачу в очередь задач сервиса ActionsService, если выполнение задачи превысило время, указанное в параметре `Intarval`. Время устанавливается в секундах. Задачи, выполненные в установленный интервал, не попадают в очередь

Существует два вида дополнительных параметров пользовательского действия: универсальные и уникальные.

Например, универсальным считается параметр `Adapter`. Уникальным параметр live в пользовательском действии «Продукт».

Примечание: набор параметров необходимо уточнять у разработчика пользовательского действия.

Редактирование пользовательского действия

Для редактирования пользовательского действия необходимо кликнуть по пользовательскому действию. Клик по кнопке вызовет новую вкладку, описанную выше. Для редактирования необходимо внести правки в соответствующие поля и кликнуть по кнопке «Сохранить».

7.1.3. Описание пользовательских действий

В таблице 13 указаны пользовательские действия, являются ли они интерфейсными, тип сущности, с которой работают действия, тип действия и адрес расположения.

Таблица 13. Пользовательские действия

Название	Интерфейсное ли действие	Тип сущности	Тип действия	Адрес
Восстановить	Нет	Archive Article	Multiple Restore	http://admin_url/action/RestoreAction?Adapter=TaskActionAdapter
Продукт	Да	Article	Read	http://admin_url/product/index?filters=NotArchivedFilter
Продукт Live	Да	Article	Read	http://admin_url/product/index?live=true&filters=NotArchivedFilter&filters=VisibleFilter&filters=PublishedFilter
Каналы	Да	Site	Preview	http://admin_url/Notification
Xml Продукта	Да	Article	Read	http://admin_url/product/getxml
XAML продукта	Да	Article	Read	http://admin_url/product/GetProductData?formatter=XamlProductFormatter
Json Продукта	Да	Article	Read	http://admin_url/product/GetProductData?formatter=JsonProductFormatter
Мои задачи	Да	Site	Preview	http://admin_url/Task?showOnlyMine=true&Notify=true
Все Задачи	Да	Site	Save	http://admin_url/Task?allowSchedule=true
Публикация продуктов по ID	Да	Site	Update	http://admin_url/PartialSend?IgnoredStatus=Created
Удалить из архива	Нет	Archive Article	Archive Remove	https://admin_url/action/DeleteAction?Adapter=CountActionAdapter&Count=5
HighloadFront	Да	Site	Preview	http://admin_url/HighloadFront
Скачать XML	Нет	Article	Update	http://admin_url/product/GetXmlDownloadJson
Скачать live XML	Нет	Article	Read	http:/admin_url/ /product/GetXmlDownloadJson?live=true
Клонировать	Нет	Article	Create Like	http://admin_url/action/CloneBatchAction?Adapter=TaskActionAdapter
Публиковать	Нет	Article	Multiple Update	http://admin_url/action/PublishAction?IgnoredStatus=Created
Отправить JSON	Нет	Article	Multiple Update	http://admin_url/action/SendProductAction?IgnoredStatus=Created&IgnoredStatus=None&skipPublishing=True&skipLive=False&Channels=LiveJson&Adapter=CountActionAdapter&Count=0
Отправить на stage	Нет	Article	Multiple Update	http://admin_url/action/SendProductAction?IgnoredStatus=Created&skipPublishing=True&skipLive=True&Adapter=TaskActionAdapter
Отправить Json на stage	Нет	Article	Multiple Update	http://admin_url/action/SendProductAction?IgnoredStatus=Created&IgnoredStatus=None&skipPublishing=True&skipLive=True&Channels=StageJson&Adapter=CountActionAdapter&Count=0

Редактировать описание local	Да	Article	Update	`http://admin_url/DefinitionEditor`
Архивировать	Да	Article	Multiple Update	`http://admin_url/action/ArchiveAction?Adapter=CountActionAdapter&Count=5`
Удалить	Нет	Article	Multiple Remove	`http://admin_url/action/DeleteAction?Adapter=CountActionAdapter&Count=5`
Валидация	Нет	Article	Multiple Update	`http://admin_url/action/ValidationAction?IgnoredStatus=Created&Localize=true`
Definition Editor	Да	Article	Update	`https://admin_url/DefinitionEditor`
Definition Editor (Редактор описаний) **Действие выполняется вызовом метода `DefinitionEditor` и позволяет редактировать описания продуктов. Подробнее – см. Редактор описаний.**				
Пользовательские действия для маркетинговых продуктов				
Публиковать маркетинговый продукт	Нет	Article	Multiple Update	`http://admin_url/action/MarketingPublishAction?IgnoredStatus=Created&Localize=true`
Клонировать маркетинговый продукт	Нет	Article	Create Like	`http://admin_url/action/MarketingCloneAction?Adapter=TaskActionAdapter`
Архивировать маркетинговый продукт	Нет	Article	Multiple Archive	`http://admin_url/action/MarketingArchiveAction&Adapter=TaskActionAdapter`
Восстановить маркетинговый продукт	Нет	Archive Article	Multiple Restore	`http://admin_url/action/MarketingRestoreAction&Adapter=TaskActionAdapter`
Удалить маркетинговый продукт	Нет	Article	Multiple Remove	`http://admin_url/action/MarketingDeleteAction`
Удалить маркетинговый продукт из архива	Нет	Archive Article	Multiple Remove	`http://admin_url/action/MarketingDeleteAction?Adapter=TaskActionAdapter`

Восстановить

Пользовательское действие «Восстановить» выполняется вызовом метода `RestoreAction` и позволяет восстановить статью из архива.

По умолчанию действие не активно. Если выбран элемент или к определенному элементу вызвано контекстное меню, то действие активно.

Принимаемые методом параметры указаны в таблице 14.

Таблица 14. Входные параметры метода «RestoreAction»

Название параметра	Описание

`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)
`Count`	Задается при выборе параметра CountActionAdapter (см. выше)

Действие доступно:

1. На панели инструментов группы архивных статей контента `Продукты` (рис. 7.15);

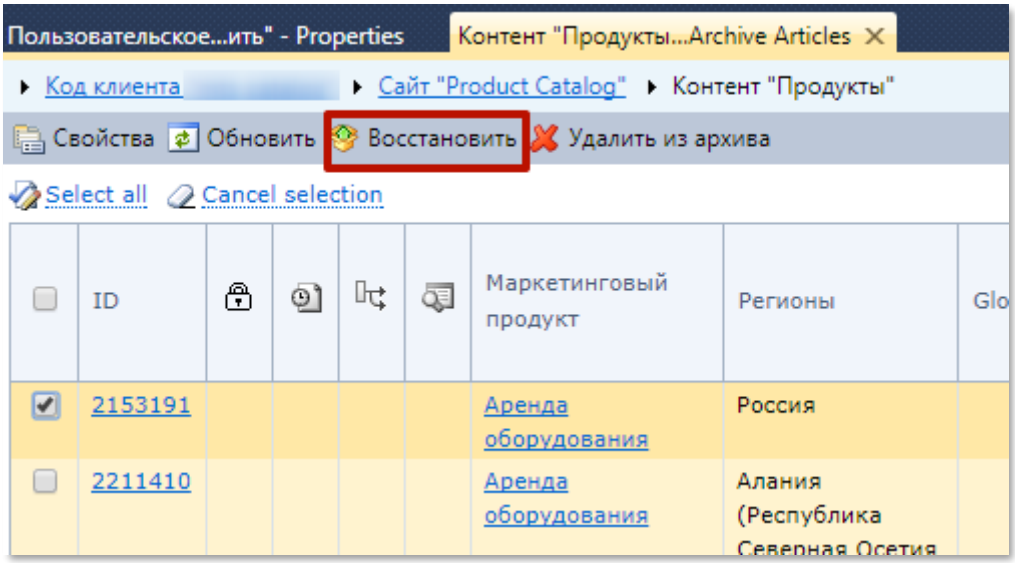


Рисунок 7.15. Пользовательское действие «Восстановить» на панели инструментов

2. В контекстном меню списка архивных статей контента `Продукты` (рис. 7.16).

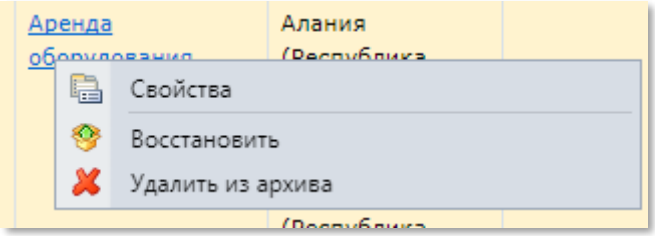


Рисунок 7.16. Пользовательское действие «Восстановить» в контекстном меню

Примечание: панель инструментов, выбор элементов списка, список архивных статей и контекстное меню описаны в приложении А.

Продукт, Продукт live

Пользовательское действие «Продукт» выполняется вызовом метода `index` и позволяет получить агрегированную информацию о продукте.

Принимаемые методом параметры указаны ниже (табл. 15).

Таблица 15. Входные параметры метода «index»

Название параметра	Описание
`filters`	Фильтр, который устанавливает условия для сборки продукта. Принимаемые значения: • `NotArchivedFilter` 3. `VisibleFilter`

	4. PublishedFilter
live	Если параметра имеет значение true, то продукт собирается с live-окружения
Localize	Если параметр имеет значение true, то продукт будет представлен в соответствующей локализации (при условии, что существует необходимая локализация). Если требуемой локализации нет, то будет использована первая зарегистрированная в DPC. Также будет возможность смены языка в карточке продукта. Подробнее в разделе Локализация DPC
includeRelevanceInfo	Если параметр имеет значение true, то проверяется актуальность продукта на витрине
lang	Язык, в котором будет собираться продукт

Если метод вызван с параметром filters с установленным значением NotArchivedFilter, то выполнится пользовательское действие «Продукт».

Если метод вызван с параметром live с установленным значением true, то выполнится пользовательское действие «Продукт live».

Действие доступно:

1. На панели инструментов контента Продукты (рис. 7.17);

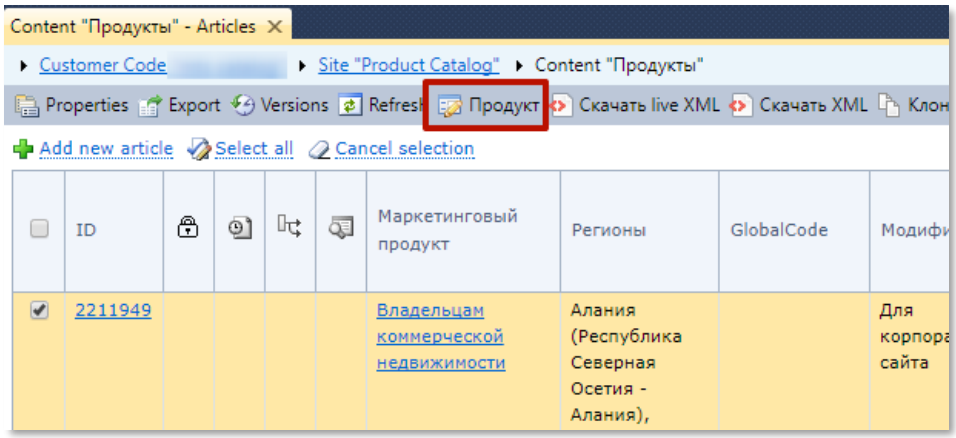


Рисунок 7.17. Пользовательское действие «Продукт» на панели инструментов

2. В контекстном меню списка статей контента Продукты (рис. 7.18).

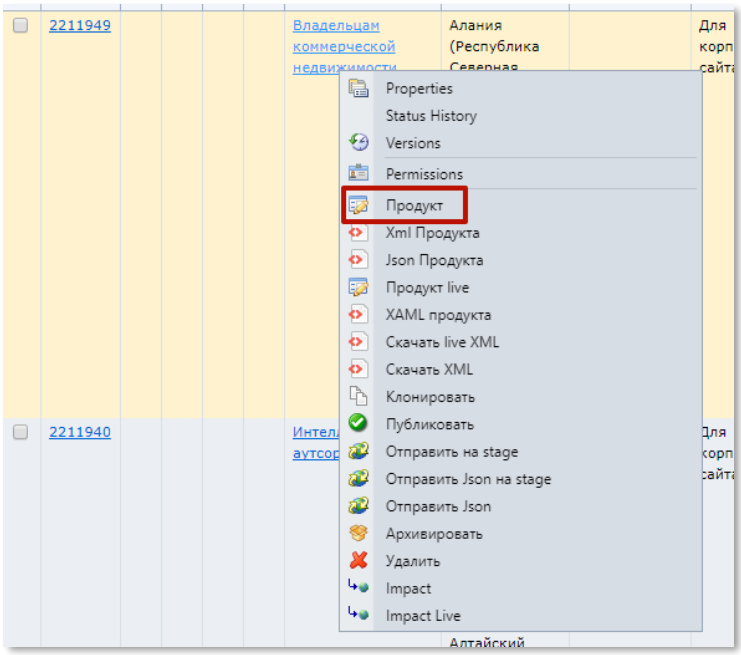


Рисунок 7.18. Пользовательское действие «Продукт» в контекстном меню

Примечание: панель инструментов, список статей и контекстное меню описаны в приложении А.

Каналы

Пользовательское действие «Каналы» выполняется вызовом метода `Notification` и позволяет получить сведения о публикациях продуктов в каналы службой NotificationSender (см. Настройка канала публикации).

Действие доступно в контекстном меню Сайта (рис. 7.19). Подробное описание в подразделе «ГПИ службы. Функциональные возможности ГПИ».

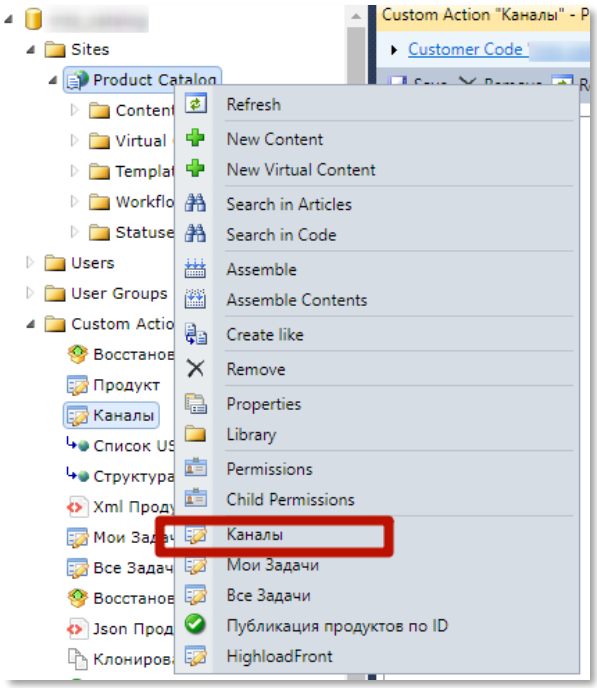


Рисунок 7.19. Пользовательское действие «Каналы» в контекстном меню Сайта

Xml Продукта

Пользовательское действие «Xml Продукта» выполняется вызовом метода `getxml` и позволяет получить xml-разметку статьи продукта.

Действие доступно:

1. На панели инструментов, после вызова пользовательского действия «Продукт» (рис. 7.20);

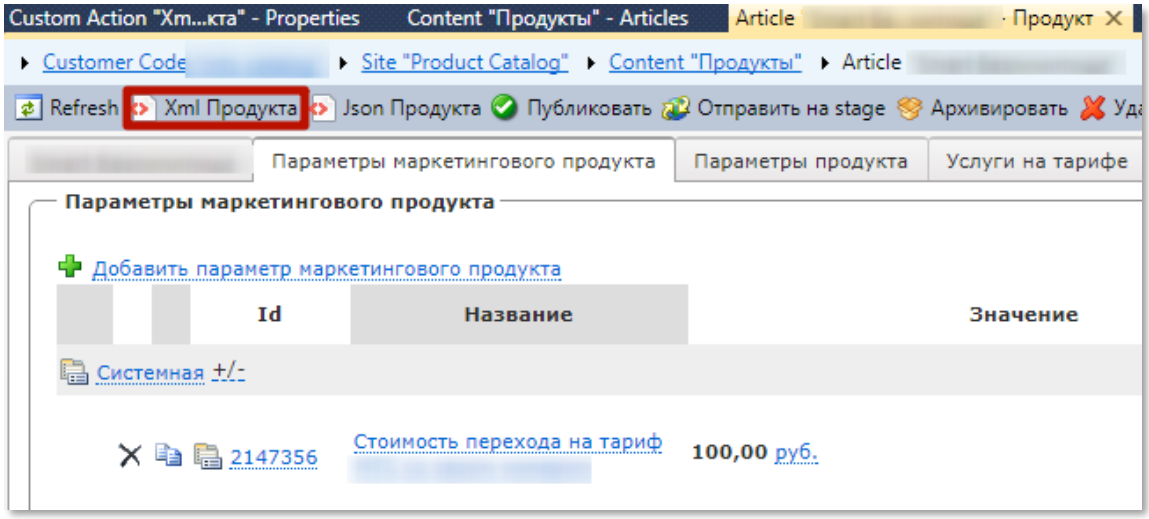


Рисунок 7.20. Пользовательское действие «Xml Продукта» на панели инструментов

2. В контекстном меню списка статей контента «Продукт» (рис. 7.21).

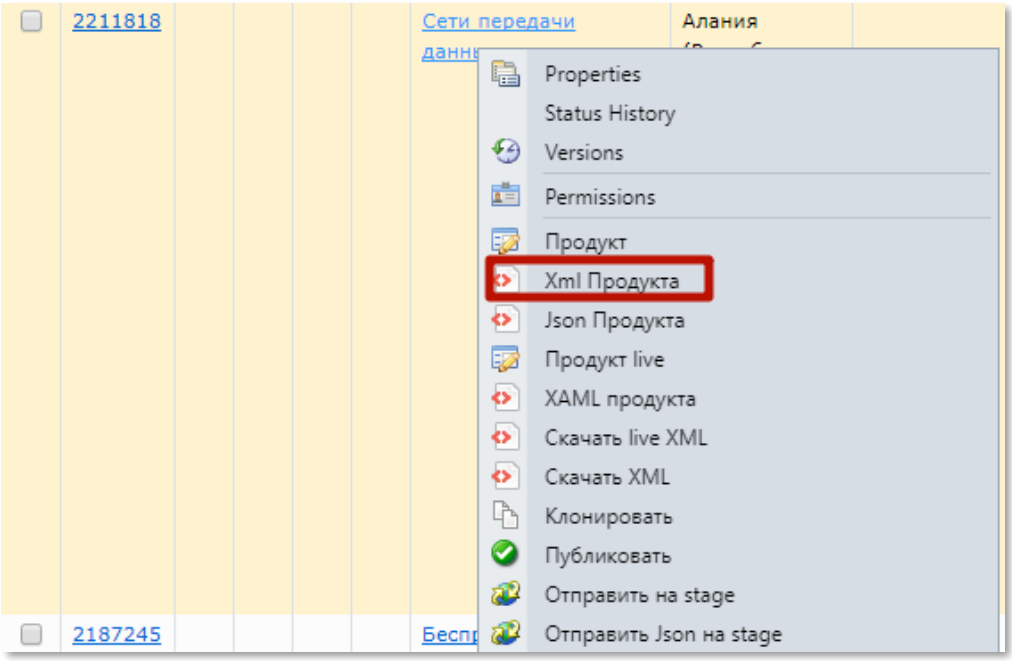


Рисунок 7.21. Пользовательское действие «Xml Продукта» в контекстном меню

XAML, JSON продукта

Пользовательское действие выполняется вызовом метода `GetProductData` и позволяет получить разметку статьи продукта в требуемом формате. Формат разметки передается параметром `formatter`.

Принимаемые методом параметры указаны ниже (табл. 16).

Таблица 16. Входные параметры для метода «GetProductData»

Название параметра	Описание
`formatter`	Задает формат разметки продукта. Принимает значения доступные в DPC форматов, которые регистрируются в контенте `Форматеры` (см. Форматеры).

Если метода вызван с параметром `formatter` с установленным значением `XamlProductFormatter`, то будет выполнено действие «XAML Продукта».

Если метода вызван с параметром `formatter` с установленным значением `JsonProductFormatter`, то будет выполнено действие «Json Продукта».

Действие доступно в контекстном меню списка статей контента в виде пункта «*Формат* Продукта», где формат это XAML или JSON. Также на панели инструментов, после вызова пользовательского действия «Продукт»;

Мои задачи, Все задачи

Пользовательское действие выполняется вызовом метода `Task`.

Методы «Мои задачи», «Все задачи» позволяют получить информацию от службы ActionsService (см. Сервис выполнения отложенных задач ActionsService).

Принимаемые методом параметры указаны ниже (табл. 17).

Таблица 17. Входные параметры метода «Task»

Название параметра	Описание
`showOnlyMine`	Если параметр принимает значение `true`, то будет выполнено действие «Мои задачи»
`Notify`	Если параметр принимает значение
`allowSchedule`	Если параметр принимает значение `true`, то будет выполнено действие «Все задачи»

Пользовательское действие «Мои Задачи» позволяет получить информацию о состоянии задач, запущенных пользователем. Для выполняющихся задач отображается прогресс выполнения. Для некоторых задач возможны отмена выполнения и перезапуск.

Пользовательское действие «Все Задачи» позволяет получить информацию о состоянии задач, запущенных пользователями Системы. Для выполняющихся задач отображается прогресс выполнения. Для некоторых задач возможны отмена выполнения и перезапуск.

Публикация продуктов по ID

Пользовательское действие «Публикация продуктов по ID» выполняется вызовом метода `PartialSend` и позволяет публиковать продукты по списку их идентификаторов, а также удалять с витрин неактуальные продукты.

Принимаемые методом параметры указаны ниже (табл. 18).

Таблица 18. Входные параметры метода «PartialSend»

Название параметра	Описание
`IgnoredStatus`	Если параметр установлен, то не публикуются продукты со специальным статусом.

	По умолчанию установлено значение `Created`

Данный блок предназначен для быстрой публикации региональных продуктов по списку их идентификаторов, удаления с витрин неактуальных продуктов.

Действие находится в контекстном меню Сайта (рис. 7.22).

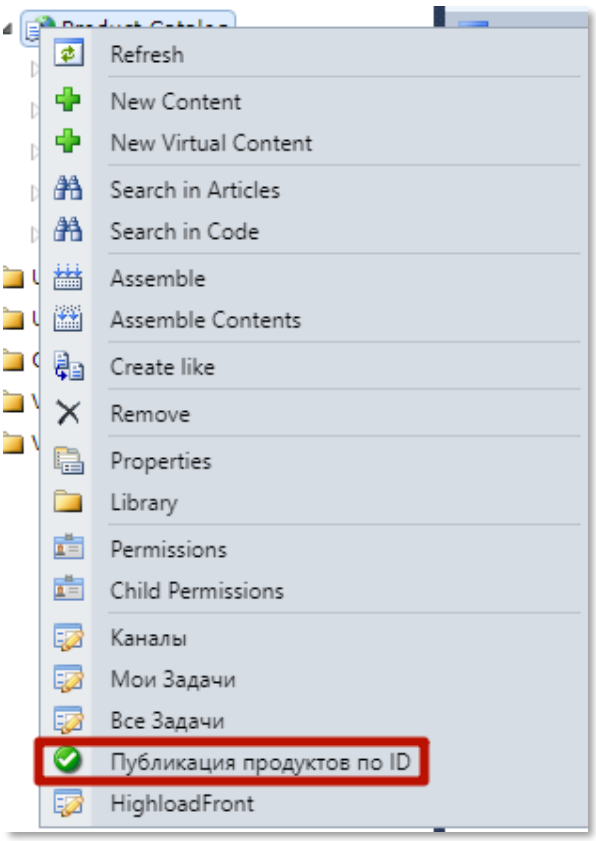


Рисунок 7.22. Пользовательское действие «Публикация продуктов по ID» в контекстном меню

Инструмент «Публикация продуктов по ID» изображен ниже (рис. 7.23).

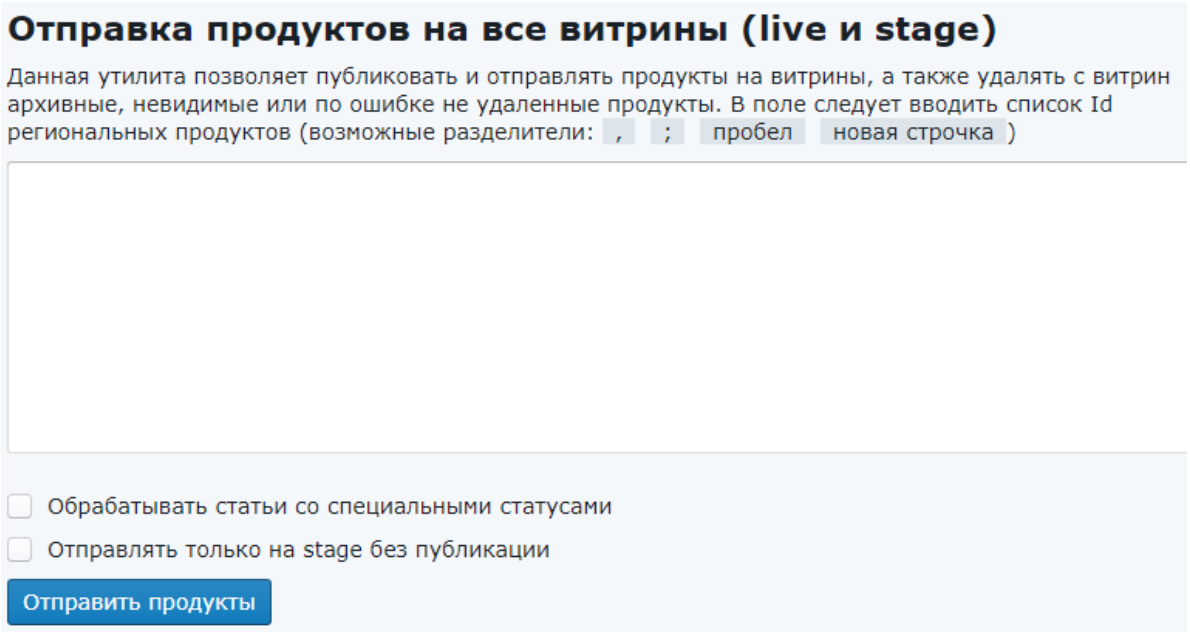


Рисунок 7.23. Инструмент «Публикация продуктов по ID»

В текстовую область вводится список идентификаторов продуктов, разделенные запятой, точкой с запятой, пробелом или переводом каретки на новую строку.

Под полем ввода идентификаторов расположены две опции:

- Признак «Обрабатывать статьи со специальными статусами». Если признак установлен, то не публикуются продукты со специальными статусами;
- Если установлен признак «Отправлять только на stage без публикации», то статьи публикуются только в stage-окружении и не публикуются на live.

После ввода списка идентификаторов и нажатия по кнопке «Отправить продукты» отображается статус и прогресс выполнения (рис. 7.24). Прогресс выполнения отображается в процентах.

Рисунок 7.24. Процесс публикации списка продуктов

После окончания выполнения задачи отображается информация о результатах выполнения публикации, кнопка для отправки нового списка продуктов «Отправить новый пакет» (рис. 7.25).

Рисунок 7.25. Результат выполнения публикации списка продуктов

Примечание: продукты, которые не были найдены в DPC, то в результате выполнения задачи указываются идентификаторы всех ненайденных продуктов (рис. 7.26).

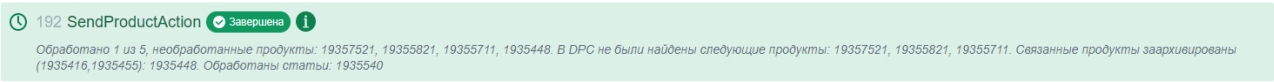


Рисунок 7.26. Необработанные продукты

Ход выполнения задачи также отображается в интерфейсе «Мои задачи» или «Все задачи».

Примечание: продукт(-ы) уделенный(-е) из базы DPC, но присутствующий(-е) на витрине, удаляется(-ются) если задать их идентификаторы в списке публикации по ID (рис. 7.27).

Рисунок 7.27. Удаленные продукты

Примечание: если веб-браузер поддерживает показ уведомлений и открыта вкладка «Мои Задачи», то выводится уведомление о результатах выполнения задачи публикации списка продуктов (рис. 7.28).

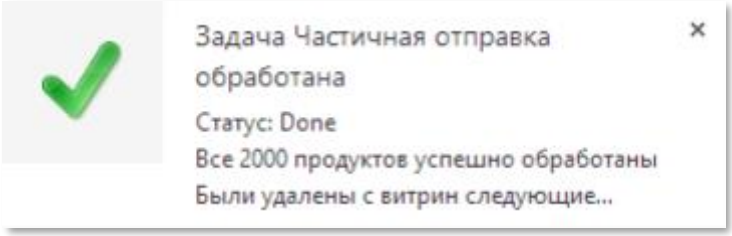


Рисунок 7.28. Уведомление о завершении задачи публикации списка продуктов

Удалить из архива

Пользовательское действие «Удалить из архива» выполняется вызовом метода `DeleteAction` и позволяет удалить продукт из архива статей. При удалении продукта учитывается его описание (см. Описание продуктов), где определяется как именно удаляется продукт: какие связанные сущности удаляются и т.п.

Принимаемые методом параметры указаны в таблице 19.

Таблица 19. Входные параметры метода «DeleteAction»

Название параметра	Описание

Adapter	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)
Count	Задается при выборе параметра CountActionAdapter (см. выше)

Действие доступно:

1. На панели инструментов списка архивных статей контента Продукты (рис. 7.29);

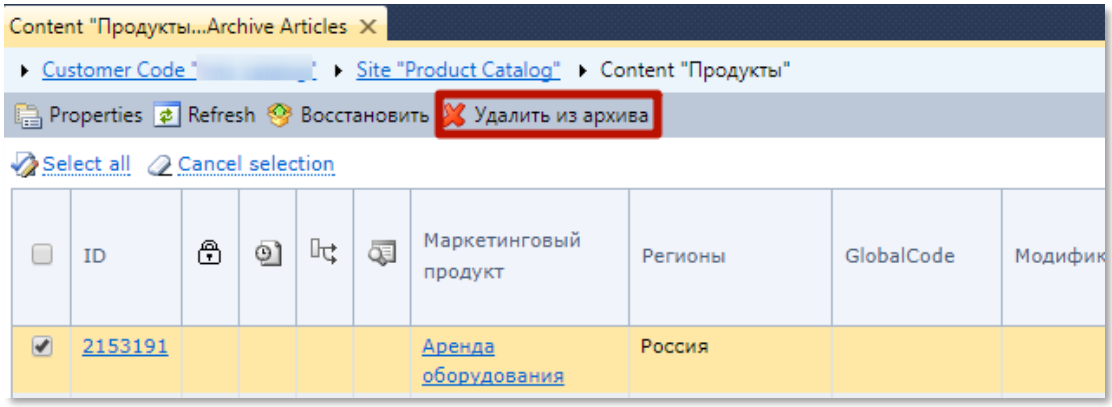


Рисунок 7.29. Пользовательское действие «Удалить из архива» на панели инструментов списка архивных статей

2. В контекстном меню списка архивных статей контента Продукты (рис. 7.30);

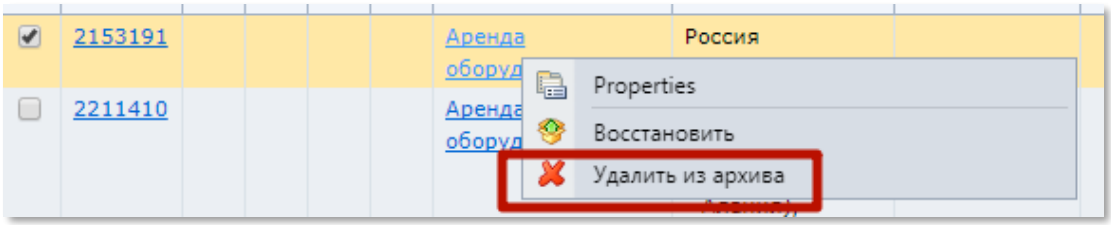


Рисунок 7.30. Пользовательское действие «Удалить из архива» в контекстном меню

3. На панели инструментов формы редактирования архивной статьи (рис. 7.31).

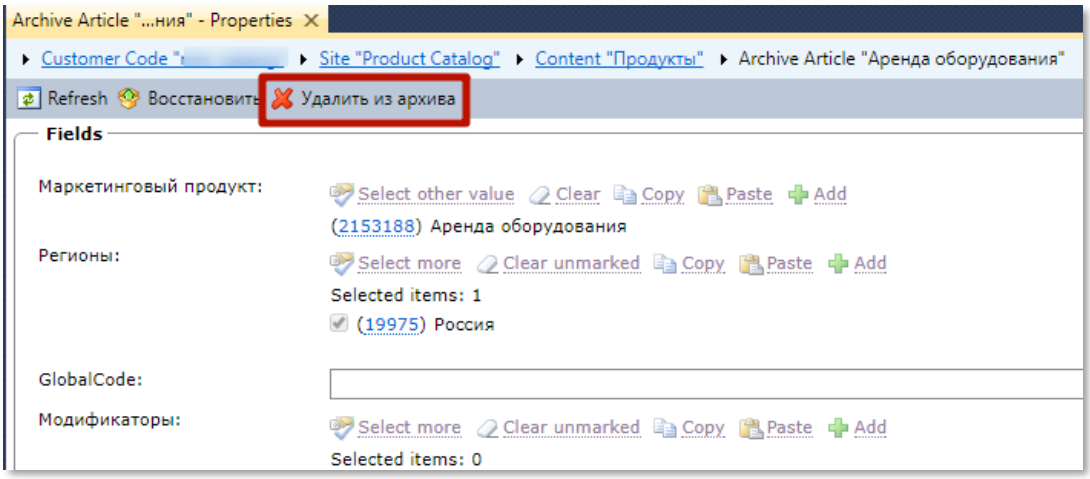


Рисунок 7.31. Пользовательское действие «Удалить из архива» в форме редактирования архивной статьи

HighloadFront (Высокопроизводительная витрина)

Общие сведения

Пользовательское действие выполняется вызовом метода `HighloadFront`.

Позволяет получить информацию об индексах, указанных в контенте `Индексы Elastic` (рис. 7.32) а также выполнять их индексацию и переиндексацию.

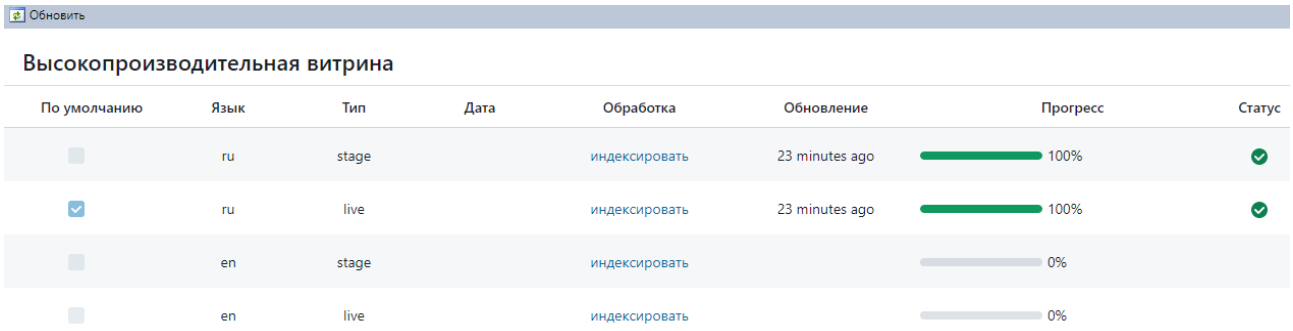


Рисунок 7.32. ГПИ управления Elastic-витринами

Для вызова пользовательского действия необходимо кликнуть правой кнопкой мыши по Сайту и выбрать пункт «HighloadFront».

Конфигурация о каналах представлена в виде таблицы, которая состоит из следующих полей:

- По умолчанию (Default) – если флаг установлен, то конфигурация используется по умолчанию если задается неполный адрес обращения к Elastic Sync API;
- Язык (Language) – язык представления продуктов;
- Тип (Type) – окружение, в которое выгружаются продукты. Может принимать значение live или stage;
- Дата (Date) – дата, за которую выводится состояние витрины;
- Обработка (Processing) – псевдоссылка, позволяющая запустить индексацию. Выполняет вызов метода переиндексации;
- Обновление (Updating) – отображает время последнего обновления состояния индексации;
- Прогресс (Progress) – индикатор процесса, отображает ход выполнения индексации в процентах;
- Статус (Status) – иконка со статусом последней индексации (если такие были за время работы приложения).

Механизм переиндексации

При индексации данных продуктового каталога в Elastic название индекса задаётся в QP в разделе «Служебные» – «Индексы Elastic» – в поле «Имя индекса» (Рисунок 7.33).

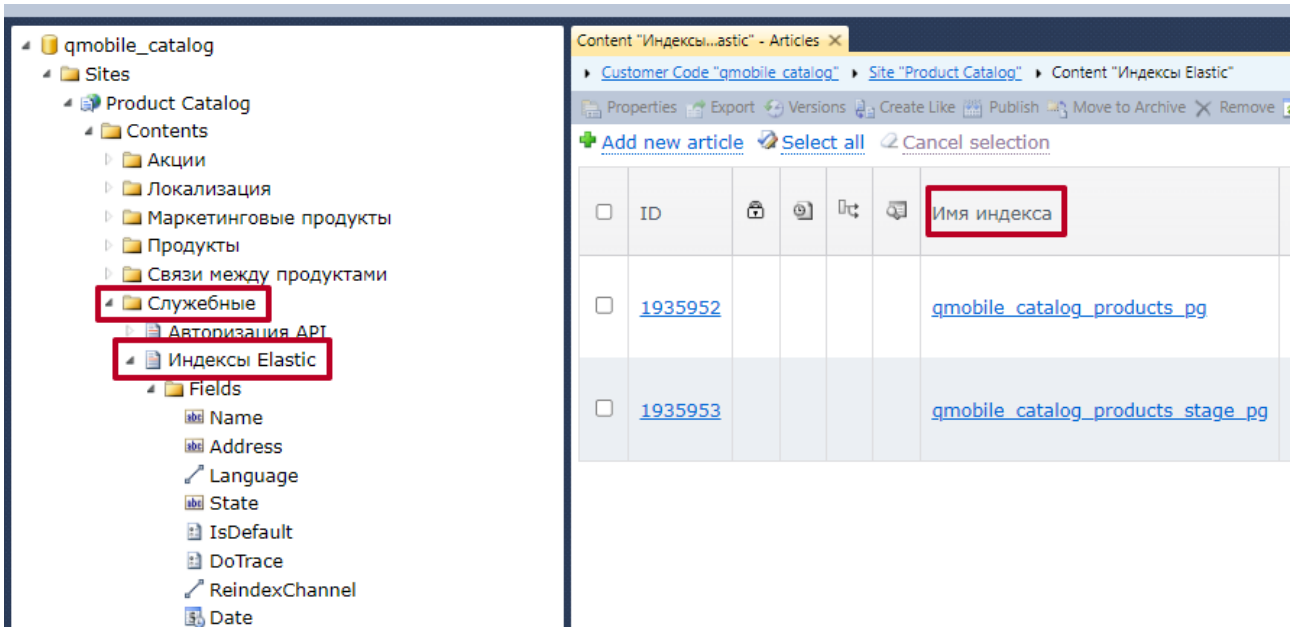


Рисунок 7.33. Индексация данных в QP

Обновлённый механизм переиндексации задействует **алиасы индексов**. Каждый алиас ссылается на конкретный индекс. Это позволяет сначала создавать новый индекс с новым именем, а затем (при безошибочном завершении переиндексации) переключать на него алиас. Только после этого удаляется старый индекс.

При создании алиаса его имя задаётся из того же поля «Имя индекса». В имя самого индекса добавляется дата и время его создания (по UTC) в формате «alias.yyyy-MM-ddtHH-mm-ss», где:

- alias – имя из поля «Имя индекса»,
- yyyy – год,
- MM – месяц,
- dd – день,
- HH – часы в 24-часовой нотации,
- mm – минуты,
- ss – секунды.

Например, если в поле «Имя индекса» задано «qmobile_catalog_products_pg» и индексация запускается 12.12.2023 в 14:11:11 по МСК, то индекс будет называться «qmobile_catalog_products_pg.2023-12-12t11-11-11».

Обновлённый механизм переиндексации работает следующим образом:

1. Получение имени алиаса из QP.
2. Поиск всех индексов, которые начинаются с этого имени (могут различаться по дате и/или времени создания). Все обнаруженные индексы фиксируются механизмом индексации.
3. Создание нового индекса с именем описанного выше формата.
4. Загрузка актуальных данных в индекс.
5. Переключение алиаса на новый индекс.
6. Удаление старых индексов, имена которых получены в шаге 2.

Примечание: если в процессе переиндексации происходит ошибка, каждый созданный в процессе индекс остаётся «висеть» в Elastic. С каждой неудачной попыткой число таких индексов растёт. Однако при первой успешной переиндексации, после переключения алиаса на новый индекс, все старые индексы (включая бывший рабочий и все ошибочные) будут удалены.

Преимущества данного механизма:

- непрерывность работы: на время процесса переиндексации старый индекс остаётся доступен с тем же именем алиаса и все запросы идут на него (функционирование системы не прекращается);
- безотказность: в случае ошибки переинедксации алиас не переключаются на новый индекс, все запросы продолжают использовать старый индекс. В интерфейсе DPC при этом отобразится сообщение об ошибке переиндексации.

Скачать XML, Скачать live XML

Пользовательское действие выполняется вызовом метода `GetXmlDownloadJson`.

Пользовательское действие «Скачать XML» позволяет скачать XML-разметку продукта (рис. 7.34).


```
<ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Products>
    <Product xsi:type="Tariff">
      <Id>1713794</Id>
      <Description><![CDATA ]></Description>
      <FullDescription><![CDATA[<p> России и 10 Гб Интернета всего за 550 рублей в месяц<
<p>Пакеты минут, SMS и интернета сохраняются <a href="[replacement]tag=ab[/replacement]transfer/">Подробнее</a></p
<p>Спутниковое  без абонентской платы <a href="[replacement]tag=ab[/replacement]dom/sputnik_tv/discount/smar
      <SortOrder>3</SortOrder>
      <ForisID>21225</ForisID>
      <Icon>//spbdev01 pload/images/products/msk/unlim.png</Icon>
      <PDF>
        <Name> pdf</Name>
        <FileSizeBytes>0</FileSizeBytes>
        <AbsoluteUrl>//spbdev01/ /upload/images/products/msk/smart_bezlimit_msk_311017.pdf</AbsoluteUrl>
      </PDF>
      <MarketingProduct>
        <Id>1713791</Id>
        <Title> /Title>
        <SortOrder>15</SortOrder>
        <ListImage>//spbdev01/ /upload/contents/383/ g</ListImage>
        <DetailsImage>//spbdev01/ /upload/contents/383/ jpg</DetailsImage>
        <Alias>smart_bezlimitishhe</Alias>
        <Type>MarketingTariff</Type>
        <Section>
          <Id>19625</Id>
          <Title>Частным клиентам
          <OldSiteId>1</OldSiteId>
        </Section>
        <CommunicationType>
          <Id>338802</Id>
          <Title>Мобильная связь</Title>
          <Alias>mobile</Alias>
        </CommunicationType>
```


Рисунок 7.34. XML-разметка продукта

Принимаемые методом параметры указаны в таблице ниже.

Таблица 20. Входные параметры метода «GetXmlDownloadJson»

Название параметра	Описание
`live`	Если параметр принимает значение true, то будет загружена XML-разметка продукта с live-окружения

Действие доступно:

1. На панели инструментов списка продуктов (рис. 7.35);

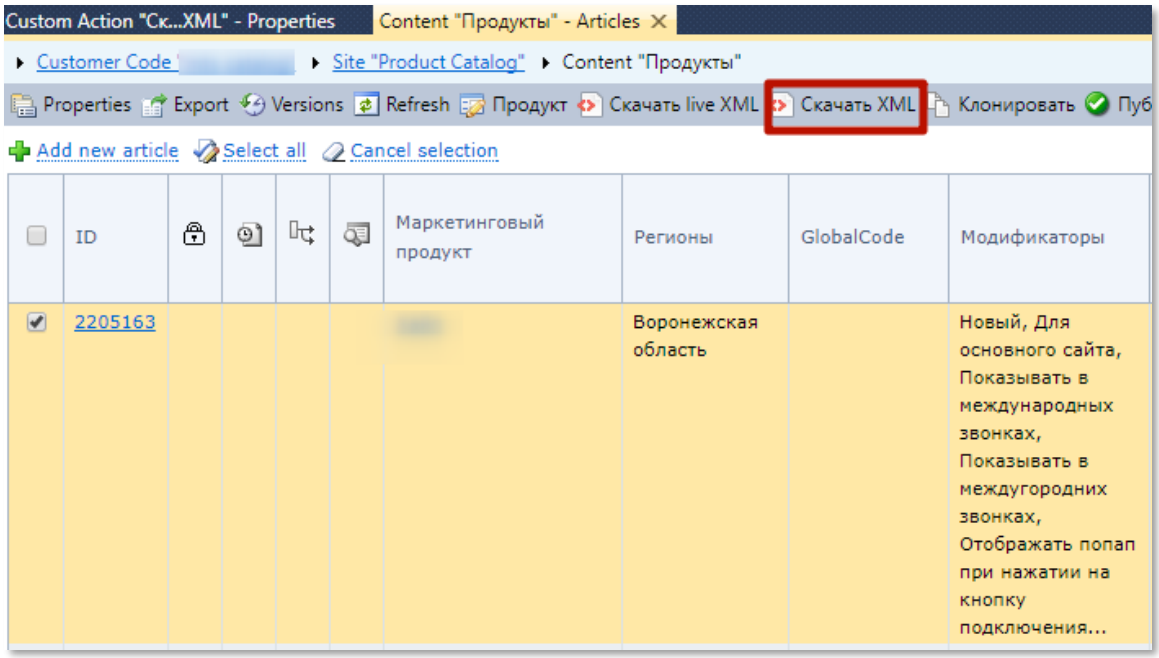


Рисунок 7.35. Пользовательское действие «Скачать XML» на панели управления списка продуктов

2. В контекстном меню списка продуктов (рис. 7.36);

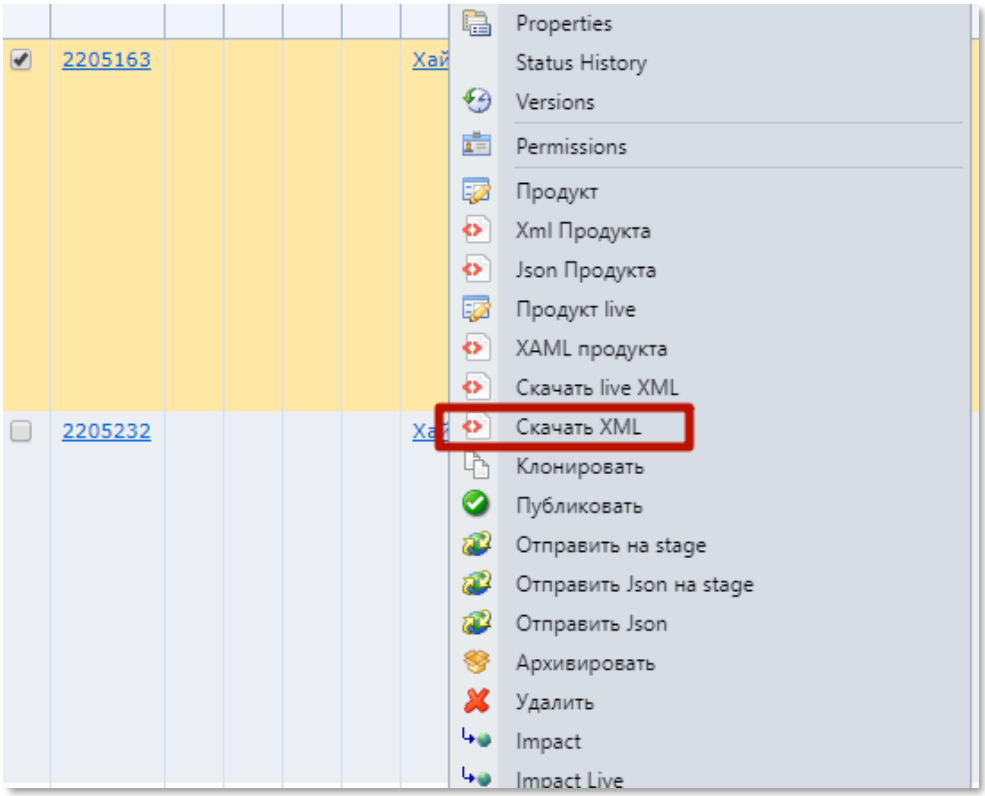


Рисунок 7.36. Пользовательское действие «Скачать XML» в контекстном меню

3. На панели инструментов формы редактирования архивной статьи (рис. 7.37).

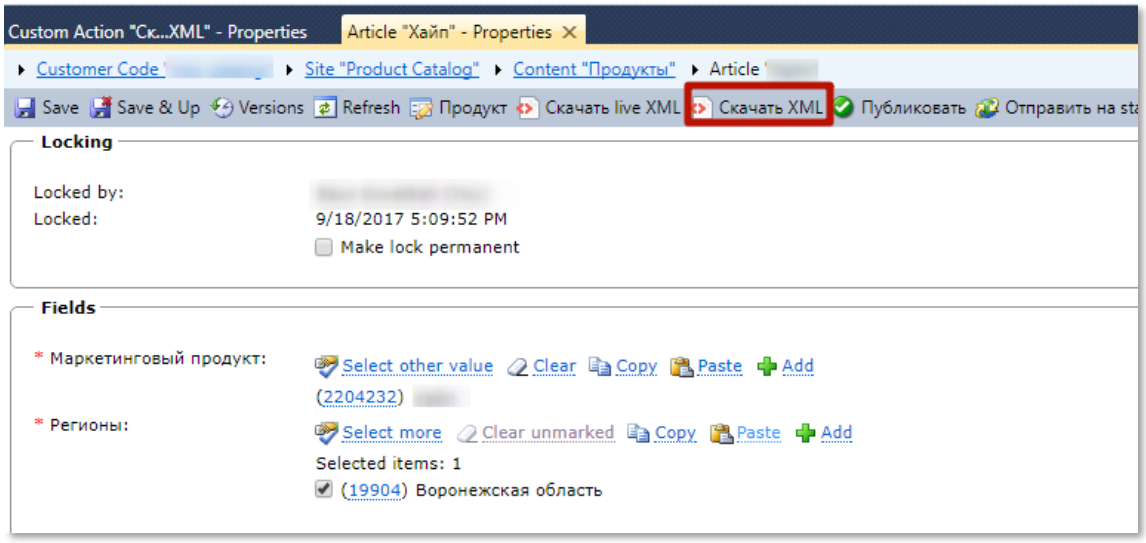


Рисунок 7.37. Пользовательское действие «Скачать XML» в форме редактирования статьи

Если метод, вызывающий действие «Скачать XML», будет вызван с параметром `live`, со значением `true`, то будет вызвано действие «Скачать live XML». Действие позволяет получить XML-разметку продукта, опубликованного в live – окружении.

Клонировать

Пользовательское действие «Клонировать» выполняется вызовом метода `CloneBatchAction` и позволяет создать копию продукта. Вызов действия создает задачу, которая выполняется с помощью сервиса выполнения отложенных задач. Процесс выполнения задачи доступен через вызов действия «Мои задачи» и «Все задачи».

При копировании продукта учитывается его описание (см. Описание продуктов), где определяется как именно копируется продукт: какие связанные сущности копируются и т.п.

Клонированному продукту присваивается новый идентификатор.

Принимаемые методом параметры указаны в таблице 21.

Таблица 21. Входные параметры метода «CloneBatchAction»

Название параметра	Описание
`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)

Действие доступно:

1. На панели инструментов списка продуктов (рис. 7.38);

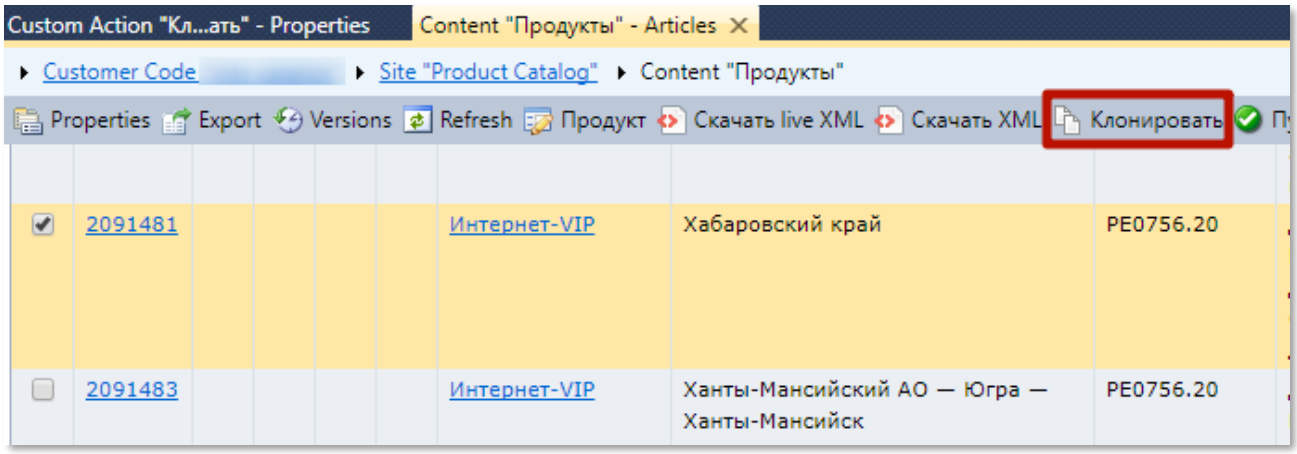


Рисунок 7.38. Пользовательское действие «Клонировать» на панели инструментов

2. В контекстном меню списка продуктов (рис. 7.39).

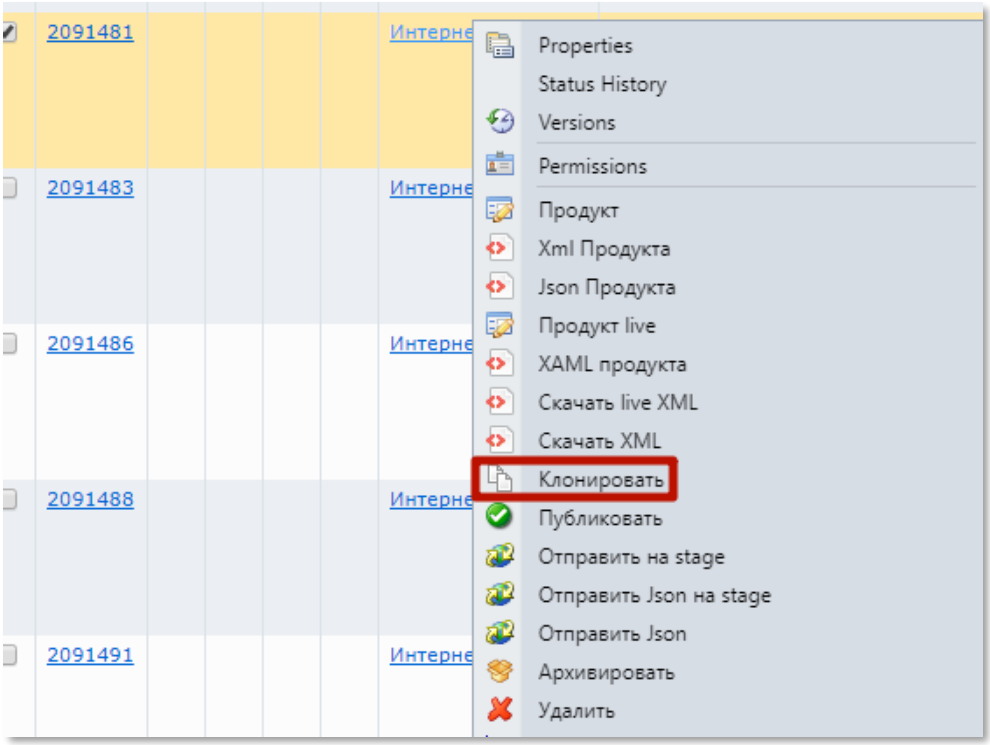


Рисунок 7.39. Пользовательское действие «Клонировать» в контекстном меню

Публиковать

Пользовательское действие выполняется вызовом метода `PublishAction` и позволяет опубликовать продукт в live и stage окружения.

Принимаемые методом параметры указаны в таблице 22.

Таблица 22. Входные параметры метода «PublishAction»

Название параметра	Описание
`IgnoredStatus`	Если параметр установлен, то не публикуются продукты со специальным статусом. По умолчанию установлено значение `Created`
`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия).
`Localize`	Если параметр принимает значение `true`, то продукт собирается с учетом локализации (см. Локализация DPC)
`formatter`	Задает формат разметки продукта. Принимает значения доступные в DPC форматов, которые регистрируются в контенте `Форматеры` (см. Форматеры).
`lang`	Дополнительный параметр. Работает в совокупности с параметром `formatter`, задает язык продукта в указанном формате

Действие доступно:

1. На панели инструментов списка продуктов (рис. 7.40);

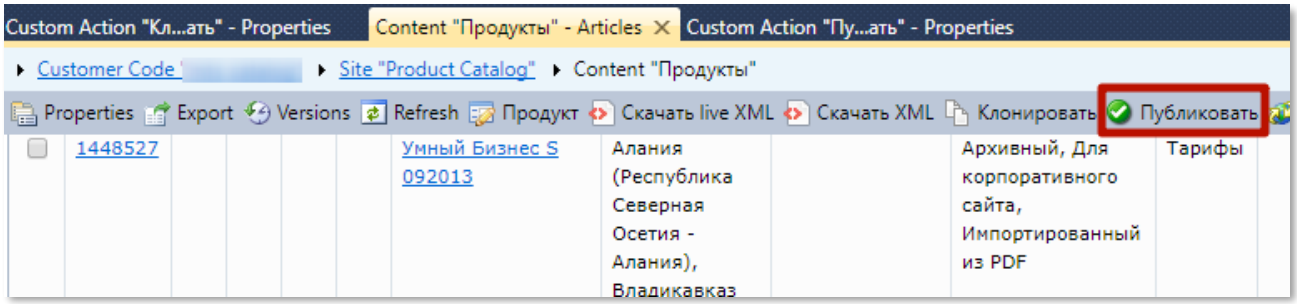


Рисунок 7.40. Пользовательское действие «Публиковать» на панели инструментов

2. В контекстном меню списка продуктов (рис. 7.41);

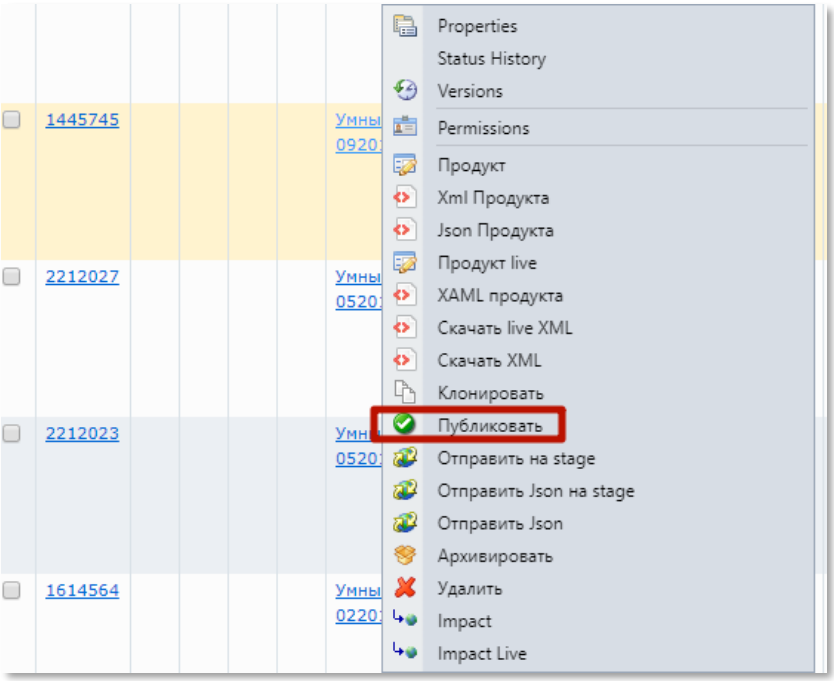


Рисунок 7.41. Пользовательское действие «Публиковать» в контекстном меню

3. В форме редактирования статьи продукта (рис. 7.42).

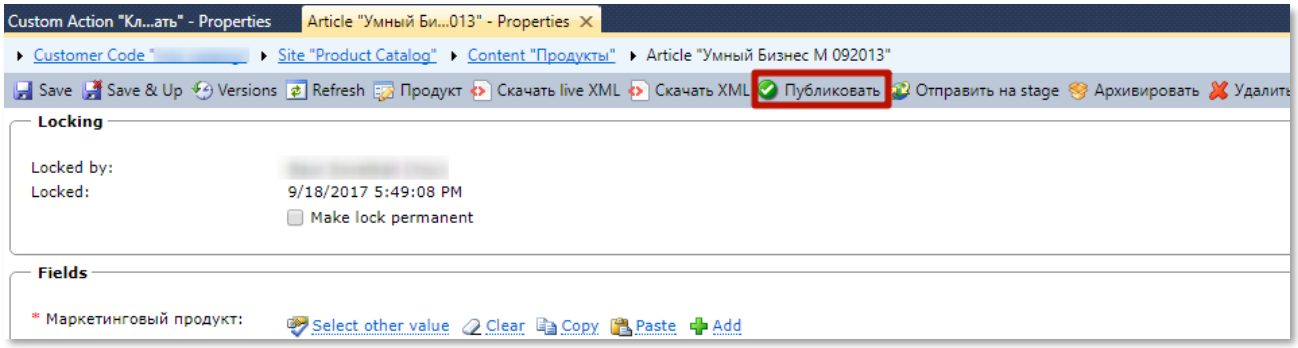


Рисунок 7.42. Пользовательское действие «Публиковать» в форме редактирования статьи

Отправить JSON, Отправить на Stage, Отправить Json на Stage

Пользовательское действие выполняется вызовом метода `SendProductAction`.

Пользовательское действие позволяет отправить продукт на витрину.

Принимаемые методом параметры указаны в таблице 23.

Таблица 23. Входные параметры метода «SendProductAction»

Название параметра	Описание
`IgnoredStatus`	Если параметр установлен, то не публикуются продукты со специальным статусом. По умолчанию установлено значение `Created`
`skipPublishing`	Если значение параметра true, то
`skipLive`	Если значение параметре true, то продукт не будет отправлен в live-окружение
`Channels`	Указывает по какому каналу отправить продукт (см. Настройка канала публикации).
`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)
`Count`	Задается при выборе параметра CountActionAdapter (см. выше)

Для вызова пользовательского действия «Отправить Json» необходимо передать обработчику следующие параметры:

```
IgnoredStatus=Created&IgnoredStatus=None&skipPublishing=True&skipLive=False&Channels=LiveJson&Adapter=CountActionAdapter&Count=0
```

Продукты отправляются только на референсную JSON-витрину.

Для вызова пользовательского действия «Отправить на Stage» необходимо передать обработчику следующие параметры:

```
IgnoredStatus=Created&skipPublishing=True&skipLive=True&Adapter=TaskActionAdapter
```

Продукты отправляются в live и stage витрины.

Для вызова пользовательского действия «Отправить Json на Stage» необходимо передать обработчику следующие параметры:

```
IgnoredStatus=Created&IgnoredStatus=None&skipPublishing=True&skipLive=True&Channels=Sta
geJson&Adapter=CountActionAdapter&Count=0
```

Продукты отправляются только на референсную JSON-витрину.

Примечание: пользовательские действия «Отправить JSON» и «Отправить JSON на stage» публикуют продукты на референсную витрину. В случае если в DPC поставлялся без Elasticsearch, а позже был подключен, то данные из референсной витрины будут проиндексированы в Elasticsearch (см. HighloadFront).
Примечание: пользовательские действия, описанные выше, являются частью ElasticAPI. Если DPC поставляется без ElasticAPI, то этих пользовательских действий не будет, даже если существует JSON-витрина.

Архивировать

Вызываемый метод – `ArchiveAction`. Действие позволяет переместить продукт в группу «Архивные статьи». Также продукт удаляется с витрины. В контексте БД статье, отвечающей за продукт, присваивается признак «архивная».

Действие доступно на панели инструментов DPC и в контекстном меню, вызываемом для статьи.

Таблица 24. Входные параметры метода «ArchiveAction»

Название параметра	Описание
`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)
`Count`	Задается при выборе параметра CountActionAdapter (см. выше)

Позволяет архивировать несколько продуктов, выбранных в списке статей контента.

Удалить

Вызываемый метод – `DeleteAction`. Действие удаляет продукт без возможности восстановления. Также продукт удаляется с витрины.

При удалении продукта учитывается его описание (см. Описание продуктов), где определяется как именно удаляется продукт: какие связанные сущности удаляются и т.п.

Таблица 25. Входные параметры метода «DeleteAction»

Название параметра	Описание
`Adapter`	Отвечает за способ обработки пользовательского действия (см. Дополнительные параметры пользовательского действия)
`Count`	Задается при выборе параметра CountActionAdapter (см. выше)

Действие доступно на панели инструментов DPC и в контекстном меню, вызываемом для статьи.

Позволяет удалять несколько продуктов, выбранных в списке статей контента.

Валидация

Пользовательское действие выполняется вызовом метода `ValidationAction` и позволяет валидировать продукт без его публикации. Действие является множественным (распространяется на несколько продуктов).

Принимаемые методом параметры указаны в таблице ниже.

Таблица 26. Входные параметры метода «ValidationAction»

Название параметра	Описание
`IgnoredStatus`	Если параметр установлен, то не публикуются продукты со специальным статусом. По умолчанию установлено значение `Created`
`Localize`	Если параметр принимает значение `true`, то продукт собирается с учетом локализации (см. Локализация DPC)

Definition Editor (Редактор описаний)

Действие выполняется вызовом метода `DefinitionEditor` и позволяет редактировать описания продуктов. Подробнее – см. Редактор описаний.

Пользовательские действия для маркетинговых продуктов

В системе реализованы следующие пользовательские действия для маркетинговых продуктов (Таблица 27):

Таблица 27. Пользовательские действия для маркетинговых продутов

Действие для маркетинговых продуктов	Вызываемый метод	Аналогичное действие
Публиковать маркетинговый продукт	`MarketingPublishAction`	Публиковать
Клонировать маркетинговый продукт	`MarketingCloneAction`	Клонировать
Архивировать маркетинговый продукт	`MarketingArchiveAction`	Архивировать
Восстановить маркетинговый продукт	`MarketingRestoreAction`	Восстановить
Удалить маркетинговый продукт	`MarketingDeleteAction`	Удалить
Удалить маркетинговый продукт из архива	`MarketingDeleteAction`	Удалить из архива

Данные действия функционируют аналагично указанным в таблице. Отличным является способ обработки: действие поочередно распространяется на каждый региональный продукт соответствующего маркетингового продукта.

7.1.4. Пользовательская валидация

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

- ограничение ввода по типу поля;
- свойство поля «Обязательное»;
- свойство поля «Уникальное»;
- для поля типа «Строка»:
  - маска ввода;
  - длина значения.

Примечание: подробное описание пользовательской валидации приведено в приложении Б.

Remote-валидация

В QP существует возможность использовать remote-валидацию. В этом случае введённые пользователем данные передаются на проверку стороннему приложению. Для обмена данными со сторонним приложением используется формат 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">
<!-- Ключ валидатора определяет необходимый валидатор и его адрес -->
      <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>
```

7.1.5. Настройки конфигурационного файла

Пример синтаксиса:

```
{
  "Properties": {
    "Name": "DPC.Admin"
  },
  "Connection": {
    "DpcConnectionString": "Application Name=ActionsService;Initial Catalog=catalog;Data Source=server;User ID=user;Password=pass",
    "TasksConnectionString": "Application Name=ActionsService;Initial Catalog=tasks;Data Source=server;User ID=user;Password=pass",
    "QpMode": true,
    "UsePostgres": false,
    "TransactionTimeout": "00:10:00"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "System": "Warning"
    }
  },
  "Loader": {
    "UseFileSizeService": true
  },
  "S3": {
    "Endpoint": "minio:9000",
    "AccessKey": "minioLogin",
    "SecretKey": "minioPassword",
    "Bucket": "qp8"
  }
}
```

Параметры:

Таблица 28. Настройки конфигурационного файла DPC.Admin

Название	Тип данных	Описание
Properties	Объект	Общие настройки
Name	Строка	Имя приложения (для логов)
Connection	Объект	Раздел настроек подключения к БД

DpcConnectionString	Строка	Строка подключения к БД каталога (используется при QpMode = false)
TasksConnectionString	Строка	Строка подключения к БД задач (используется при QpMode = false)
LiveMonitoringConnectionString	Строка	Строка подключения к Live-референсной витрине (используется при QpMode = false, UseQpMonitoring = false)
StageMonitoringConnectionString	Строка	Строка подключения к Stage-референсной витрине (используется при QpMode = false, UseQpMonitoring = false)
QpMode	Булевый	Режим использования QP (если `true`, строки подключения берутся из конфигурационного файла QP или сервиса конфигурации для БД, в которых включен режим DPC)
UseQPMonitoring	Булевый	Режим, используемый при QPMode = false, в котором данные референсной витрины хранятся в самой базе каталога (таблица Products), а не в отдельной базе. Если true, то нужно задавать LiveMonitoringConnectionString и StageMonitoringConnectionString
UsePostgres	Булевый	Подключение к PostgreSQL либо SQL Server (используется при QpMode = false)
TransactionTimeout	Строка	Таймаут транзакции
Logging	Объект	Раздел настроек логирования
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
System	Строка	Минимальный уровень логирования для модулей System
Loader	Объект	Настройки компонента загрузки продуктов
LoaderWarmUpProductId	Число	ID продукта, используемый для «прогрева» кэшей
LoaderWarmUpRepeatInMinutes	Число	Интервал «прогрева» кэшей
UseFileSizeService	Булевый	Использовать ли сервис для получения размера файлов (QP.Storage) или пытаться вычислить размер файла из локальной файловой системы.

Integration	Объект	Настройки интеграции с другими сервисами
RestNotificationUrl	Строка	REST-адрес сервиса DPC.NotificationSender
HighloadFrontSyncUrl	Строка	Адрес сервиса DPC.SyncAPI
ExtraValidationLibraries	Массив строк	Имена дополнительных библиотек, загружаемых динамически и используемых для валидации
S3	**Объект**	**Настройки доступа к MinIO**
Endpoint	Строка	Адрес MinIO
AccessKey	Строка	Ключ доступа к MinIO (логин)
SecretKey	Строка	Секрет доступа к MinIO (пароль)
Bucket	Строка	Имя бакета

Настройка доступа к MinIO

Настройка глобального доступа к использованию MinIO – см. Руководство администратора QP8.

Настройка доступа на уровне приложения `DPC.Admin` выполняется в файле `appsettings.json`, в отдельной секции `S3` (содержимое аналогично настройкам в QP). Также необходимо переключить `Loader` из режима вызова storage-сервиса в режим использования файловой системы / MinIO для определения размера файлов (описание параметров – в таблице выше):

```
"S3": {
   "Endpoint": "minio:9000",
   "AccessKey": "minioLogin",
   "SecretKey": "minioPassword",
   "Bucket": "qp8"
},
"Loader": {
   "UseFileSizeService": false
}
```

Следующая функциональность DPC поддерживает MinIO:

- версионирование файловых полей при обновлении,
- определение размера файла при публикации.

7.2. Сервис выполнения отложенных задач (DPC.ActionsService)

Сервис является windows-службой, который выполняет пользовательские действия в порядке очереди и обновляет информацию о состоянии задачи. Сервис взаимодействует с БД. БД хранит очередь задач. Сервис вносит правки в очередь.

ГПИ сервиса доступен через вызов пользовательских действий: Все задачи, Мои задачи. ГПИ сервиса позволяет наблюдать за прогрессом выполнения задач.

Также сервис позволяет публиковать продукты по расписанию.

В настройках пользовательского действия задается адаптер. Адаптер позволяет определить, как обрабатывать вызов действия. Существует 3 вида адаптеров:

Сервис ActionsService выполняет следующие задачи:

- Частичная публикация продуктов;
- Публикация продуктов на витрины;
- Любые неинтерфейсные действия QP.

Сервис использует следующие БД:

- dpc_tasks – БД хранит задачи, которые требуется выполнить;
- dpc_catalog – БД содержит статьи, из которых состоит продукт. Сервис читает информацию о продуктах;
- dpc_xml – БД используется для сверки отправленных продуктов и продуктов хранящихся в dpc_catalog.

Процесс выполнения задач отображается с помощью пользовательских действий «Мои задачи» и «Все задачи».

Сервис выполняет задачи, а QP и DPC Admin выполняет пользовательские действия. В DPC существует функционал, который может выполняться как задача, так и как пользовательское действие. Пользовательские действия по умолчанию являются таким функционалом, например, «Клонировать», «Удалить» и т.п.

Адаптер пользовательского действия (см. Дополнительные параметры пользовательского действия), который указывается в настройках пользовательского действия, определяет где будет выполняться действие, в DPC Admin или в ActionsService.

Если адаптер не указан, то ГПИ бэкэнда блокируется и пользователю необходимо ждать выполнения задачи. Но ожидание выполнения имеет положительную сторону, в таком случае задача выполняется в реальном времени. В случае же выполнения задачи с помощью сервиса накладывает временные затраты на то, чтобы поставить задачу в очередь, ожидание выполнения задачи в очереди и т.п. Также если пользовательское действие вызывается из таблицы статей, то после выполнения действия данные в таблице обновляются. Задачи, которые выполняются быстро, подходят в этом случае.

7.2.1. Мои задачи

Действие находится в контекстном меню Сайта (рис. 7.43).

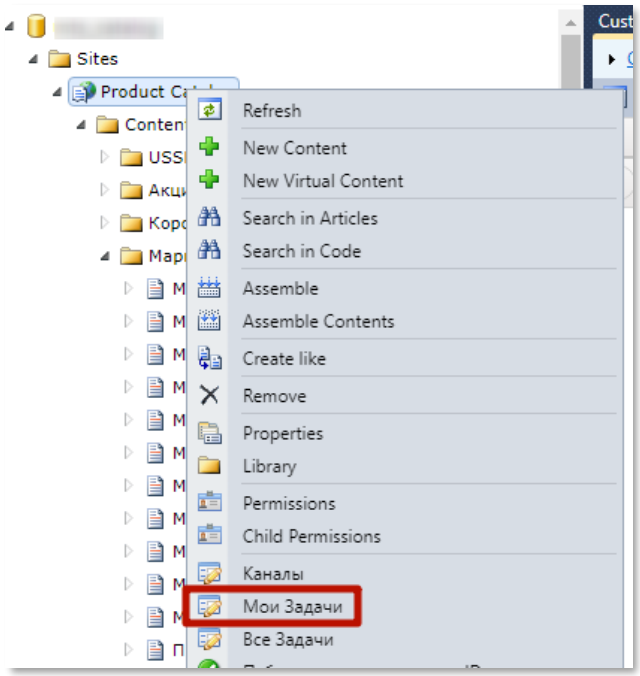


Рисунок 7.43. Пользовательское действие «Мои Задачи» в контекстном меню

На странице, вызванной действием, размещены следующие элементы:

- Блок «Ваша последняя задача», которая выводит детальную информацию о последней выполняемой задаче пользователя (рис. 7.44);

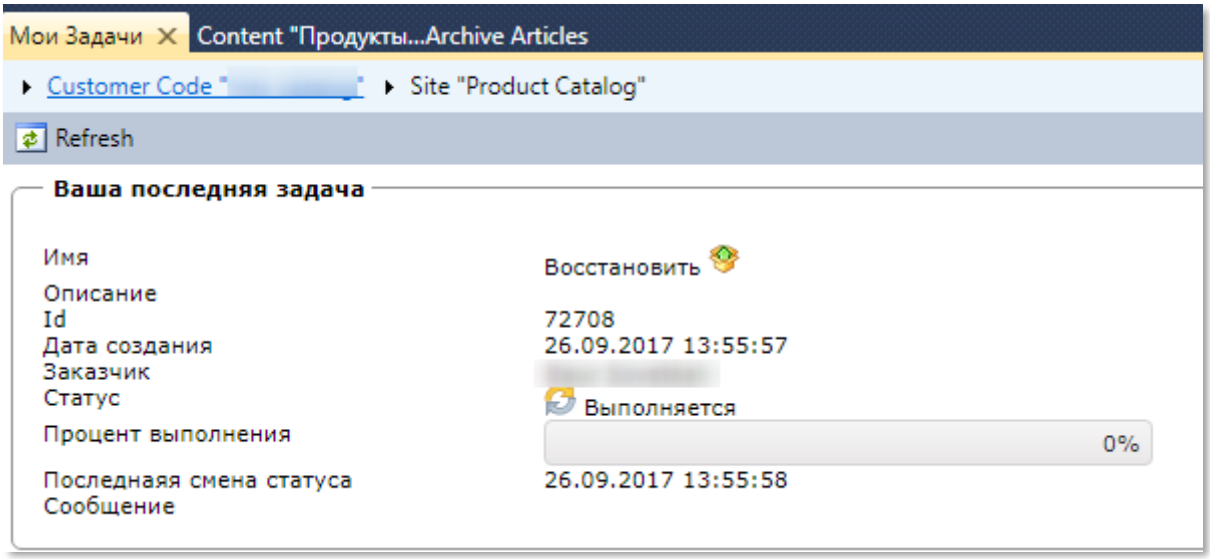


Рисунок 7.44. Блок «Ваша последняя задача»

- Список задач пользователя (рис. 7.45).

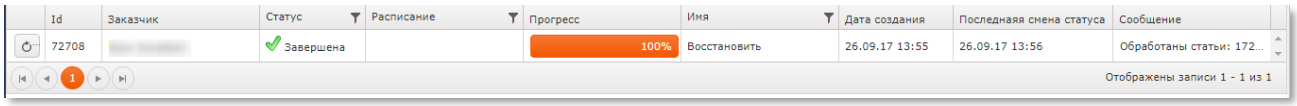


Рисунок 7.45. Список задач пользователя

По каждой из задачи можно получить следующую информацию:

- Заказчик. Пользователь QP, который запустил поставил задачу в очередь. В случае с пользовательским действием «Мои задачи» в поле выводится имя текущего пользователя;
- Статус. Сообщает о состоянии задачи и может принимать значения:
  - Ошибка: в ходе выполнения задачи возникли ошибки;
  - Завершена: выполнение задачи завершилось успешно;
  - Выполняется: задача выполняется сервисом;
  - Отклонена: выполнение задачи прервано кликом по пиктограмме «».
- Расписание (см. Автопубликация (отложенная публикация));
- Прогресс. Графический индикатор, который информирует о прогрессе выполнения работы в процентах;
- Дата создание. Дата создания задачи в формате `ДД.ММ.ГГГГ ЧЧ:ММ`;
- Имя. По умолчанию содержит название пользовательского действия;
- Последняя смена статуса;
- Сообщение. Текстовое описание статуса задачи. Например, «Обработаны задачи: …».

Для удобства пользователя предусмотрены уведомления о выполнении задач. Уведомления работают в последних версиях следующих веб-браузерах: Firefox, Chrome, Safari, Opera.

Если пользователь подтвердил показ уведомлений, то по окончанию выполнения задачи отобразится уведомление. Несколько уведомлений складываются в стек (Рисунок 7.46).

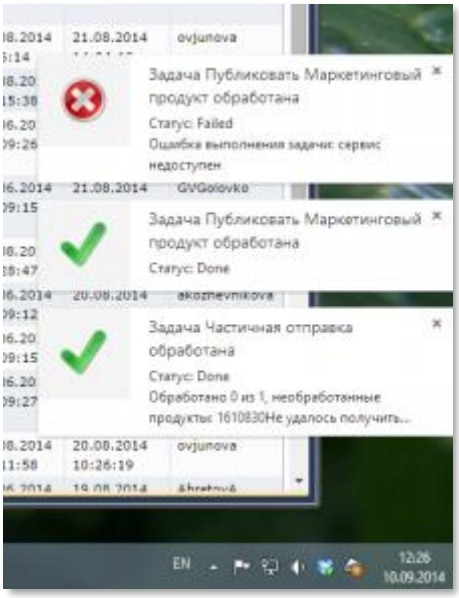


Рисунок 7.46. Уведомления QP8

Уведомления работают для задач, каналов и для HighloadFront. Разрешить показ можно в личном профиле кликнув на «Разрешить push-уведомления».

Если пользователь ранее не получал уведомлений, то выводится подтверждение браузер на разрешение показа уведомления. Подтверждение в браузере Chrome изображено ниже (Рисунок 7.47).

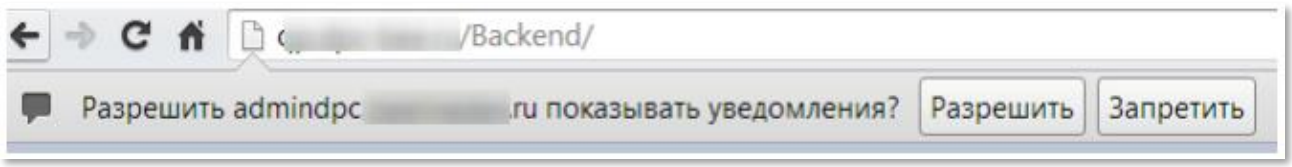


Рисунок 7.47. Подтверждение вывода уведомлений

Для того, чтобы получать уведомления QP необходимо нажать на кнопку «Разрешить».

Примечание. Актуальная информация о правах доступа отражается в консоли, при использовании http-соединения консоль выдаст ошибку (рис. 7.48).

Разрешение на отправку уведомлений может быть предоставлено только при защищённом соединении.

Рисунок 7.48. Ошибка в консоли про отправку уведомлений

Если пользователь подтвердил показ уведомлений, то по окончанию выполнения задачи отобразится уведомление (рис. 7.49).

Рисунок 7.49. Пример уведомления

Если задача поддерживает отмену и была отменена, то выводится уведомление об отмене публикации (рис. 7.50).

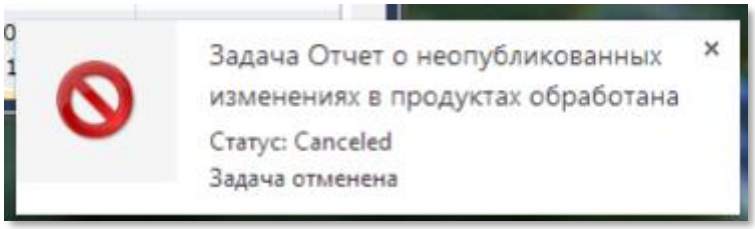


Рисунок 7.50. Уведомление QP8 об отмене публикации задачи

Если уведомления не отображаются, то следует:

- Проверить поддерживает ли веб-браузер Html5 Notifications;
- Проверить не запрещены ли уведомления в веб-браузере;
- Проверить не запрещены ли уведомления для данного домена.

Для того, чтобы посмотреть разрешения на показ уведомлений необходимо:

- Ввести в адресную строку ссылку: `chrome://settings/content`;
- В появившейся вкладке выбрать пункт «Оповещения».

Во вкладке с оповещениями убедиться, что в группе «Разрешить» добавлен адрес размещения QP8. Если адреса нет, то добавить ссылку кликнув по псевдоссылке «Добавить» в группе «Оповещения».

Примечание. Уведомления в браузер приходят от приложения QP8, хотя их могут посылать разные источники: QP8, DPC и другие приложения. При этом DPC и другие приложения отправляют запрос в QP8, чтобы отправить уведомление. На текущий момент функциональность отправки уведомлений через QP8 реализована только для DPC, но разрешение получается на все такие случаи вне зависимости от фактического источника уведомлений.

7.2.2. Все задачи

Действие находится в контекстном меню Сайта (рис. 7.51).

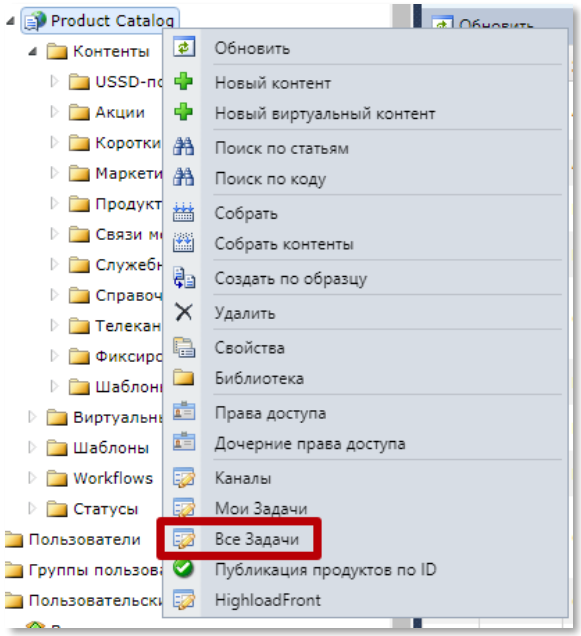


Рисунок 7.51. Пользовательское действие «Все Задачи» в контекстном меню

Примечание: пользовательское действие аналогично описанному в подразделе «Мои Задачи». Отличием является то, что в списке задач выводятся задачи всех пользователей CMS.

7.2.3. Фильтрация задач

Список задач поддерживает фильтрацию по полям (рис. 7.52):

- Статус;
- Расписание;
- Имя.

Рисунок 7.52. Поля с фильтрами

Для фильтрации необходимо кликнуть по пиктограмме ( ).

Клик по пиктограмме:

- Поля `Статус` выводит выпадающий список со статусами (рис. 7.53). Задачи фильтруются по выбранному статусу;

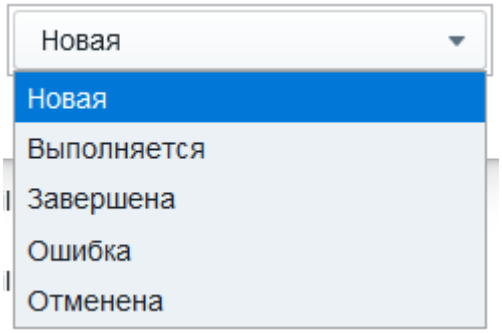


Рисунок 7.53. Фильтрация по полю «Статус»

- Поля Расписание выводит радиокнопки (рис. 7.54). Если выбрана радиокнопка «Да», то выводятся задачи, которые должны быть опубликованы или уже опубликованы по расписанию. Если выбрана радиокнопка «Нет», то выводятся задачи, которые публикуются или опубликованы не по расписанию.

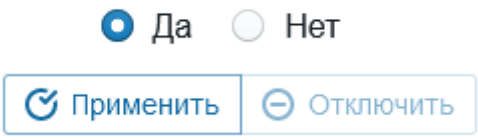


Рисунок 7.54. Фильтрация по полю «Расписание»

- Поля Имя поле ввода названия задачи (рис. 7.55).

Рисунок 7.55. Фильтрация по полю «Имя»

- Поля Создано выводит поля для ввода диапазона дат необходимых для фильтрации (рис. 7.56).

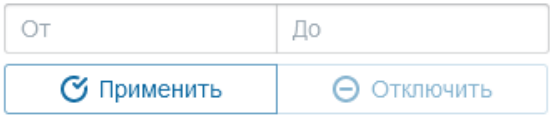


Рисунок 7.57. Фильтрация по полю «Создано»

Для применения условия фильтрации необходимо настроить фильтр и кликнуть по кнопке «Применить». Для отмены фильтрации кликнуть по кнопке «Отключить».

Задачи фильтруются по нескольким полям по логическому условию «И», т.е. выводятся задачи, соответствующие всем указанным фильтрам.

7.2.4. Расписание задач

На вкладке «Все задачи» также есть возможность посмотреть и изменить расписание задач. С помощью фильтра можно выделить только те задачи, на которое уже назначено расписание (рис. 7.*58*). В интерфейсе для задач с назначенным расписанием заполнен столбец Расписание, в котором указано время разового выполнения, либо текстовое описание интервала повторения. Переход к созданию или редактированию расписания выполняется с помощью клика по пиктограмме. Активное расписание отмечено пиктограммой « » , а неактивное или отсутствующее расписание аналогичным символом синего цвета « ».

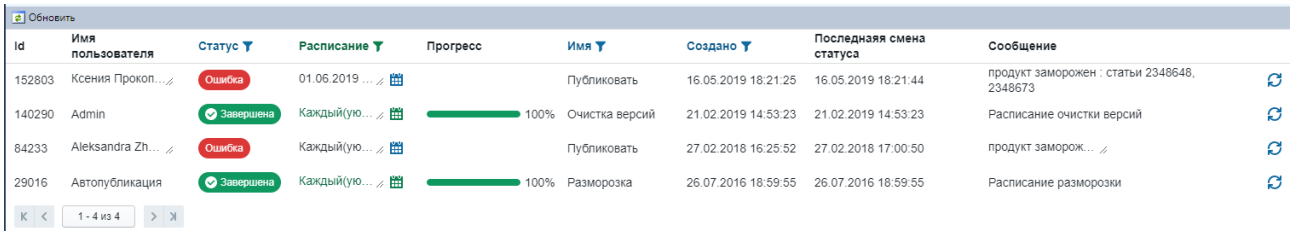


Рисунок 7.58. Отфильтрованные по расписанию задачи

В состав QP8.ProductCatalog входит действие UnreezeAction, которое может быть настроено для выполнения по расписанию. Например, на рис. 7.*58* по расписанию выполняется задача Разморозка, которая ищет в продукте поле FreezeDate и, если дата истекла, выполняет публикацию продукта от имени пользователя Автопубликация.

При редактировании расписания задачи показывается следующая форма (рис. 7.*59*):

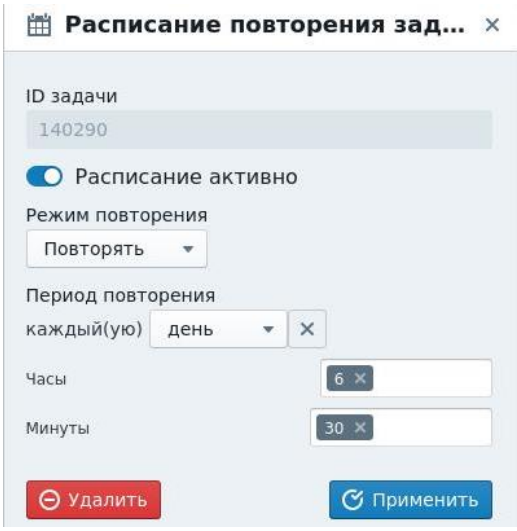


Рисунок 7.59. Форма редактирования расписания задачи

На форме доступны:

- Признак «Расписание активно». Если признак установлен, то задача активна, т.е. будет выполнена в указанное время. Если признак сброшен, то задача не будет выполнена. Признак позволяет выключить задачу, не удаляя ее;
- Режим повторения:
  - Повторять – задается период повторения задачи и время, в которое она выполняется в рамках указанного периода (рис. 7.*59*);
  - Одноразовое – задается дата и время выполнения задачи.
- Кнопка Сохранить позволяет сохранить новое или измененное расписание для текущей задачи.
- Кнопка Удалить позволяет удалить расписание для текущей задачи. Если требуется отключить задачу, но не удалять её, можно выключить признак Расписание активно и сохранить изменения.

7.2.5. Настройки конфигурационного файла

Пример синтаксиса:

```
{
```

```
    "Connection": {
        "DpcConnectionString": "Application Name=ActionsService;Initial Catalog=catalog;Data
Source=server;User ID=user;Password=pass",
        "TasksConnectionString": "Application Name=ActionsService;Initial Catalog=tasks;Data
Source=server;User ID=user;Password=pass",
        "DesignConnectionString": "Application Name=ActionsService;Initial
Catalog=tasks_design;Data Source=server;User ID=user;Password=pass",
        "QpMode": true,
        "UsePostgres": false,
        "TransactionTimeout": "00:10:00"
    },
    "Properties": {
        "Name" : "DPC.ActionsService",
        "EnableScheduleProcess": true
    },

    "Logging": {
        "IncludeScopes": false,
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "System": "Warning"
        }
    },
    "Loader":
    {
        "UseFileSizeService": true
    },
   "S3": {
        "Endpoint": "minio:9000",
        "AccessKey": "minioLogin",
        "SecretKey": "minioPassword",
        "Bucket": "qp8"
    }
}
```

Таблица 29. Настройки конфигурационного файла ActionsService

Название	Тип данных	Описание
Connection	Объект	Задает настройки соединения
DpcConnectionString	Строка	Строка подключения к БД каталога (используется при QpMode = false)
TasksConnectionString	Строка	Строка подключения к БД задач (используется при QpMode = false)

DesignConnectionString	Строк	Строка подключения, использующаяся при разработке структуры БД (используется при QPMode = false)
LiveMonitoringConnectionString	Строка	Строка подключения к Live-референсной витрине (используется при QpMode = false, UseQpMonitoring = false)
StageMonitoringConnectionString	Строка	Строка подключения к Stage-референсной витрине (используется при QpMode = false, UseQpMonitoring = false)
QpMode	Булевый	Режим использования QP (если true, строки подключения берутся из конфигурационного файла QP или сервиса конфигурации для БД, в которых включен режим DPC)
UseQPMonitoring	Булевый	Режим, используемый при QPMode = false, в котором данные референсной витрины хранятся в самой базе каталога (таблица Products), а не в отдельной базе. Если true, то нужно задавать LiveMonitoringConnectionString и StageMonitoringConnectionString
UsePostgres	Булевый	Подключение к PostgreSQL либо SQL Server (используется при QpMode = false)
TransactionTimeout	Строка	Таймаут транзакции
Properties	Объект	Общие свойства
Name	Строка	Имя приложения (для логов)
EnableScheduleProcess	Булевый	Включить выполнение задач по расписанию
Logging	Объект	Раздел настроек логирования
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
System	Строка	Минимальный уровень логирования для модулей System
Loader	Объект	Настройки компонента загрузки продуктов
UseFileSizeService	Булевый	Использовать ли сервис для получения размера файлов (QP.Storage) или пытаться вычислить размер файла из локальной файловой системы.
Integration	Объект	Настройки интеграции с другими сервисами

RestNotificationUrl	Строка	REST-адрес сервиса DPC.NotificationSender
S3	Объект	Настройки доступа к MinIO
Endpoint	Строка	Адрес MinIO
AccessKey	Строка	Ключ доступа к MinIO (логин)
SecretKey	Строка	Секрет доступа к MinIO (пароль)
Bucket	Строка	Имя бакета

Настройка доступа к MinIO

Настройка глобального доступа к использованию MinIO – см. Руководство администратора QP8.

Настройка доступа на уровне приложения `DPC.ActionsService` выполняется в файле `appsettings.json`, в отдельной секции `S3` (содержимое аналогично настройкам в QP). Также необходимо переключить `Loader` из режима вызова storage-сервиса в режим использования файловой системы / MinIO для определения размера файлов (описание параметров – в таблице выше):

```
"S3": {
   "Endpoint": "minio:9000",
   "AccessKey": "minioLogin",
   "SecretKey": "minioPassword",
   "Bucket": "qp8"
},
"Loader": {
   "UseFileSizeService": false
}
```

Следующая функциональность DPC поддерживает MinIO:

- определение размера файла при публикации.

7.3. Сервис отправки продуктов на витрины (DPC.NotificationSender)

Сервис NotificationSender – это windows-служба, которая автоматически формирует запросы к Системам-Витринам. Служба позволяет осуществлять автоматическое формирование уведомлений о выполнении операций создания, изменения, удаления продуктов, содержащихся в Системе «ПК».

Сервис взаимодействует со следующими БД:

- dpc_notifications – используется для работы с очередью публикаций;
- dpc_catalog – используется для работы с продуктами.

Сервис взаимодействует со следующими сервисами:

- ActionsService – сервис выполнения отложенных задач.

Управление параметрами службы осуществляется с использованием штатных возможностей бэкэнда QP на странице «Настройки». Вызов настроек находится в контекстном меню кода клиента (рис. 7.60).

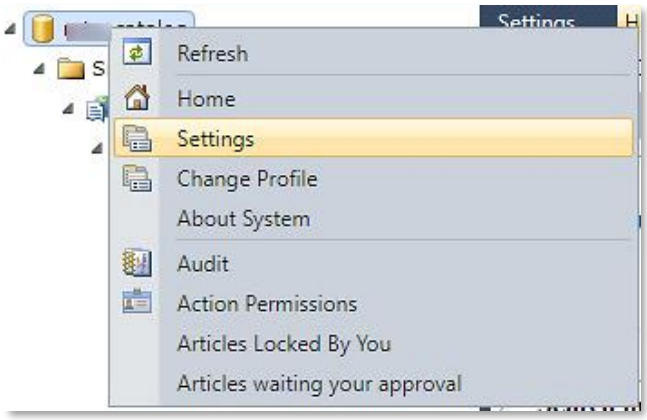


Рисунок 7.60. Настройки бэкэнда

Список параметров указан в таблице 30.

Таблица 30. Параметры инструмента публикации продуктов

Название	Описание
NOTIFICATION_SENDER_CHANNELS_CONTENT_ID	Идентификатор контента, в котором содержатся данные о каналах публикации
NOTIFICATION_SENDER_CHECK_INTERVAL	Период времени между попытками публикации. **Примечание:** значение в секундах
NOTIFICATION_SENDER_ERROR_COUNT_BERORE_WAIT	Количество попыток публикации до паузы
NOTIFICATION_SENDER_FORMATTERS_CONTENT_ID	Идентификатор контента, в котором содержатся данные о форматах, в которых сервис может передавать данные о продуктах на Витрину
NOTIFICATION_SENDER_PACKAGE_SIZE	Количество записей в очереди публикации, обрабатываемое службой за один раз
NOTIFICATION_SENDER_TIMEOUT	Время ожидания ответа от Витрины
NOTIFICATION_SENDER_WAIT_INTERVAL_AFTER_ERRORS	Длительность паузы. **Примечание:** значение в секундах
PRODUCTS_CONTENT_ID	Идентификатор контента, содержащего корневые статьи

В штатном режиме работы службы каждая публикация (количество продуктов из очереди равно `NOTIFICATION_SENDER_PACKAGE_SIZE`) осуществляется с интервалом времени `NOTIFICATION_SENDER_CHECK_INTERVAL`. Если публикация не завершается успешно за количество попыток, равное `NOTIFICATION_SENDER_ERROR_COUNT_BERORE_WAIT`, то следующая публикация откладывается на время, равное `NOTIFICATION_SENDER_WAIT_INTERVAL_AFTER_ERRORS`. После этого попытки публикации начинаются повторно. Такой подход гарантирует, что продукт будет опубликован.

Примечание: управление службой для предыдущей версии инструмента осуществлялось с использованием конфигурационного файла службы. Действующая версия в целях совместимости

также может получать собственные параметры и данные о Каналах из конфигурационного файла. Для использования параметров исключительно из конфигурационного файла в нём в качестве источника данных необходимо задать значение `NotificationConfigurationProvider`. Если задано значение `NotificationContentProvider`, то в первую очередь данные берутся из бэкенда, при их отсутствии в бэкенде – из конфигурационного файла службы.

Если в QP не заданы параметры, указанные в таблице 30, то продукты публикуются DPC Admin отправляет продукты в XML формате.

7.3.1. ГПИ службы. Функциональные возможности ГПИ

Для вызова ГПИ службы необходимо кликнуть правой кнопкой мыши по Сайту и выбрать пункт «Каналы».

На рисунке 7.61 изображен ГПИ службы NotificationSender.

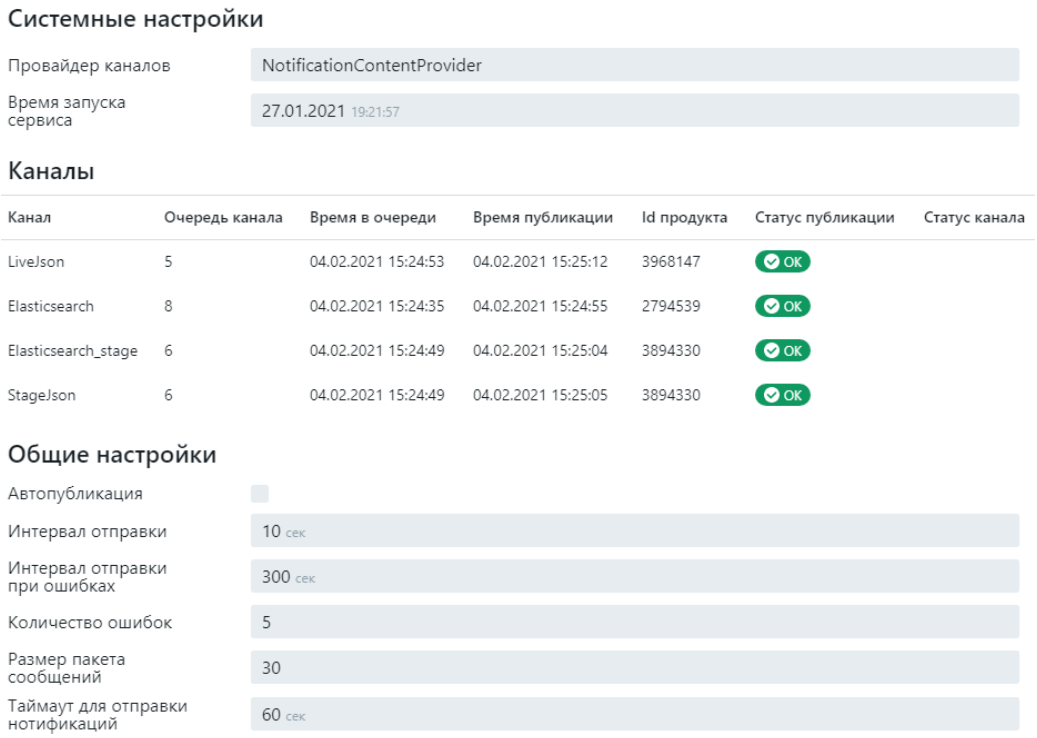


Рисунок 7.61. ГПИ службы NotificationSender

ГПИ состоит из следующих блоков:

- Системные настройки. Содержит информацию о провайдере каналов и времени запуска сервиса.
- Каналы. Блок содержит таблицу каналов, которая состоит из следующих полей:
  - Канал – название канала для Витрины. Каналы находятся в контенте `Каналы` группы `Служебные`;
  - Очередь – количество задач по публикации продуктов на Витрину, находящееся в очереди публикации. Продукты на публикацию находятся в БД сервиса;
  - Время в очереди – дата и время, когда последний продукт из очереди был помещён в очередь на публикацию.;
  - Время публикации – Дата и время, когда последний продукт из очереди был опубликован на Витрину;

  - Id продукта – идентификатор, последнего опубликованного продукта из очереди;
  - Статус публикации – статус публикации последнего продукта из очереди:
  - Статус канала – указатель, отличаются ли актуальные настройки Канала от настроек, используемых службой. Поддерживаемые состояния:
    - «Настройки совпадают»;
    - «Настройки изменены» (актуальные настройки Канала отличаются от настроек, используемых службой);
    - «Канал добавлен» (данные о Канале есть в настройках, Канал не используется службой);
    - «Канал удалён» (данных о Канале нет в настройках, Канал используется службой).
- Общие настройки. Блок содержит информацию об общих настройках:
  - Интервал отправки;
  - Интервал отправки при ошибках;
  - Количество ошибок;
  - Размер пакета сообщений;
  - Таймаут отправки на витрину.

Статистика по каналам публикации хранится в памяти службы. При этом периодически опрашивается БД. Если обнаружены изменения, то в интерфейсе отображается информационный блок, о том, что произошли изменения и псевдоссылка «Обновить», при нажатии на которую эти изменения применяются (Рисунок 7.62).

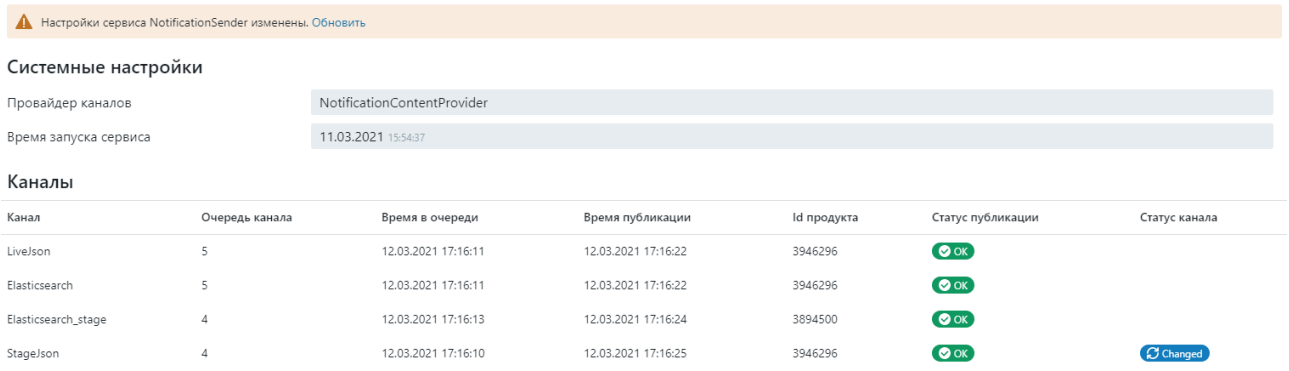


Рисунок 7.62. Обнаружены изменения в настройках

Примечание: информация о настройках, выводимая в блоке «Общие настройки» отражает значения, установленные в разделе Настройки (Settings) для кастомер кода (см. Сервис отправки продуктов на витрины NotificationSender).

7.3.2. Управление Каналами

Управление Каналами осуществляется с использованием штатных средств бэкенда – используется контент «Каналы» (группа контентов «Служебные»). Каждая статья контента содержит данные об одном Канале. Для Канала можно задать формат данных, в котором требуется передавать данные о продукте на Витрину.

Поддерживаемые форматы:

- XML;

- XAML;
- JSON.

Сервис поддерживает многопоточную передачу данных на Витрину. Также сервис позволяет ограничить набор передаваемых данных с помощью фильтрации.

Описание полей контента приведено в подразделе «Каналы».

Фильтрация данных публикации

Правило фильтрации данных (поле `Format`) не зависит от заданного для Канала формата данных, т.к. применяется к данным до их форматирования. Фильтрация проводится на стороне DPC Admin.

Примечание: для получения исходных данных (названия и значения параметров) для задания правил удобнее всего использовать представление данных о продукте в формате XAML.

Допускается использование множества правил. Значение поля требуется задавать с использованием синтаксиса EBNF. Примеры правил:

```
[Type='305']
[SortOrder='300'][!SortOrder='200']
MarketingProduct[ProductType='289']
MarketingProduct[ProductType/ProductFilters/SortOrder='1']
MarketingProduct[ProductType='289']/ProductType/ProductFilters[SortOrder='1']
Parameters[Modifiers/Alias='ExcludeFromPdf']
```

Работа Витрины в stage- и production-окружениях

В Системе DPC существует возможность работать с данными в stage- и production-окружениях. В production-окружении (live) используются исключительно данные о продуктах, статьи для которых имеют статус `Published`. В stage-окружении также доступны данные о продуктах, статьи для которых имеют статусы, отличные от `Published`.

Для Витрины допускается возможность работы с обеими окружениями. В этом случае для каждого окружения требуется предоставить отдельный URL Канала.

7.3.3. Настройка канала публикации

Сервис NotificationSender позволяет публиковать продукты на витрины, которые работают с разными форматами данных. В контенте `Форматеры` содержится список доступных форматов данных (рис. 7.63).

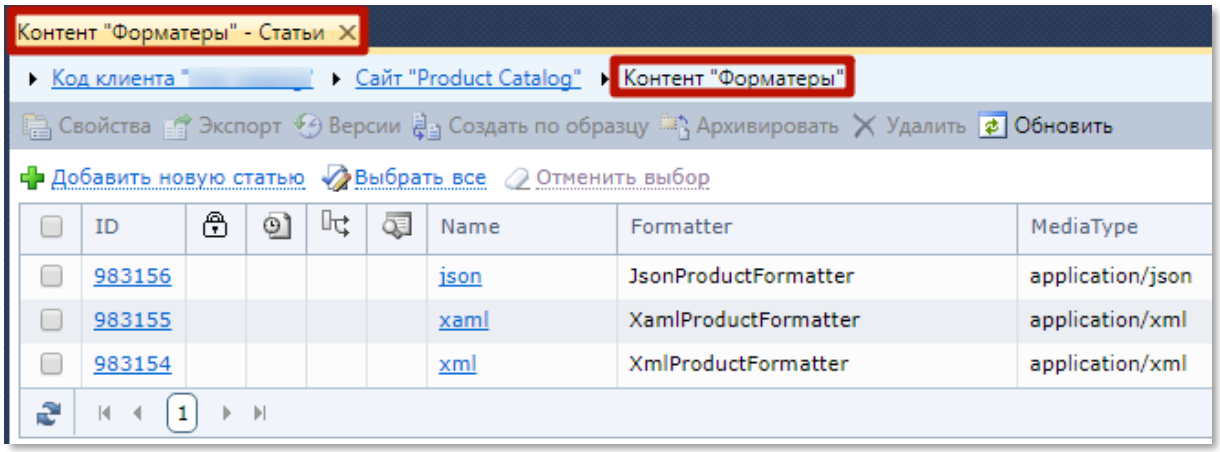


Рисунок 7.63. Контент «Форматеры»

Новый форматы регистрируются в контенте `Форматеры`. Например, если поддержка JSON еще не зарегистрирована, то ее можно зарегистрировать как форматер – `JsonProductFormatter`.

В контенте `Каналы` содержится список каналов публикации. В контенте задается соответствие, на какие адреса витрин в каком формате отправляются продукты.

Необходимо зарегистрировать минимум 4 JSON-витрины или убедиться, что они существуют. JSON-витрины, которые необходимо зарегистрировать:

- Sync API (live);
- Sync API (stage);
- Референсная витрина (live);
- Референсная витрина (stage).

Если необходима поддержка разных языков, количество витрин увеличится соответствующим образом.

В контекстном меню сайта содержится пользовательское действие «Каналы». Вызов действия позволяет получить список актуальных каналов доступных сервису публикации NotificationSender. При добавлении, редактировании каналов, требуется их публикация. Публикация позволяет применить изменения к сервису NotificationSender. Интерфейс пользовательского действия позволяет увидеть неопубликованные изменения.

Примечание: для обновления данных о публикациях продуктов также возможен вариант перезагрузки сервиса `NotificationSender`.

При изменении настроек каналов публикации вызов пользовательского действия «Каналы» выведет сообщение о том, что настройки сервиса публикации изменены. Для применения новых конфигураций необходимо кликнуть по псевдоссылке `обновить`.

Добавление нового канала

Каналы регистрируется через контент «Каналы». Для этого необходимо создать статью в контенте «Каналы» и указать:

- Название канала (поле `Name`);
- Язык канала (поле `Language`). Языки регистрируются в контенте «Языки» (см. Языки);

- URL Витрины, на которую будут публиковать продукты по текущему каналу (поле `Url`). В URL можно передавать параметры. Например,
  - язык, на котором будут передаваться продукты;
  - площадка публикации: stage или live.
- Формат продукта, в котором будут отправлять продукты на Витрину (поле `Format`);
- Признак `isStage`, если необходимо отправлять данные на stage-окружение (флаг `isStage`);
- Количество параллельно публикуемых продуктов (поле `DegreeOfParallelism`). В одной группе публикуемых продуктов не могут находится продукты с одинаковым идентификатором;

  Примечание: если одновременно публикуются продукты с одинаковым идентификатором, то продукты публикуются по одному.

- Фильтр, если требуется фильтровать продукты по какому-либо признаку (поле `Filter`). Не зависит от формата продукта;
- Сохранить статью.

Примечание: служба NotificationSender может использовать каналы указанные в бэкэнде или в конфигурационном файле (см. Сервис отправки продуктов на витрины NotificationSender). Если служба получает данные из конфигурационного файла, то данные о канала считываются из секции `<dpc:NotificationSenderConfig>`.

Создание референсной витрины для существующего каталога

Референсная витрина предназначена для первоначального создания индекса. Рекомендуется использовать консолидированную витрину, которая позволяет хранить данные различных каналов, таких как:

- live/stage;
- json/xml;
- различные языки.

Каналы консолидированной витрины хранятся в одной БД, которая может быть БД каталога, так и сторонней. Экземпляр консолидированной витрины может работать с различными экземплярами каталога, разделяя их по `customer code`. В таком случае URL может выглядеть следующим образом:

```
http://host/api/products/invariant/live
http://host/api/products/invariant/stage
```

Описанное выше решение позволяет восстанавливать индекс при сбоях в ElasticSearch. Предложенное решение гарантирует актуальную версию индекса, с учетом того, что во время сбоя данные менялись. Это возможно, поскольку референсная витрина обновляется постоянно. В связи с этим сервису индексации необходимо в настройках задать адрес референсной JSON-витрины, для того чтобы задавать команды загрузки и обновления всего индекса.

Примечание: ElasticSearch имеет встроенные механизм восстановления индекса.

Для восстановления индекса при сбоях в ElasticSearch необходимо заполнить JSON-витрину. Для автоматического обновления при каждой публикации необходимо в QP8 создать пользовательское действие с названием «Отправить JSON». Действие аналогично действию «Публиковать», но с дополнительными параметрами `Channels={имя референсной JSON-витрины}`. При выборе этого действия продукты будут отправляться только на референсную витрину.

После заполнения JSON-витрины и создания пользовательского действия необходимо отфильтровать опубликованные продукты и отправить их на публикацию в референсную витрину. Это позволит заполнить витрину актуальными данными.

7.3.4. Запросы к Каналу

Создание и изменение продукта

При создании и изменении продукта Система “ПК” формирует запрос следующего формата:

```
PUT URL Канала
```

В теле запроса содержится XML-файл с данными о продукте.

Удаление продукта

При удалении продукта Система “ПК” формирует запрос следующего формата:

```
DELETE URL Канала
```

В теле запроса содержится XML-файл с идентификатором удалённого продукта. Пример содержимого тела запроса:

```
<ProductInfo>
  <Products>
    <Product>
      <Id>9345417</Id>
    </Product>
  </Products>
</ProductInfo>
```

У Канала должна быть возможность штатно обрабатывать ситуации, когда поступает уведомление об удалении продукта, данные о котором на Витрине отсутствуют. В данной ситуации от Канала ожидается ответ с кодом 200.

7.3.5. Лог работы службы

Данные по всем выполненным операциям вносятся в лог, например, продукт с указанным идентификатором отправился на витрину А, но не был отправлен на витрину Б и т.п.

Лог представляет из себя текстовый файл `NLog.config`. Также получить информацию о работе службы можно с помощью пользовательского действия Каналы.

Каналы (Публикации каналов) — это пользовательское действие, позволяющее увидеть процесс работы службы.

7.3.6. Очередь публикации

Инструмент использует очередь публикации. Данные для очереди вносятся в таблицу «Messages» БД `dpc_notifications`. Описание полей таблицы приведено в подразделе «Таблица Messages БД dpc_notifications».

Задача на публикацию находится в очереди до момента успешной передачи данных на Витрину. Продукты с одинаковым идентификатором публикуются по очереди, даже если для канала задана параллельная публикация.

Если в настройках канала установлен признак `Invisible`, то продукты не будут помещены в очередь этого канала.

7.3.7. Защита от ошибочных публикаций

Функциональность «Защита от ошибочных публикаций» позволяет предотвратить публикацию продуктов одного продуктового каталога на витрину другого. Защита реализована следующим образом:

1. В конфигурационных файлах службы `NotificationSender` и витрины продуктового каталога задается параметр `instanceId`. Если параметр не задан, то каталог работает без защиты от ошибочной публикации. Отсутствие защиты обеспечивает обратную совместимость.
2. При публикации продуктов значения сравниваются витриной. Если значения совпадают, то продукты публикуются. Если не совпадают, то публикация продуктов блокируется. В ГПИ каналов (пользовательское действие `Каналы`, см. Каналы) доступен статус публикации. Если публикация заблокирована механизмом защиты, то в статусе будет указано - «Forbidden».

Защита от ошибочных публикаций также добавлена в референсную витрину и elastic-витрину. В конфигурационных файлах референсной и elastic витрин хранятся значения `instanceId`, которые также сравниваются. Если значения совпадают, то продукты с референсной витрины будут опубликованы на elastic-витрине при переиндексации. Если значения не совпадают, то в поле «Обработка» (пользовательское действие `HighloadFront`, см. HighloadFront) выводится пиктограмма «X». Наведение курсора мыши на пиктограмму отображает сообщение об ошибке. Если ошибка вызвана механизмом защиты от ошибочной публикации, то выводится сообщение: «Не удалось пройти валидацию по InstanceId».

Примечание: если витрина является нестандартной (custom), то вышеописанная логика должна быть самостоятельно реализована разработчиком.

7.3.8. Настройки конфигурационного файла

Пример синтаксиса:

```
{
    "Connection": {
        "DpcConnectionString": "Application Name=NotificationSender;Initial Catalog=catalog;Data Source=server;User ID=user;Password=pass",
        "NotificationsConnectionString": "Application Name=NotificationSender;Initial Catalog=notifications;Data Source=server;User ID=user;Password=pass",
        "DesignConnectionString": "Application Name=NotificationSender;Initial Catalog=notifications_design;Data Source=server;User ID=user;Password=pass",
        "QpMode": true,
        "UsePostgres": false
    },
    "Properties" :
    {
```

```
        "Name" : "DPC.NotificationsSender"
    },
    "Logging": {
        "IncludeScopes": false,
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "System": "Warning"
        }
    }
}
```

Параметры:

Таблица 31. Настройки конфигурационного файла NotificationSender

Название	Тип данных	Описание
Connection	Объект	Задает параметры соединения
DpcConnectionString	Строка	Строка подключения к БД каталога (используется при QpMode = false)
NotificationsConnectionString	Строка	Строка подключения к базе уведомлений (используется при QPMode = false)
DesignConnectionString	Строка	Строка подключения, использующаяся при разработке структуры БД (используется при QPMode = false)
QpMode	Булевый	Режим использования QP (если true, строки подключения берутся из конфигурационного файла QP или сервиса конфигурации для БД, в которых включен режим DPC)
UsePostgres	Булевый	Подключение к PostgreSQL либо SQL Server (используется при QpMode = false)
Properties	Объект	Общие настройки
InstanceId	Строка	ID конкретной инсталляции каталога (задается, чтобы исключить передачу информации между различными инсталляциями)
Name	Строка	Имя приложения (для логов)
Logging	Объект	Раздел настроек логирования
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию

Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
System	Строка	Минимальный уровень логирования для модулей System

7.4. DPC Web API (DPC.Api)

Сервис Web API позволяет создавать, редактировать, обновлять и удалять продукты DPC с помощью методов.

У каждого неинтерфейсного действия существует метод Web API. Вызов метода равносилен вызову пользовательского действия.

7.4.1. Описание API

Получение списка продуктов

Описание: метод предназначен для получения списка продуктов.

Формат запроса:

```
GET /api/{version}/{slug}/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
startRow	integer	Указывает с какого продукта возвращать продукты Значение по умолчанию – 0
pageSize	integer	Количество выводимых продуктов, если запрос задан без указания идентификатора Пример запроса: `http://host:post/api/customer_code/V1/Services/json/?pageSize=100`

Ответ:

получаем ID продуктов

Получение списка продуктов (требуется customerCode)

Описание: метод предназначен для получения списка продуктов.

Формат запроса

```
GET /api/{customerCode}/{version}/{slug}/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
startRow	integer	Указывает с какого продукта возвращать продукты Значение по умолчанию – 0
pageSize	integer	Количество выводимых продуктов, если запрос задан без указания идентификатора Пример запроса: `http://host:post/api/customer_code/V1/Services/json/?pageSize=100`
customerCode	string	Код пользователя Required

Ответ:

получаем ID продуктов

Поиск продуктов с фильтром

Описание: метод предназначен для получения информации о списке продуктов с использованием запроса (позволяет искать продукт с заданным в поле «query» значением).

Формат запроса:

```
GET /api/{version}/{slug}/search/{format}/{query}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса

		Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

query	string	Запрос
		
		Required
		
		Заполняется по принципу:
		
		`Field=value;`
		`Field_ChildField=value.`
		
		Field и ChildField должны быть в описании продукта, которое соответствует указанному slug
		
		Пример запроса:
		
		`MarketingProduct_Alias=mts_b_smart_122018`
		
		Можно искать по иерархии полей, в этом случае разделитель иерархии это "_"

Ответ:

получаем ID продуктов

Поиск продуктов с фильтром (требуется costumerCode)

Описание: метод предназначен для получения информации о списке продуктов с использованием запроса (требуется custumerCode, позволяет искать продукт с заданным в поле «query» значением)

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/search/{format}/{query}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required

isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

query	string	Запрос
		
		Required
		
		Заполняется по принципу:
		
		`Field=value;`
		`Field_ChildField=value.`
		
		Field и ChildField должны быть в описании продукта, которое соответствует указанному slug
		
		Пример запроса:
		
		`MarketingProduct_Alias=mts_b_smart_122018`
		
		Можно искать по иерархии полей, в этом случае разделитель иерархии это "_"

customerCode	string	Код пользователя Required

Ответ:

получаем ID продуктов

Детальный поиск продуктов с фильтром

Описание: метод предназначен для получения детальной информации о списке продуктов с использованием запроса (позволяет искать продукт с заданным в поле «query» значением).

Формат запроса:

```
GET /api/{version}/{slug}/search/detail/{format}/{query}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения

		По умолчанию установлено значение false

query	string	Запрос
		Required
		Заполняется по принципу:
		`Field=value;`
		`Field_ChildField=value.`
		Field и ChildField должны быть в описании продукта, которое соответствует указанному slug
		Пример запроса:
		`MarketingProduct_Alias=mts_b_smart_122018`
		Можно искать по иерархии полей, в этом случае разделитель иерархии это "_"

includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине

Ответ:

получаем ID продуктов

Детальный поиск продуктов с фильтром (требуется customeCode)

Описание: метод предназначен для получения детальной информации о списке продуктов с использованием запроса (позволяет искать продукт с заданным в поле «query» значением, требует на вход парметра customerCode).

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/search/detail/{format}/{query}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения

		• Если не установлен, то stage-окружения По умолчанию установлено значение false

query	string	Запрос
		Required
		Заполняется по принципу:
		`Field=value;`
		`Field_ChildField=value.`
		Field и ChildField должны быть в описании продукта, которое соответствует указанному slug
		Пример запроса:
		`MarketingProduct_Alias=mts_b_smart_122018`
		Можно искать по иерархии полей, в этом случае разделитель иерархии это "_"

includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине
customerCode	string	Код пользователя Required

Ответ:

получаем ID продуктов

Поиск продуктов с расширенной фильтрацией (поиск по нескольким полям)

Описание: метод предназначен для получения иинформации о списке продуктов с использованием расширенногор запроса. POST запросы позволяют делать расширенную фильтрацию по нескольким полям.

Формат запроса:

```
POST /api/{version}/{slug}/search/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary)

		Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Request body		Запрос
		
		Required
		
		Пример заполнения:
		
		`{`
		`"Regions.Alias": "orenburg",`
		`"MarketingProduct.Alias": "mts_b_smart",`
		`"not": { "GlobalCode": [ "2119.1", "2119.2" ] }`
		`}`
		
		Иерархия полей разделяется “.”, например, “MarketingProduct.Alias”
		
		Если на одном уровне появляются одинаковые названия, то они разделяются уникальным символом “@”, например “and” и “@and”, “field” и “@field” и “@@field” и т.д.
		
		Сейчас поддерживаются условия равенства полей значениям и логические условия “and”, ”or”, ”not”
		
		Если условие явно не задано, по умолчанию подразумевается “and”. Следующие условия эквивалентны:
		
		`{ "and": { "f1": "v1", "f2": "v2" } }` и `{ "f1": "v1", "f2": "v2" }`
		
		`{ "not": { "and": { "f1": "v1", "f2": "v2" } } }` и `{ "not": { "f1": "v1", "f2": "v2" } }`
		
		Для условия “or”, примененного к одному полю, возможен упрощенный синтаксис. Следующие условия эквивалентны:
		
		`{ "or": { "f": "v1", "@f": "v2" } }` и `{ "f": ["v1", "v2"] }`

Ответ:

получаем ID продуктов

Поиск продуктов с расширенной фильтрацией (поиск по нескольким полям, требуется customeCode)

Описание: метод предназначен для получения иинформации о списке продуктов с использованием расширенногор запроса. POST-запросы позволяют делать расширенную фильтрацию по нескольким полям.

Формат запроса:

```
POST /api/{customerCode}/{version}/{slug}search/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Request body		Запрос
		
		Required
		
		Пример заполнения:
		
		`{`
		`"Regions.Alias": "orenburg",`
		`"MarketingProduct.Alias": "mts_b_smart",`
		`"not": { "GlobalCode": [ "2119.1", "2119.2" ] }`
		`}`
		
		Иерархия полей разделяется “.”, например, “MarketingProduct.Alias”
		
		Если на одном уровне появляются одинаковые названия, то они разделяются уникальным символом “@”, например “and” и “@and”, “field” и “@field” и “@@field” и т.д.
		
		Сейчас поддерживаются условия равенства полей значениям и логические условия “and”, ”or”, ”not”
		
		Если условие явно не задано, по умолчанию подразумевается “and”. Следующие условия эквивалентны:
		
		`{ "and": { "f1": "v1", "f2": "v2" } }` и `{ "f1": "v1", "f2": "v2" }`
		
		`{ "not": { "and": { "f1": "v1", "f2": "v2" } } }` и `{ "not": { "f1": "v1", "f2": "v2" } }`
		
		Для условия “or”, примененного к одному полю, возможен упрощенный синтаксис. Следующие условия эквивалентны:

		`{ "or": { "f": "v1", "@f": "v2" } }` и `{ "f": ["v1", "v2"] }`

Ответ:

получаем ID продуктов

Получение детальной информации о продукте с использованием расширенного запроса

Описание: метод предназначен для получения детальной иинформации о списке продуктов с использованием расширенногор запроса. POST-запросы позволяют делать расширенную фильтрацию по нескольким полям.

Формат запроса:

```
POST /api/{version}/{slug}/search/detail/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Request body		Запрос
		
		Required
		
		Пример заполнения:
		`{`
		`"Regions.Alias": "orenburg",`
		`"MarketingProduct.Alias": "mts_b_smart",`
		`"not": { "GlobalCode": [ "2119.1", "2119.2" ] }`
		`}`
		
		Иерархия полей разделяется “.”, например, “MarketingProduct.Alias”
		
		Если на одном уровне появляются одинаковые названия, то они разделяются уникальным символом “@”, например “and” и “@and”, “field” и “@field” и “@@field” и т.д.
		
		Сейчас поддерживаются условия равенства полей значениям и логические условия “and”, ”or”, ”not”

		Если условие явно не задано, по умолчанию подразумевается “and”. Следующие условия эквивалентны:
		
		`{ "and": { "f1": "v1", "f2": "v2" } }` и `{ "f1": "v1", "f2": "v2" }`
		
		`{ "not": { "and": { "f1": "v1", "f2": "v2" } } }` и `{ "not": { "f1": "v1", "f2": "v2" } }`
		
		Для условия “or”, примененного к одному полю, возможен упрощенный синтаксис. Следующие условия эквивалентны:
		
		`{ "or": { "f": "v1", "@f": "v2" } }` и `{ "f": ["v1", "v2"] }`

includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине

Ответ:

получаем ID продуктов

Получение детальной информации о продукте с использованием расширенного запроса (требуется CustomerCode)

Описание: метод предназначен для получения детальной иинформации о списке продуктов с использованием расширенногор запроса. POST-запросы позволяют делать расширенную фильтрацию по нескольким полям (требуется customerCode).

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/search/detail/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Request body		Запрос
		
		Required
		
		Пример заполнения:
		
		`{`
		`  "Regions.Alias": "orenburg",`
		`  "MarketingProduct.Alias": "mts_b_smart",`
		`  "not": { "GlobalCode": [ "2119.1", "2119.2" ] }`
		`}`
		
		Иерархия полей разделяется “.”, например, “MarketingProduct.Alias”
		
		Если на одном уровне появляются одинаковые названия, то они разделяются уникальным символом “@”, например “and” и “@and”, “field” и “@field” и “@@field” и т.д.
		
		Сейчас поддерживаются условия равенства полей значениям и логические условия “and”, ”or”, ”not”
		
		Если условие явно не задано, по умолчанию подразумевается “and”. Следующие условия эквивалентны:
		
		`{ "and": { "f1": "v1", "f2": "v2" } }` и `{ "f1": "v1", "f2": "v2" }`
		
		`{ "not": { "and": { "f1": "v1", "f2": "v2" } } }` и `{ "not": { "f1": "v1", "f2": "v2" } }`
		
		Для условия “or”, примененного к одному полю, возможен упрощенный синтаксис. Следующие условия эквивалентны:
		
		`{ "or": { "f": "v1", "@f": "v2" } }` и `{ "f": ["v1", "v2"] }`

includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине
customerCode	string	Код пользователя Required

Ответ:

получаем ID продуктов

Получение продукта по ID

Описание: метод позволяет получить продукт с указанным идентификатором.

Формат запроса:

```
GET /api/{version}/{slug}/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
id	integer	Идентификатор продукта Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине

Ответ:

Получаем продукт в соответсвии с описанием продукта (product definition), которое определяет slug.

Отправка изменений продукта по ID

Описание: метод позволяет отправить отредактированный продукт в DPC.

Формат запроса:

```
POST /api/{version}/{slug}/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
id	integer	Идентификатор продукта, к которому необходимо применить правки

		Required
isLive	boolean	Если флаг установлен, то изменения применяются к продукту на live-окружении По умолчанию установлено значение false
customerCode	string	Код пользователя
createVersions	boolean	Сохранять ли версию статьи во время обновления? В QP поддерживается версионность статей. При редактировании статьи через бэкенд, сохраняются старые версии и можно вернуться к предыдущей версии. При редактировании через API, можно выбирать: сохранять версию статьи или нет.
format	string	Формат (json, xml or binary) Required

Получение продукта по id (требуется customeCode)

Описание: метод получения данных о продукте по ID (требуется customerCode). Поддерживает различные форматы: json, xml, binary

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
id	integer	Идентификатор продукта Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта

includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине
customerCode	string	Код пользователя Required

Ответ:

Получаем продукт в соответсвии с описанием продукта (product definition), которое определяет slug.

Отправка изменений продукта по ID (требуется customerCode)

Описание: метод позволяет отправить отредактированный продукт в DPC.

Формат запроса:

```
POST /api/{customerCode}/{version}/{slug}/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
id	integer	Идентификатор продукта, к которому необходимо применить правки Required
isLive	boolean	Если флаг установлен, то изменения применяются к продукту на live-окружении По умолчанию установлено значение false
createVersions	boolean	Сохранять ли версию статьи во время обновления? В QP поддерживается версионность статей. При редактировании статьи через бэкенд сохраняются старые версии и можно вернуться к предыдущей версии. При редактировании через API можно выбирать: сохранять версию статьи или нет.
format	string	Формат (json, xml or binary) Required
customerCode	string	Код пользователя Required

Получение продуктов по списку ID

Описание: метод получения данных о продукте по списку ID.

Формат запроса:

```
GET /api/{version}/{slug}/list/{format}/{ids}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
ids	Array integer	Список идентификаторов продуктов Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине

Ответ:

Получаем продукты в соответсвии с описанием продукта (product definition), которое определяет slug.

Получение продуктов по списку ID (требуется customerCode)

Описание: метод получения данных о продукте по списку ID.

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/list/{format}/{ids}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
format	string	Формат (json, xml or binary) Required
ids	Array integer	Список идентификаторов продуктов

		Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
includeRelevanceInfo	boolean	Если параметр имеет значение `true`, то проверяется актуальность продукта на витрине
customerCode	string	Код пользователя Required

Ответ:

Получаем продукты в соответсвии с описанием продукта (product definition), которое определяет slug.

Получение данных является ли продукт актуальным

Описание: метод получения данных является ли продукт актуальным (продукт из базы данных совпадает с продуктом на референсной витрине).

Формат запроса:

```
GET /api/relevance/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
format	string	Формат (json, xml or binary) Required
id	integer	Идентификатор продукта Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Получение данных является ли продукт актуальным (требуется customerCode)

Описание: метод получения данных является ли продукт актуальным (продукт из базы данных совпадает с продуктом на референсной витрине, требуется customerCode).

Формат запроса:

```
GET /api/{customerCode}/relevance/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание

format	string	Формат (json, xml or binary) Required
id	integer	Идентификатор продукта Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
customerCode	string	Код пользователя Required

Получение данных является ли список продуктов актуальным

Описание: метод получения данных является ли список продуктов актуальным (продукт из базы данных совпадает с продуктом на референсной витрине).

Формат запроса:

```
GET /api/relevance/{format}/{ids}
```

Входные данные:

Название	Тип данных	Описание
format	string	Формат (json, xml or binary) Required
ids	Array integer	Список идентификаторов продуктов Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false

Получение данных является ли список продуктов актуальным (требуется customerCode)

Описание: метод получения данных является ли список продуктов актуальным (продукт из базы данных совпадает с продуктом на референсной витрине, требуется customerCode).

Формат запроса:

```
GET /api/{customerCode}/relevance/{format}/{ids}
```

Входные данные:

Название	Тип данных	Описание
format	string	Формат (json, xml or binary) Required

ids	Array integer	Список идентификаторов продуктов Required
isLive	boolean	• Если флаг установлен, то запрашивается продукт с live-окружения • Если не установлен, то stage-окружения По умолчанию установлено значение false
customerCode	string	Код пользователя Required

Сборка продукта

Примечание: метод находится в разработке

Описание: метод позволяет собирать продукты в tarantool. В начале продукт собирается в tarantool: делается запрос в API тарантула, чтобы продукт собрался. Далее продукт дополняется текущим методом (данными, которых не было в tarantool). Возвращаетя продукт идентичный тому, как если бы он собирался сразу из базы (Запросили метод -> метод запросил tarantool -> тарантул вернул недополненный продукт -> метод дополнил продукт -> метод вернул продукт).

Формат запроса:

```
GET /api/tarantool/{format}/{productId}
```

Сборка продукта (требуется customeCode)

Примечание: метод находится в разработке

Описание: метод позволяет собирать продукты в tarantool. В начале продукт собирается в tarantool: делается запрос в API тарантула, чтобы продукт собрался. Далее продукт дополняется текущим методом (данными, которых не было в tarantool). Возвращаетя продукт идентичный тому, как если бы он собирался сразу из базы (Запросили метод -> метод запросил tarantool -> тарантул вернул недополненный продукт -> метод дополнил продукт -> метод вернул продукт).

Формат запроса:

```
GET /api/{customerCode}/tarantool/{format}/{productId}
```

Публикация продукта на витрине

Примечание: метод находится в разработке

Описание: метод позволяет публиковать продукт на витрине.

Формат запроса:

```
PUT /api/tarantool/publish/{format}/{productId}
```

Публикация продукта на витрине (требуется customerCode)

Примечание: метод находится в разработке

Описание: метод позволяет публиковать продукт на витрине.

Формат запроса:

```
PUT /api/{customerCode}/tarantool/publish/{format}/{productId}
```

Удаление продукта по ID

Описание: метод позволяет удалить продукт с указанным идентификатором. Равносилен вызову пользовательского действия «Удалить».

Формат запроса:

```
DELETE /api/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
id	integer	Идентификатор удаляемого продукта Required
format	string	Формат продукта Required

Удаление продукта по ID (требуется customeCode)

Описание: метод позволяет удалить продукт с указанным идентификатором (требуется customerCode). Равносилен вызову пользовательского действия «Удалить».

Формат запроса:

```
DELETE /api/{customerCode}/{format}/{id}
```

Входные данные:

Название	Тип данных	Описание
id	integer	Идентификатор удаляемого продукта Required
customerCode	string	Код пользователя Required
format	string	Формат продукта Required

Вызов пользовательского действия по названию

Описание: метод позволяет вызвать зарегистрированное неинтерфейсное действие DPC по его идентификатору.

Формат запроса:

```
POST /api/custom/{format}/{name}/{id}
```

Входные данные:

Название	Тип данных	Описание
name	string	Название пользовательского действия
id	integer	Идентификатор пользовательского действия
format	string	Формат продукта Например, json или xml. Стандартный формат DPC – XAML

Вызов пользовательского действия по названию (требуется customeCode)

Описание:

Формат запроса:

```
POST /api/{customerCode}/custom/{format}/{name}/{id}
```

Входные данные:

Название	Тип данных	Описание
name	string	Название пользовательского действия
id	integer	Идентификатор пользовательского действия
format	string	Формат продукта Например, json или xml. Стандартный формат DPC – XAML
customeCode	string	Код пользователя Required

Получение схемы продукта

Описание: метод позволяет получить схему продукта. Схема продукта — это описание структуры данных продукта с указанием типов данных и значений сущностей и т.п.

Формат запроса:

```
GET /api/{version}/{slug}/schema/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
forList	boolean	
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта

format	string	Формат (json, xml or binary) Required

Получение схемы продукта (требуется customerCode)

Описание: метод позволяет получить схему продукта. Схема продукта — это описание структуры данных продукта с указанием типов данных и значений сущностей и т.п.

Формат запроса:

```
GET /api/{customerCode}/{version}/{slug}/schema/{format}
```

Входные данные:

Название	Тип данных	Описание
slug	string	Название сервиса Required
version	string	Версия описания продукта Required
forList	boolean	
includeRegionTags	boolean	Теги региональных замен (реплейсы) Если параметр задан, то будет выведен продукт с расшифровкой региональных замен для региона продукта
format	string	Формат (json, xml or binary) Required

7.4.2. Настройки конфигурационного файла

Пример синтаксиса:

```
{
  "Properties": {
    "UseAuthorization": false,
    "Name": "DPC.WebAPI"
  },

  "Connection": {
    "QpMode": true,
    "UsePostgres": false,
    "TransactionTimeout": "00:10:00"
  },

  "Logging": {
    "IncludeScopes": false,
```

```
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "System": "Warning"
    }
  },
  "Loader": {
    "UseFileSizeService": true
  },
  "Integration": {
    "RestNotificationUrl": "Notification",
    "HighloadFrontSearchUrl": "SearchApi"
  },
  "Camunda": {
    "IsEnabled": true,
    "Address": "https://camunda7.localhost.ru/engine-rest",
    "Login": "demo",
    "Password": "demo",
    "DefaultFetchCount": 10,
    "WorkerId": "DPC",
    "DefaultLockDuration": "00:01:00",
    "ExternalTaskFetchWaitTime": "00:00:10"
  }
}
```

Параметры:

Таблица 32. Настройки конфигурационного файла WebAPI

Название	Тип данных	Описание
Connection	**Объект**	**Задает параметры соединения**
DpcConnectionString	Строка	Строка подключения к БД каталога (используется при QpMode = false)
NotificationsConnectionString	Строка	Строка подключения к базе уведомлений (используется при QPMode = false)
DesignConnectionString	Строка	Строка подключения, использующаяся при разработке структуры БД (используется при QPMode = false)
QpMode	Булевый	Режим использования QP (если true, строки подключения берутся из конфигурационного файла QP или сервиса конфигурации для БД, в которых включен режим DPC)

UsePostgres	Булевый	Подключение к PostgreSQL либо SQL Server (используется при QpMode = false)
Properties	**Объект**	**Общие настройки**
InstanceId	Строка	ID конкретной инсталляции каталога (задается, чтобы исключить передачу информации между различными инсталляциями)
Name	Строка	Имя приложения (для логов)
Logging	**Объект**	**Раздел настроек логирования**
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	**Объект**	**Уровни логирования для различных модулей**
Default	Строка	Минимальный уровень логирования по умолчанию
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
System	Строка	Минимальный уровень логирования для модулей System
Integration	**Объект**	**Настройки интеграции с другими сервисами**
RestNotificationUrl	Строка	REST-адрес сервиса DPC.NotificationSender
HighloadFrontSearchUrl	Строка	REST-адрес сервиса DPC.Search
Camunda	**Объект**	**Раздел настроек интеграции с Camunda**
IsEnabled	Булевый	Активация интеграции
Address	Строка	Адрес REST-API платформы Camunda
Login	Строка	Логин в платформе Camunda для взаимодействия с ней через REST-API
Password	Строка	Пароль в платформе Camunda для взаимодействия с ней через REST-API
DefaultFetchCount	Число	Количество задач, которые будут выбраны для обработки за один раз
WorkerId	Строка	Идентификатор обработчика. **Примечание:** если DPC запускается в нескольких экземплярах, то у каждого экземпляра должен быть свой уникальный WorkerId.
DefaultLockDuration	Строка (интервал времени)	Время, на которое выбранная для выполнения задача блокируется
ExternalTaskFetchWaitTime	Строка (интервал времени)	Период частоты, с которой DPC будет проверять наличие задач для обработки

7.5. Elastic API – витрина для записи в Elastic (DPC.Sync)

Elastic API построен на основе поисковой машины Elastic. Elastic API разворачивается в двух экземплярах, каждый из экземпляров имеет свой адрес:

- DPC.Sync– позволяет публиковать продукты на витрины, доступен только из локальной сети;
- DPC.Search– позволяет вести поиск по продуктам, доступен во внешней сети (см. ниже).

7.5.1. Общие сведения

Сервис Sync API реализует интерфейс JSON витрины DPC. Это значит, в DPC можно настроить канал публикации с указанием на этот сервис.

Для создания интерфейса в DPC создается канал публикации. Канал публикации регистрируется в контенте Каналы.

Канал Sync API имеет следующий формат:

```
{host}/api/{sync}/{language}/{env}
```

где:

- `host` – адрес размещения сервиса;
- `sync` – Sync API;
- `language` – параметр, который задает язык представления продукта;
- `env` – площадка. Например, stage или live;

Комбинация параметров `language` и `env` определяют **Elastic-индекс**. Elastic-индексы регистрируются в контенте `Индексы Elastic`.

При публикации продукты попадают в сервис, который индексирует их в Elasticsearch. Запросы на публикацию выполняются с помощью методов PUT или DELETE. Запросы посылаются по указанному каналу на витрину.

7.5.2. Интерфейс индексации

Сервис позволяет запускать первичную и/или повторную индексацию данных с референсной витрины. Это необходимо в следующих случаях:

- первичное заполнение индекса;
- восстановление индекса после сбоев Elasticsearch;
- получение состояния витрины на указанную дату (см. ниже);
- переиндексация данных после изменения следующих настроек в API:
  - общие настройки индекса (`MaxResultWindow`, `TotalFieldsLimit`),
  - настройки анализаторов (`Types`, `NotAnalizedFields`),
  - настройки дополнительного индексирования (`IndexingOptions`),
  - настройки N-грам (`EdgeNgramOptions`).

Подробнее о данных действиях – см. HighloadFront (Высокопроизводительная витрина).

7.5.3. Состояние витрины на указанную дату

Получить состояние elastic-витрины на указанную дату можно, используя обычный запрос, при условии специальной настройки. Если у индекса задано значение поля `date`, то запрос возвращает состояние (`state`) витрины на указанную дату в поле `date`.

Примечание: дата, за которую выводится состояние elastic-витрины, задается в контенте `Индексы Elastic` (см. Индексы Elastic).

7.5.4. Настройки конфигурационного файла

Пример синтаксиса:

```
{
    "SonicElasticStore": {
      "DefaultSize": 10,
      "MaxResultWindow": 20000,
      "TotalFieldsLimit": 100000,
      "IdPath": "Id",
      "TypePath": "Type",
      "DefaultType": "untyped",
      "IndexingOptions": [
        {
          "Name": "ParamsDic",
          "Path": "Parameters",
          "Keys": [
            "BaseParameter.Alias",
            "BaseParameterModifiers[*].Alias"
          ]
        }
      ],
      "Types": [
        "_default_"
      ],
      "NotAnalyzedFields": [
        "GlobalCode",
        "Alias",
        "ForisID"
      ],
      "DynamicDateFormats": {
        "Default": [
          "MM/dd/yyyy",
          "MM/dd/yyyy HH:mm",
          "MM/dd/yyyy HH:mm:ss",
          "dd.MM.yyyy",
```

```
        "dd.MM.yyyy HH:mm",
        "dd.MM.yyyy HH:mm:ss",
        "yyyy-MM-dd",
        "yyyy-MM-dd'T'HH:mm",
        "yyyy-MM-dd'T'HH:mm:ss",
        "yyyy-MM-dd'T'HH:mm:ssZ",
        "yyyy-MM-dd'T'HH:mm:ss.SSS",
        "yyyy-MM-dd'T'HH:mm:ss.SSS||yyyy-MM-dd'T'HH:mm:ss.SSSZ"
      ],
      "Elastic8": [
        "MM/dd/yyyy",
        "MM/dd/yyyy HH:mm",
        "MM/dd/yyyy HH:mm:ss",
        "dd.MM.yyyy",
        "dd.MM.yyyy HH:mm",
        "dd.MM.yyyy HH:mm:ss",
        "yyyy-MM-dd",
        "yyyy-MM-dd'T'HH:mm",
        "yyyy-MM-dd'T'HH:mm:ss",
        "yyyy-MM-dd'T'HH:mm:ssX",
        "yyyy-MM-dd'T'HH:mm:ss.S||yyyy-MM-dd'T'HH:mm:ss.SS||yyyy-MM-dd'T'HH:mm:ss.SSS",
        "yyyy-MM-dd'T'HH:mm:ss.SX||yyyy-MM-dd'T'HH:mm:ss.SSX||yyyy-MM-dd'T'HH:mm:ss.SSSX"
      ]
    },
    "CreationDateField": "UpdateDate",
    "EdgeNgramOptions": {
      "MinNgram": 3,
      "MaxNgram": 20,
      "NgramFields": [
          "Title",
          "Description"
      ]
    },
    "Harvester": {
      "ChunkSize": 100
    },
    "Data": {
      "QpMode": true,
      "CanUpdate": true,
      "VersionCacheExpiration": "00:05:00.000",
      "SyncName": "DPC.Sync",
```

```
        "SearchName": "DPC.Search",
        "Version": "0.3.0",
        "BuildVersion": "0.3.0.141-4a8dddf",
        "InstanceId": "dev",
        "FixedCustomerCode": "qmobile_catalog",
        "FixedConnectionString": "Initial Catalog=qpmobile_catalog;Data Source=DBSERVER;User
ID=login;Password=password",
        "ElasticTimeout": 180
      }
    },
    "Logging": {
      "IncludeScopes": false,
      "LogLevel": {
        "Default": "Information",
        "System": "Warning",
        "Microsoft": "Warning"
      }
    },
      "Integration": {
        "ConfigurationServiceUrl": "http://config-service",
        "ConfigurationServiceToken": "jwt-token",
    },
      "ApiRestrictions": {
        "MaxExpandDepth": 3
    }
}
```

Параметры:

Таблица 33. Настройки конфигурационного файла

Название	Тип данных	Описание
SonicElasticStore	Объект	Настройки хранилища
DefaultSize	Число	Размер объекта по умолчанию
MaxResultWindow	Число	Устанавливает настройку `max_result_window` для индекса Elastic (максимально возможное выводимое в ГПИ количество продуктов на странице)
TotalFieldsLimit	Число	Максимальное число полей в продукте
IdPath	Строка	Пусть к ID документа (поле ID в продукте)
TypePath	Строка	Путь к типу документа (поле типа в продукте)
DefaultType	Строка	Тип по умолчанию (если не найден в продукте)

IndexingOptions	Массив объектов	Настройки дополнительного индексирования (массивы в обьекты) - дополнительные индексы, создаваемые в продукте для поиска по коллекциям
Name	Строка	Название создаваемого обьекта
Path	Строка	Путь к исходному массиву элементов
Keys	Массив строк	Данные из элемента массива, которые становятся ключами в новом объекте
Types	Массив строк	Типы, для которых задаются `NotAnalyzedFields`
NotAnalyzedFields	Массив строк	Не анализируемые поля, исключенные из полнотекстовой индексации (не используется разбиение на токены, словоформы). Поиск строго по значению. Применяется к `Types`.
DynamicDateFormats	Массив строк	Задает форматы дат
Default	Массив строк	Форматы дат по умолчанию
Elastic8	Массив строк	Форматы дат для Elastic8
CreationDateField	Строка	Название добавляемого поля для даты создания продукта
EdgeNgramOptions	Массив объектов	Параметры выделения элементов (N-грамм) в поисковой фразе
MinNgram	Число	Минимальный размер N-граммы
MaxNgram	Число	Максимальный размер N-граммы
NgramFields	Массив строк	Список имён полей, для которых при сохранении в систему индексации (ElasticSearch/OpenSearch) будут формироваться N-граммы
Harvester	Объект	Настройки переиндексации
ChunkSize	Число	Размер пакета при переиндексации
Data	Объект	Настройки данных
QpMode	Булевый	Использование данных контентов QP в настройках (см. ниже).
CanUpdate	Булевый	Параметр запрещает (значение `false`) или разрешает (значение `true`) приложению обновлять данные. Для Sync-версии API устанавливается в `true`, для Search-версии API в `false`.
VersionCacheExpiration	Строковое представление Timespan	Кэширование версии Elastic для определенного канала
SyncName	Строка	Имя Sync-версии API (для лог-файлов)

SearchName	Строка	Имя Search-версии API (для лог-файлов)
Version	Строка	Версия описания продукта
BuildVersion	Строка	Версия сборки
InstanceId	Строка	ID конкретной инсталляции каталога (задается, чтобы исключить передачу информации между различными инсталляциями)
FixedCustomerCode	Строка	Кастомер код
FixedConnectionString	Строка	Строка подключения к БД
ElasticTimeout	Число	Таймаут подключения к Elastic
Logging	Объект	Раздел настроек логирования
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию
System	Строка	Минимальный уровень логирования для модулей System
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft
Integration	Массив объектов	Параметры настройки при установке продукта (см. Установка на Linux)
ConfigurationServiceUrl	Строка	Параметр настройки при установке продукта (см. Установка на Linux)
ConfigurationServiceToken	Строка	
ApiRestrictions	Объект	Параметры оганичений запроса
MaxExpandDepth	Число	Ограничитель глубины рекурсии расширенного поиска (`Expand`). Значение по умолчанию – 3.

7.6. Elastic API – высокопроизводительная витрина (DPC.Search)

Elastic API построен на основе поисковой машины Elastic. Elastic API разворачивается в двух экземплярах, каждый из экземпляров имеет свой адрес:

- DPC.Sync– позволяет публиковать продукты на витрины, доступен только из локальной сети (см. выше);
- DPC.Search– позволяет вести поиск по продуктам, доступен во внешней сети.

7.6.1. Общие сведения

Для поиска продуктов в Системе используются GET-запросы с необязательным набором параметров. Результат поиска выводится в формате JSON.

Помимо GET-запросов каждый из запросов (кроме методов с короткими URL-ми) можно вызывать через POST.

7.6.2. Методы поиска

Элементы запроса и его вариации

Обобщенная форма поиска выглядит следующим образом:

```
{host}/api/{customerCode}/1.0/{language}/{state}/products/{search_method}
```

где:

Элемент запроса	Описание
Обязательные компоненты запроса	
`host`	Адрес размещения API
`api`	API
`search_method`	Элемент, определяющий метод поиска. Ниже рассмотрены следующие методы: За счёт избирательного использования опциональных компонентов запрос может принимать различные сокращённые формы: • `{host}/api/products` • `{host}/api/1.0/products` • `{host}/api/1.0` • `{host}/api/1.0/{language}/{state}/products` • `{host}/api/1.0/{language}/{state}` • `{host}/api/{customerCode}/products` • `{host}/api/{customerCode}/{language}/{state}/products` • `{host}/api/{customerCode}/1.0/products` • `{host}/api/{customerCode}/1.0` • `{host}/api/{customerCode}/1.0/{language}/{state}/products` • `{host}/api/{customerCode}/1.0/{language}/{state}` Ниже описаны возможные методы поиска (`{search_method}`): • `{host}/api/1.0/`**`{id}`** • `{host}/api/1.0/`**`{type}`** • `{host}/api/1.0/`**`search`** • `{host}/api/1.0/`**`query/{alias}`** 1) Получение продукта по идентификатору – `{id}`; 2) Поиск продуктов по типу – `{type}`; 3) Полнотекстовый поиск – `search`; 4) Пользовательские методы поиска – `query/{alias}`.
Опциональные компоненты запроса	

customerCode	Код пользователя каталога (уникальное наименование клиента/проекта). Например, qmobile_catalog. По умолчанию принимает значение параметра FixedCustomerCode (если таковой задан). **Примечание:** мультитенантность реализуется при QPMode = true, когда FixedCustomerCode и FixedConnectionString не заданы (см. Параметры Data).
1.0	Версия API
language	Язык продукта. Возможные значения задаются настройками, по умолчанию доступно значение invariant. Invariant принимает значение местной локализации. Доступные языки зарегистрированы в контенте Языки.
state	Состояние продута. Возможные значения задаются настройками, по умолчанию доступны значения: live, stage.
products	Продукты

За счёт избирательного использования опциональных компонентов запрос может принимать различные сокращённые формы:

- {host}/api/products
- {host}/api/1.0/products
- {host}/api/1.0
- {host}/api/1.0/{language}/{state}/products
- {host}/api/1.0/{language}/{state}
- {host}/api/{customerCode}/products
- {host}/api/{customerCode}/{language}/{state}/products
- {host}/api/{customerCode}/1.0/products
- {host}/api/{customerCode}/1.0
- {host}/api/{customerCode}/1.0/{language}/{state}/products
- {host}/api/{customerCode}/1.0/{language}/{state}

Ниже описаны возможные методы поиска ({search_method}):

- {host}/api/1.0/**{id}**
- {host}/api/1.0/**{type}**
- {host}/api/1.0/**search**
- {host}/api/1.0/**query/{alias}**

Получение продукта по идентификатору

Общие сведения

Поиск по идентификатору доступен в следующей форме:

```
{host}/api/1.0/{language}/{state}/products/{id}
```

где id – идентификатор продукта.

Упрощенные формы запроса см. выше.

Если запрос прошел удачно, то в ответ будет выведен JSON-объект. Если продукт не найден, то будет выведена ошибка с кодом статуса 400.

Запрос принимает параметры:

Параметр	Описание
fields	Список полей продукта (см. Ограничение списка возвращаемых полей)

Примеры запросов

- Вывести все поля (упрощенная форма live, полная форма live, полная форма stage):

```
GET http://host/api/1.0/products/106801
GET http://host/api/1.0/invariant/live/products/106801
```

- Вывести поля ID и Type:

```
GET http://host/api/1.0/products/106801?fields=Id,Type
```

Поиск продуктов по типу

Общие сведения

Поиск продуктов по типу доступен в следующей форме:

```
{host}/api/1.0/{language}/{state}/products/{type}
```

где `type` – тип продукта, например: `Tariff, Service, Action`.

Данный тип запроса можно использовать как для получения продуктов, так и контентов-справочников DPC (например, список регионов, фильтров, семейств). Статьи справочников публикуются на витрину Elasticsearch так же, как и стандартные продукты DPC (если статья в справочнике настроена для публикации). При этом `type` – фиксированное значение поля для каждого из справочников (ReadOnly).

Упрощенные формы запроса см. выше.

Если запрос прошел успешно, то выводится массив продуктов.

Запрос принимает параметры:

Параметр	Описание
fields	Список полей продукта (см. Ограничение списка возвращаемых полей)
per_page	Параметры постраничного вывода (см. Постраничный вывод)
page	
take	
skip	
sort	Параметры сортировки (см. **Примечание:** подробнее об альтернативных форматах запросов с `{or}` – см. Поиск по списку значений.
order	Сортировка)

Примечание: при индексации продуктов в Elasticsearch в индекс БД попадают только продукты. Существует отдельная сущность – **региональные теги** (`RegionTags`). Чтобы не нарушить схему данных продукта, теги в сам продукт не включаются. Они индексируются как отдельный тип продукта, содержащий теги продукта и ссылку на продукт.

Примеры запросов

- Вывести первые 10 тарифов с основными полями:

```
GET http://host/api/1.0/products/Tariff
```

- Вывести первые 2 услуги со всеми полями:

```
GET http://host/api/1.0/products/Service?fields=*&per_page=2
GET http://host/api/1.0/products/Service?fields=*&take=2
```

- Вывести список регионов из справочника DPC:

```
GET http://host/api/1.0/Region
```

- Вывести информацию о продукте:

```
GET http://host/api/1.0/products/106516
```

- Выполнить поиск региональных тегов продукта по его ID. Если у продукта нет тегов результатом запроса будет пустой массив:

```
GET http://host/api/1.0/products/RegionTags?ProductId=106516
```

Примечание: по умолчанию выводятся поля, заданные в параметре `DefaultFields` конфигурации. Для продукта типа `RegionTags` по умолчанию выводятся поля региональные теги, `Type` и поле `ProductId` (ссылка на продукт, к которому относятся теги). Поля можно переопределить параметром `fields`.

Полнотекстовый поиск

Общие сведения

Форма запроса полнотекстового поиска:

```
{host}/api/1.0/{language}/{state}/products/search
```

где `search` – поиск.

Упрощенные формы запроса см. выше.

Запрос принимает параметры, указанные в разделе Поиск продуктов по типу, а также:

Параметр	Описание
q	Полнотекстовый поиск по всем полям
Type	Дополнительная фильтрация по типу продукта. Поддерживается список типов через запятую.

Примеры запросов

- Вывести первые десять продуктов со словом *тест*:

```
GET http://host/api/1.0/products/search?q=тест
```

- Вывести идентификатор и тип для продукта для первых десяти продуктов со словом *тест*, искать среди `тарифов` и `акций`:

```
GET http://host/api/1.0/products/search?q=тест&Type=Action,Tariff&fields=Id,Type
POST http://host/api/1.0/products/search
{

"fields": [
       "Id",
       "MarketingProduct.Id"
   ],
"query":
    {
        "q": "тест",
        "Type": ["Action ","Tariff"]
    }
}
```

Пользовательские методы поиска

Общие сведения

Форма запроса (поддерживаются только GET-запросы):

```
GET {host}/api/1.0/query/{alias}?{Params}
```

где:

- `alias` – значение в контенте;
- `Params` – параметры, которые передаются в пользовательский метод. По умолчанию в качестве дополнительных можно передавать следующие параметры: `id`, `take`, `skip`. Дополнительные параметры можно настроить при настройке метода.

Настройка

Пользовательские методы поиска могут использовать ограничения по данным (`data_filters`), которые задаются в контенте Highload API Methods.

Кроме пользовательских в контенте также хранятся системные методы:

1) GetById – получения продукта по ID;
2) GetByType – получение списка продуктов по типу;
3) Search – поиск по всем продуктам.

Структура данных контента Highload API Methods приведена в таблице ниже.

Название поля	Тип данных	Описание
Title	String	Короткое название в запросе, должно быть уникальным (алиас метода)

System	Булевое	Только для системных методов, которые по умолчанию – GetById, GetByType, Search
JSON	JSON	Body POST-запроса в формате DPC SearchAPI, который задаст параметры запроса для определяемого метода.

Пример:

- Для витрины (мобильного приложения) добавить возможность прямого обращения только к мобильным публичным тарифам. Требуется иметь возможность чтения по ID, по списку, делать фильтрацию по регионам (по умолчанию выборка делается по Москве).

Для реализации примера заводим метод GetPublicMobileTariffRegion и поле заполняем поле JSON следующим образом:

```
{
  "take" : 999,
  "fields": ["Regions.Id", "Regions.Alias", "Id", "Modifiers.Alias"],
  "query": {
    "Regions.Alias": "||ralias:default(moskovskaya-obl)||",
    "Type": "MobileTariff",
    "Modifiers.Alias" : "!Private"
  }
}
```

Для задания динамических значений используются конструкции формата:

```
||{parameter_name}:type({type_name}):regex({regular_expression}):default({defau
lt_value})||
```

где:

- `{parameter_name}` – имя передаваемого параметра
- `{type_name}` – тип данных передаваемого параметра (если не задано, то строка), можно задать:
  - int,
  - decimal,
  - list_int,
  - list_decimal.
- `{regular_expression}` – регулярное выражение для проверки параметра
- `{default_value}` – значение параметра по умолчанию, которое будет присвоено, если не будет пройдена провека по типу или регулярному выражению

Примеры значений полей:

4. `"Regions.Id": "||rid:type(list_int):default(1954)||"`
5. `"Regions.Id": "||rid:type(int):regex(^[\\d]{3,4}$):default(1954)||"`

Примечание: также возможно использовать кэширование (описано в разделе Кэширование).

Примеры запросов

Для примера, описанного ранее в настройке, можно вызвать метод `GetPublicMobileTariffRegion` с указанием дополнительного параметра ID продукта:

```
GET http://host/api/1.0/query/GetPublicMobileTariffRegion?id=3706003
```

Примечание: если в метод будет передан ID непубличного мобильного тарифа или продукта с другим типом, продукт не будет получен.

Можно делать фильтрацию и по полю ralias, который задан в символах `||`:

```
GET http://host/api/1.0/query/GetPublicMobileTariffRegion?ralias=leningradskaya-obl
```

7.6.3. Параметры и режимы поиска

Ограничение списка возвращаемых полей

Общие сведения

Параметр `fields` задает список полей продукта через запятую. По умолчанию выводятся только основные поля, список которых настраивается в конфигурационном файле. Чтобы получить все поля, нужно задать `fields=*`. Также можно использовать `*` в названии поля, например, `Old*Id`.
Символ `.` используется для навигации по JSON-документу. Например, чтобы вывести псевдоним продукта, необходимо указать псевдоним в качестве имени поля: `Product.Alias`. Доступно дополнительное поле UpdateDate, в котором сохраняется дата записи продукта в БД Elasticsearch.

Примеры запросов

- Вывести поля Id, внешние идентификаторы продукта, все поля маркетингового продукта, у регионов только поля Id и Alias:

```
GET
http://host/api/1.0/products/106801?fields=Id,Old*Id,MarketingProduct,Regions.Id,Regions
.Alias
```

- Вывести всю структуру продукта (все поля):

```
GET http://host/api/qmobile_catalog/1.0/products/1937552?fields=*
```

- Вывести всю структуру продукта, но выводить только идентификаторы:

```
GET http://host/api/1.0/products/106801?fields=Id,*.Id
```

- Вывести дату сохранения/обновления продукта в Elasticsearch:

```
GET http://host/api/1.0/products/106801?fields=UpdateDate
```

- Вывести названия продуктов и регионов:

```
POST http://host/api/qmobile_catalog/1.0/products/1937552
{
    "fields": ["MarketingProduct.Title", "Regions.Title"]
}
```

Примечание: для POST-запроса доступны альтернативные варианты записи тела запроса: через массив JSON (пример выше) и через список строк (пример ниже).

- Вывести алиасы продуктов и ID регионов:

```
POST http://host/api/qmobile_catalog/1.0/products/1937552
{
    "fields": "MarketingProduct.Alias,Regions.Id"
}
```

Постраничный вывод

Общие сведения

В данном режиме задействованы следующие параметры:

- `per_page` – количество продуктов на странице при постраничном выводе (по умолчанию – 10);
- `page` – номер страницы при постраничном выводе (начинается с нуля);
- `take` – альтернативный параметр пагинации – вместо `per_page/page` – количество продуктов, которые необходимо показывать (по умолчанию 10);
- `skip` – альтернативный параметр пагинации – вместо `per_page/page` – количество продуктов, которые пропускаются (по умолчанию 0). Если `skip = n`, `take = k`, то необходимо выбрать k продуктов после n.

Примчание: Параметры `take/skip` нельзя использовать совместно с `per_page/page`: необходимо выбрать одно. При использовании `take/skip` необходимо задавать `order` (см. ниже), так как сортировка по умолчанию не задана и может отличаться для разных запросов в Elastic.

Пример запросов

Получить первые 1000 ID продуктов в Москве или Санкт-Петербурге:

```
GET http://host/api/1.0/products/Tariff?Regions.Alias=spb{or}msk&per_page=1000&fields=Id
```

ИЛИ

```
GET http://host/api/1.0/products/Tariff?Regions.Alias=spb,msk&per_page=1000&fields=Id
POST http://host/api/1.0/products/Tariff
{
  "fields": "Id",
  "per_page": 1000,
  "query": {
    "Regions.Alias": [ "spb ", "msk" ]
  }
}
```

Примечание: подробнее об альтернативных форматах запросов с `{or}` – см. Поиск по списку значений.

Сортировка

Общие сведения

Следующие параметры обеспечивают сортировку запрашиваемых данных:

- `sort` – поле, по которому проводится сортировка;
- `order` – направление сортировки:
  - `asc` – по возрастанию;

– `desc` – по убыванию (значение по умолчанию).

Пример запросов

Получить список алиасов продуктов в отсортированном по возрастанию порядке:

```
POST http://host/api/qmobile_catalog/1.0/products/Tariff
{
    "sort": "MarketingProduct.Alias",
    "order": "asc",
    "fields": "MarketingProduct.Alias"
}
```

Поиск по значению поля

Общие сведения

Форма запроса простой фильтрации: `{Field}={Value}`, где

- `Field` - название поля;
- `Value` – значение.

По умолчанию, при передаче в запросе нескольких фильтров, фильтры объединяются логической операцией `И` (`AND`). При этом можно указывать одно поле несколько раз, что может быть полезно при поиске продуктов по коллекции, в которой должны присутствовать все указанные элементы.

Все текстовые поля по умолчанию индексируются с учётом морфологии слов. Например, если пользователь ввёл запрос «оплатить интернет», а на сайте есть страница «Оплата интернета», то система определит это и выведет искомые материалы.

Возможны вариации поиска:

- **Поиск по полному совпадению**: некоторые поля можно настроить так, чтобы они активировались только при полном совпадении (обычно текстовые ID). Настраивается через параметр NotAnalyzedFields (см. разделы 7.5.3 и 7.6.8).
- **Поиск по N-граммам**: данная возможность обеспечивает реализацию функции «suggest» (варианты текста, автоматически предлагаемые системой в процессе ввода). Подробнее см. здесь.
- **Поиск с использованием *wildcards***: запросы с «подстановочными знаками» позволяют искать объекты, содержащие определенные шаблоны в их текстовых полях. Функция обеспечивет поиск, даже если точный критерий поиска неизвестен. Запрос с *wildcard* использует символы '`*`' и '`?`' для представления любого количества символов или одного символа соответственно. Подробнее см. здесь.

  Примечание: данный тип поиска отрицательно влияет на скорость обработки запроса. Для быстродействия рекомендуется использовать N-граммы.

Поддерживается специальное значение `null`, которое обозначает отсутствие поля в продукте.

Примеры запросов

- Получить тариф с псевдонимом `redtext` для региона «Москва»:

```
GET http://host/api/1.0/products/Tariff?MarketingProduct.Alias=redtext&Regions.Alias=msk
```

- Получить ID тарифов, у которых нет ни одного модификатора:

```
GET http://host/api/1.0/products/Tariff?Modifiers=null
```

Поиск по интервалу значений

Общие сведения

Форма запроса фильтрации по интервалу:

`{Field}=[{From},{To}]`, где

- `Field` – название поля;
- `From`, `To` – границы интервала, в пределах которого должно находиться исходное значение.

Такой вариант поиска возможен только для числовых полей. Также поддерживаются открытые интервалы, в которых отсутствует либо `From`, либо `To`.

Пример запросов

Вывести идентификаторы акций в диапазоне от 1684000 до 1685000:

```
GET http://host/api/1.0/products/Action?fields=Id&Id=[1684000,1685000]&per_page=1000

POST http://host/api/1.0/products/Action

{
  "fields": "Id",
  "per_page": 1000,
  "query": {
    "ID": "[1684000,1685000]"
  }
}
```

Поиск по списку значений

Общие сведения

Применимы две эквивалентные формы запроса фильтрации по списку любых значений:

- `{Field}={Value1}{or}{Value2}{or} …{ValueN}` ИЛИ
- `{Field}={Value1},{Value2}, …{ValueN}`,

где `Value1..ValueN` – список значений поля `Field`.

Примечание: альтернативная запись (без `{or}`) настраивается через параметр `ValueSeparator`).

Примеры запросов

- Получить ID продуктов в Москве для заданного списка псевдонимов:

```
GET http://host/api/1.0/products/Tariff?MarketingProduct.Alias=korporativniy-
onlayner{or}redtext&Regions.Alias=msk&fields=Id

POST http://host/api/1.0/products/Tariff

{
  "fields": "Id",
  "query": {
```

```
      "MarketingProduct.Alias": [
          "korporativniy-onlayner",
          "redtext"
      ],
      "Regions.Alias": "msk"
  }
}
```

ИЛИ

```
GET http://host/api/1.0/products/Tariff?MarketingProduct.Alias=korporativniy-onlayner,redtext&Regions.Alias=msk&fields=Id
POST http://host/api/1.0/products/Tariff
{
    "fields": "Id",
    "query": {
        "MarketingProduct.Alias": "korporativniy-onlayner,redtext",
        "Regions.Alias": "msk"
    }
}
```

- Вывести ID, маркетинговый ID и Title для вторых пяти услуг, которые предоставляются в Липецке или Тамбове:

```
POST http://host/api/1.0/products/Service
{
  "fields": [ "Id", "MarketingProduct.Id", "MarketingProduct.Title" ],
  "query": {
    "Regions.Alias": [ "lipetsk", "tambov" ],
    "page": 1,
    "per_page": 5
  }
}
```

Поиск по списку ID

Общие сведения

Форма запроса фильтрации по списку ID:

`Id={Value1},{Value2}, …{ValueN}`, где `Value1..ValueN` – список значений поля.

Примечание: существует возможность поиска по полю `ProductId` региональных тегов.

Примеры запросов

```
GET http://host/api/1.0/products/Tariff?Id=106516,106549
GET http://host/api/1.0/products/RegionTags?ProductId=106516,106549
```

Поиск по коллекциям вложенных документов

Общие сведения

Документ может содержать коллекцию вложенных документов. Фильтрация типа: `Path.Key=key1&Path.Value=val1`, не даст результата применительно к JSON-объекту:

```
"Path" : [
  {
    "Key" : "key1",
    "Value" : "val1"
  },
  ...
]
```

Фильтрация не даст результата потому, что такой поиск может найти продукты, которые не соответствуют требованию:

```
"Path" : [
  {
    "Key" : "key1",
    "Value" : "another val"
  },
  {
    "Key" : "another key",
    "Value" : "val1"
  }
]
```

Поэтому при индексации данных, JSON-описание продукта проходит препроцессинг. В описание (параметр `IndexingOptions`, см. Настройки конфигурационного файла) добавляются словари для данных, по которым проводится согласованный поиск.

```
"PathDictionary" :
{
  "key1" :
  {
    "Value" : "val1"
  },
  ...
}
```

Фильтрация продукта, прошедшего препроцессинг, выглядит следующим образом:

`PathDictionary.key1.Value=val1.`

Доступна возможность создания для словаря составного ключа из нескольких полей. Поиск по составному ключу выглядит следующим образом: `PathDictionary.key1_key2_key3.Value=val1`.

Пример запроса

Найти первые сто идентификаторов тарифов с первоначальной суммой баланса 500:

```
GET
http://host/api/1.0/products/Tariff?ParamsDic.StartBalance.NumValue=500&fields=Id&per_page=100
POST http://host/api/1.0/products/Tariff
{
  "fields": "Id",
  "per_page": 100,
  "query":
  {
      "ParamsDic.StartBalance.NumValue": "[0, 500]"
  }
}
```

Поиск с использованием логических функций и экранирования

Экранирование запросов

Общие сведения

В фильтрах любых типов можно экранировать название поля символом `@` для того, чтобы избежать конфликтов. Символ экранирования можно использовать для одного поля несколько раз, что может быть полезно для написания запросов с объединением по `OR`.

Пример запроса

```
GET http://host/api/1.0/products/Tariff?@@MarketingProduct.Alias=super
```

Отрицание запросов

Общие сведения

Для любых фильтров можно использовать инверсию условия с помощью символа `!` (символ может быть изменен в конфигурации с помощью опции `NegationMark`).

Поддерживается, значение `!null`, которое обозначает присутствие поля в продукте.

Примечание: для экранирования символа отрицания в запросе может быть использован параметр `disable_not`, принимающий имя поля с «защищаемым» символом. При этом знак `!` теряет свойство отрицания.

Примеры запросов

- Получить заголовки не архивных тарифов в Санкт-Петербурге:

```
GET
http://host/api/1.0/products/Tariff?Regions.Alias=spb&Modifiers.Alias=!Archive&fields=MarketingProduct.Title&per_page=100
POST http://host/api/1.0/products/Tariff
{
  "per_page": 100,
  "fields": [ "MarketingProduct.Title" ],
  "query": {
```

```
    "Regions.Alias": "spb",
    "Modifiers.Alias": "!Archive"
  }
}
```

- Получить тарифы в Москве, у которых задан ForisId:

```
GET http://host/api/1.0/products/Tariff?Regions.Alias=msk&ForisID=!null&fields=Id
POST http://host/api/1.0/products/Tariff
{
  "fields": "Id",
  "query": {
    "Regions.Alias": "msk",
    "ForisID": "!null"
  }
}
```

Объединение по AND

Общие сведения

В поисковом запросе доступно использование логического действия «И» (`AND`). Поля в запросе разделяются стандартным для HTTP символом `&`. Действие позволяет осуществлять поиск, удовлетворяющий **нескольким условиям одновременно**. Чтобы избежать склеивания одноименных параметров в POST-запросе при передаче JSON, используется символ экранирования `@`.

Пример запросов

Вывести ID, маркетинговый ID и Title для вторых пяти услуг, которые предоставляются одновременно в Липецке и Тамбове:

```
GET
http://host/api/1.0/products/Service?fields=Id,MarketingProduct.Id,MarketingProduct.Tit
le&Regions.Alias=lipetsk&Regions.Alias=tambov&page=1&per_page=5
POST http://host/api/1.0/products/Service
{
  "fields": [ "Id", "MarketingProduct.Id", "MarketingProduct.Title" ],
  "query": {
    "Regions.Alias": "lipetsk",
    "@Regions.Alias": "tambov",
    "page": 1,
    "per_page": 5
  }
}
```

Примечание: В POST-запросе можно задавать вложенные конструкции “*and*” / “*or*”. Условия первого уровня по дефолту считаются «И», для них не нужно указывать одинарный “*and*”. Сам запрос передается в “*query*”, выводимые поля перечисляются в “*fields*”.

Получить данные первых 5-ти тарифов по полям ID продукта, ID маркетингового продукта, у которых:

(

(регион Санк-Петербург И алиас kit-all)

ИЛИ

(регион Москва И алиас kit-1)

)

И

(

(установлен модификатор Archive у регионального)

ИЛИ

(установлен модификатор Archive у маргетингового)

)

Примечание: при использовании нескольких одинаковых операторов (and/or) на одном уровне следует добавлять необходимое кол-во символов экранирования @ перед операторами (так как с одинаковыми операторами JSON является не валидным). Последовательность увеличения кол-ва @ не обязательно соблюдать, главное, чтобы текст оператора был уникальным на одном уровне. Например, @or и @@@or.

```
{
  "take": 5,
  "skip": 0,
  "fields": [
      "Id",
      "MarketingProduct.Id"
  ],
  "query": {
      "or": {
          "and": {
              "Regions.Alias": "spb",
              "MarketingProduct.Alias": "kit-all"
          },
          "@and": {
              "Regions.Alias": "moskva",
              "MarketingProduct.Alias": "kit-1"
          }
      },
      "@or": {
          "MarketingProduct.Modifiers.Alias": "Archive",
          "Modifiers.Alias": "Archive"
      }
  }
}
```

Объединение по OR

Общие сведения

В поисковом запросе доступно использование логического действия «ИЛИ» (`OR`). Поля в запросе разделяются символом `|` (может быть изменен в конфигурации с помощью опции `DisjunctionMark`). При использовании объединения по `OR` важен порядок фильтров. Если какое-либо условие, например,

фильтрацию по региону, нужно применить ко всем частям OR-выражения, то его надо явно указать нужное количество раз. Чтобы избежать склеивания одноименных параметров, которое нарушит порядок, необходимо использовать символы экранирования `@`.

Примечание: для экранирования символа операции «ИЛИ» в запросе может быть использован параметр `disable_or`, принимающий имя поля с «защищаемым» символом. При этом знак `|` теряет значение «ИЛИ» (`OR`).

Примеры запросов

- Найти тарифы в Москве, которые либо входят в группу корпоративных, либо имеют псевдоним `connect4`:

```
GET
http://host/api/1.0/products/Tariff?Regions.Alias=msk&Groups.Alias=corp&%7CMarketingProduct.Alias=connect4&@Regions.Alias=msk
```

- Найти ID тарифов из Калмыкии, у которых присутствует модификатор `Архивный` у регионального, либо у маркетингового продукта:

```
GET
http://host/api/1.0/products/Tariff?fields=Id&Regions.Alias=elista&MarketingProduct.Modifiers.Alias=Archive&%7CModifiers.Alias=Archive&@Regions.Alias=elista
POST http://host/api/1.0/products/Tariff
{
  "fields": "Id",
  "query": {
      "Regions.Alias": "elista",
      "or": {
          "MarketingProduct.Modifiers.Alias": "Archive",
          "Modifiers.Alias": "Archive"
      }
  }
}
```

Примечание: вложенные конструкции *“and”* / *“or”* POST-запроса описаны в предыдущем разделе.

Поиск по данным внутри продукта (data_filters)

Общие сведения

Для ограничения данных по полям в методах можно использовать классические фильтры по полю `fields` (как в POST-запросах).

Для ограничения значений параметров необходимо использовать поле `data_filters`, формат которого указывается в виде JsonPath.

Пример запроса

Вывести в продуктах параметры (помимо полей, указанных в `fields`), у которых алиас базового параметра не равен `BeeNumber`:

```
{
```

```
  "fields": ["Regions.Id", "Id", "Modifiers.Alias", "Regions.Alias", "Parameters.Id",
"Parameters.BaseParameter"],
   "data_filters": {
       "Parameters": "[?(@.BaseParameter.Alias != 'BeeNumber')]"
   }
}
```

Поиск в режиме expand

Общие сведения

Для упрощения API-ответов можно использовать функцию расширения. При этом API будет возвращать части ресурса только при явном запросе. Это поможет избежать проблем при запросе как слишком малого количества информации (когда приходится делать много запросов), так и слишком большого (например, при запросе тарифа вместе с услугами, которые опубликованы отдельно).

Поиск в режиме «expand» выполняется по ID (должен быть опубликован продукт или элемент справочника с данным ID). При этом указывается запрос в формате JSONPath до элемента, содержащего ID, не включая сам ID.

Возможные варианты применения `expand`:

1) по месту (in-place) – если имя не задано, ранее найденные узлы документа заменяются на новое содержимое (с учётом параметров запроса);
2) в новый элемент (name) – создаются новые узлы документа с заданным именем, куда добавляется содержимое – см. пример ниже.

Запрос принимает параметры:

- `expand` – список параметров расширенного поиска (расширяемые поля). `Expand` может быть вложенным, но ограниченным параметром `MaxExpandDepth` (по умолчанию - 3);
- `name` – имя нового элемента;
- `path` – путь к расширяемому элементу в формате JSONPath;
- `query` – запрос (поддерживаются варианты фильтрации, описанные ранее в этом разделе);
- `Regions.OldSiteId` – идентификатор региона поиска;
- `fields` – список выводимых полей.

Пример запроса

Вывести маркетинговые услуги, статьи и баннеры по заданным параметрам:

```
POST http://host/api/1.0/products/Service
{
  "expand": [
    {
      "name": "RelatedServices",
      "path": "MarketingProduct.MarketingServicesRelations[*].RelatedMarketingService.Services",
      "query": {
        "Regions.OldSiteId": 1826
      },
```

```
      "fields": "*"
    },
    {
      "name": "MediaArticles",
      "path": "MarketingProduct.MediaArticlesRelations[*].ArticleMedia",
      "fields": [
        "Id",
        "SortOrder",
        "Title",
        "ImageUrl",
        "Link,LinkText"
      ]
    },
    {
      "name": "ProductBanners",
      "path": "MarketingProduct.ProductBannersRelations[*].ProductBanner",
      "fields": [
        "Id",
        "BannerType",
        "ImageMobile",
        "ImageBackground",
        "ColorBackground"
      ]
    }
  ]
}
```

7.6.4. Аутентификация

У каждого аутентифицируемого клиента должен быть свой токен аутентификации (см. Пользователи Highload API). При запросе к API он передается в HTTP-заголовке X-Auth-Token. Если задать неправильный токен, то будет получен ответ `Status Code: 401`. При отсутствии токена используется анонимный доступ, если он разрешен конфигурацией. Данный вариант следует использовать лишь для просмотра через браузер, а для всех приложений-клиентов необходимо заводить свой токен. Если анонимный доступ запрещен, то при попытке запроса будет получен ответ 401.

7.6.5. Лимиты запросов

В целях контроля нагрузки на сервера API введено ограничение запросов API для пользователей, в том числе анонимного. Для каждого пользователя и метода API, лимиты указываются отдельно. В лимитах указывается количество запросов, которые можно выполнить в заданный интервал времени. Для каждого пользователя с каждого IP адреса создается отдельный счетчик запросов.

Настройки лимитов находятся в контенте «Лимиты Highload API» (см. Лимиты Highload API).

В HTTP-заголовках выводится информация:

- `X-RateLimit-Limit` – лимит запросов;
- `X-RateLimit-Remaining` – количество оставшихся запросов;
- `X-RateLimit-Reset` – время в формате UnixTime, когда счетчик запросов будет сброшен.

Если лимит исчерпан раньше времени, то в ГПИ будет выводится статус `Status Code: 403`, вплоть до времени, когда счетчик сбросится.

7.6.6. Кэширование

На стороне сервера

При необходимости включить кеширование для коротких запросов в поле POST-запроса можно добавить параметр `cache_for_seconds`:

```
{
  "take" : 999,
  "fields": ["Regions.Id", "Regions.Alias", "Id", "Modifiers.Alias"],
  "cache_for_seconds": 60,
  "query": {
    "Regions.Alias": "||ralias:default(moskovskaya-obl)||",
    "Type": "MobileTariff",
    "Modifiers.Alias" : "!Private"
  }
}
```

На стороне клиента

Для поиска продукта по идентификатору выдаются следующие HTTP-заголовки:

- Cache-Control: public, max-age=600;
- Vary: fields.

Они предназначены для кэширования на стороне клиента пользователями API.

Остальные методы для кэширования на стороне клиента не предназначены и поэтому возвращают заголовки:

- Cache-Control: no-cache, max-age=0;
- Pragma: no-cache.

В целях масштабирования, API может быть развёрнуто в нескольких экземплярах. При этом можно настроить балансировщик, для кэширования данных в соответствии с заголовками API. Такое решение позволяет сформировать единый кэш.

Примечание: длинные запросы не будут попадать в кэш ARR, это ограничение расширения. Запрос может получиться длинным только если перечислить слишком много полей в параметре `fields`. Поэтому нужен компромисс. Либо выводить все поля и кэшировать данные, либо выводить только выбранные, но если полей много, то кэширования не будет.

7.6.7. Быстродействие и отказоустойчивость

Для обеспечения быстродействия и отказоустойчивости применяется комплекс мер:

- Оптимизация API, насколько это позволяет платформа ASP.NET;
- Масштабируемость и балансировка нагрузки по нескольким серверам;
- Использование производительных серверов;
- Кэширование данных;
- Введение лимитов, чтобы избежать пиковых нагрузок в короткий промежуток времени.

Комбинирование мер позволяет добавиться необходимой производительности.

7.6.8. Настройки конфигурационного файла

Настройки API задаются в файле `appsettings.json`.

Пример синтаксиса:

```
{
  "SonicElasticStore": {
    "DefaultSize": 10,
    "DefaultFields": [
      "Id",
      "Type",
      "ForisID",
      "Regions.Id",
      "Regions.Title",
      "Regions.Alias",
      "MarketingProduct.Id",
      "MarketingProduct.Alias",
      "MarketingProduct.Title",
      "Modifiers.Alias"
    ],
    "NotAnalyzedFields": [
      "GlobalCode",
      "Alias",
      "ForisID"
    ],
    "ValueSeparator": ",",
    "NegationMark": "!",
    "WildcardStarMark": "*",
    "WildcardQuestionMark": "?"
  },

  "Data": {
    "CanUpdate": false,
    "VersionCacheExpiration": "00:05:00.000",
    "SearchName": "DPC.Search"
```

```
  },

  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Information",
      "System": "Warning",
      "Microsoft": "Warning"
    }
  },

  "Integration": {
    "ConfigurationServiceUrl": "address",
    "ConfigurationServiceToken": "token"
    }
}
```

Параметры:

Таблица 34. Настройки конфигурационного файла

Название	Тип данных	Описание
SonicElasticStore	Объект	Настройки хранилища
DefaultSize	Число	Размер страницы данных (количество продуктов на странице) по умолчанию
DefaultFields	Массив строк	Поля по умолчанию (при получении списка документа)
Name	Строка	Название создаваемого обьекта
Path	Строка	Путь к исходному массиву элементов
Keys	Массив строк	Данные из элемента массива, которые становятся ключами в новом объекте
Types	Массив строк	Типы, для которых задаются `NotAnalyzedFields`
NotAnalyzedFields	Массив строк	Не анализируемые поля, исключенные из полнотекстовой индексации (не используется разбиение на токены, словоформы). Поиск строго по значению. Применяется к `Types`.
ValueSeparator	Строка	Разделитель значений поля в поиске по списку
NegationMark	Строка	Символ отрицания (отметка перед значением поля, обозначает необходимость инвертировать операцию)
WildcardStarMark	Строка	Символ от 0 до n количества символов для *wildcards* **Примечание:** для экранирования символа `*` в запросе может быть использован параметр `disable_like`,

		принимающий имя поля с «защищаемым» символом. При этом знак `*` теряет свойство *wildcard*.
WildcardQuestionMark	Строка	Символ 0 или 1 количества символов для *wildcards* **Примечание:** для экранирования символа `?` в запросе может быть использован параметр `disable_like`, принимающий имя поля с «защищаемым» символом. При этом знак `?` теряет свойство *wildcard*.
Data	Объект	Настройки данных. Параметры `Data` подробнее описаны ниже.
QpMode	Булевый	Использование данных контентов QP в настройках (см. ниже).
FixedCustomerCode	Строка	Кастомер код
FixedConnectionString	Строка	Строка подключения к БД. Задаёт явное соединение к базе, хранящей настройки.
CanUpdate	Булевый	Параметр запрещает (значение `false`) или разрешает (значение `true`) приложению обновлять данные. Для Sync-версии API устанавливается в `true`, для Search-версии API в `false`.
VersionCacheExpiration	Строковое представление Timespan	Кэширование версии Elastic для определенного канала
Integration	Объект	Параметры настройки при установке продукта (подробнее – см. Установка на Linux)
ConfigurationServiceUrl	Строка	
ConfigurationServiceToken	Строка	

Параметры Data

QpMode

Если параметр имеет значение `true`, то настройки индексов `Elastic`, токенов авторизации и лимитов устанавливаются из контентов QP. При этом ID контентов задаются в `Application Settings` для данного `customer code`.

Нужно учесть:

1. Для задания индексов используется контент из переменной `ELASTIC_INDEXES_CONTENT_ID` (по умолчанию – Индексы `Elastic`);
2. Для настроек токенов авторизации используется контент из переменной `HIGHLOAD_API_USERS_CONTENT_ID` (по умолчанию – Пользователи Highload API);
3. Для задания методов API используется контент из переменной `HIGHLOAD_API_METHODS_CONTENT_ID` (по умолчанию – Методы Highload API);
6. Для настроек пользователей используется контент из переменной `HIGHLOAD_API_LIMITS_CONTENT_ID` (по умолчанию – Лимиты Highload API).

Если параметр имеет значение `false`, то настройки индексов `Elastic`, токенов авторизации и лимитов берутся из самого файла `appsettings.json`:

- Индексы в `Data.Elastic`;
- Токены авторизации в `Users`.

Пример конфигурации Data

```
"Data": {
  "CanUpdate": false,
  "QpMode": false,
  "FixedCustomerCode":  "customer_code",
  "Elastic": [
    {
      "Url": "http://host:port",
      "Name": "name",
      "Language": "invariant",
      "State": "live",
      "DoTrace": false,
      "IsDefault": true,
      "ReindexUrl": "http://host/invariant/live"
    },
    {
      "Url": "http://host:port",
      "Name": "name",
      "Language": "invariant",
      "State": "stage",
      "DoTrace": false,
      "ReindexUrl": "http://host/invariant/stage"
    }
  ]
},
```

Примечание: параметр `Elastic` используется при `QpMode = false`. Задает путь к серверу `Elasticsearch` и название индекса.

7.7. Референсная витрина (DPC.Front)

Публикуемые на референсную витрину продукты проходят ряд проверок, прежде чем попасть в таблицы БД QP. Существует 2 таблицы:

- `Products` – хранит данные о продуктах на витрине;
- `ProductVerions` – хранит данные о версиях продукта.

Алгоритм проверки и добавления продукта в БД изображен на рисунке ниже.

Примечание: если в конфигурационном файле витрины параметр `UseProductVersions` имеет значение `true`, то история продукта будет сохранятся в БД. По умолчанию параметр имеет значение `false`.

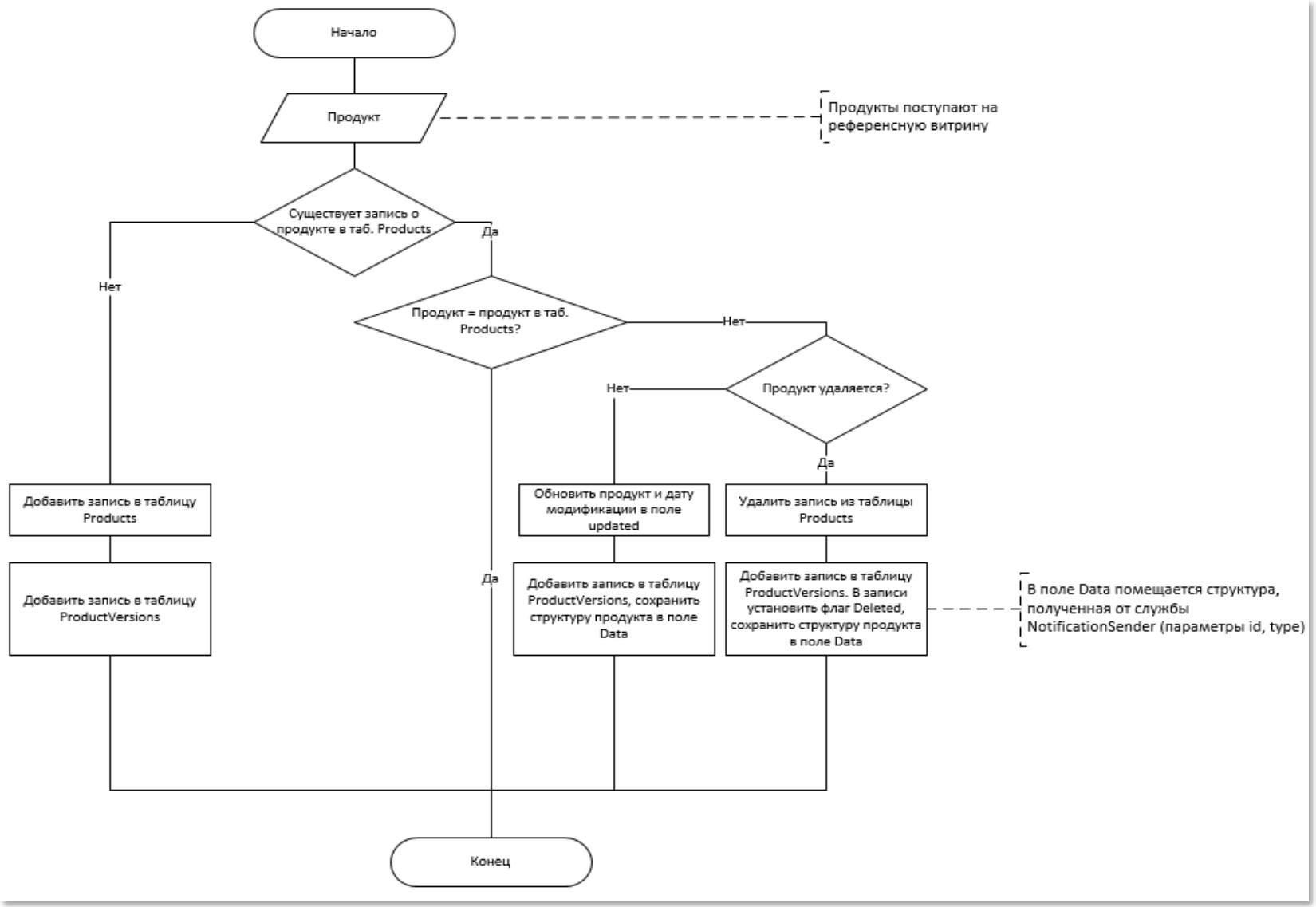


Рисунок 7.64. Алгоритм занесения данных о продукте в таблицы БД QP

7.7.1. Настройки конфигурационного файла

Пример синтаксиса:

```
{
  "Data": {
    "UseProductVersions": true,
    "UsePostgres": false,
    "QpMode": true,
    "Name": "DPC.Front"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning"
    }
  }
}
```

Параметры:

Таблица 35. Настройки конфигурационного файла DPC.Front

Параметр	Тип данных	Описание
modification	DateTime	Дата и время модификации продукта. Задается в формате: `MM/DD/YYYY hh:mm:ss AM/PM`, где: • MM – месяц; • DD– день; • YYYY – год; • hh - часы в 12-часовом формате • mm – минуты; • ss – секунды.
Data	Объект	Общие настройки
UseProductVersions	Булевый	Сохранять или нет версии продуктов
UsePostgres	Булевый	Подключение к PostgreSQL либо SQL Server (используется при QpMode = false)
QpMode	Булевый	Режим использования QP (если true, строки подключения берутся из конфигурационного файла QP или сервиса конфигурации для БД, в которых включен режим DPC)
Name	Строка	Имя приложения (для логов)
InstanceId	Строка	ID конкретной инсталляции каталога (задается, чтобы исключить передачу информации между различными инсталляциями)

FixedCustomerCode	Строка	Кастомер код (используется при QPMode = false)
FixedConnectionString	Строка	Строка подключения к БД (используется при QPMode = false)
Logging	Объект	Раздел настроек логирования
IncludeScopes	Булевый	Использовать ли области видимости для логирования (системный параметр)
LogLevel	Объект	Уровни логирования для различных модулей
Default	Строка	Минимальный уровень логирования по умолчанию
Microsoft	Строка	Минимальный уровень логирования для модулей Microsoft

8. Интеграция с Camunda

8.1. Общие сведения

Интеграция QP с платформой Camunda используется для запуска *внешнего workflow* и обеспечивает автоматизацию ряда процессов.

Внимание! Подробнее особенности интеграции с Camunda описаны в соотв. документации QP8.CMS с поддержкой PostgreSQL:

1) создание и настройка необходимых контентов в QP – см. п. 9.2 Руководства Администратора;

2) создание и настройка схемы workflow в Camunda Modeler (или аналогичном приложении) – см. п. 4.11.5 Руководства Разработчика;

3) использование внешнего workflow, созданного в рамках интеграции QP с платформой Camunda – см. п. 4.10.17 Руководства Редактора.

Настройка доступа на уровне приложения `DPC.API` выполняется в файле `appsettings.json`, в отдельной секции `Camunda` (см. п. 7.4.2). При этом нужно убедиться, что заполнены параметры секции `Integration`.

8.2. Поддерживаемые задачи

На текущий момент в рамках DPC поддерживаются только сервисные задачи (Service Tasks):

1. Отправка продукта на Stage;
2. Публикация продукта на Live;
3. Проверка публикации на высокопроизводительной витрине;
4. Проверка публикации на референсной витрине.

8.2.1. Отправка продукта на Stage

Для создания задачи отправки продукта на Stage создадим на схеме задачу типа «Service Task» и присвоим ей следующие параметры (Рисунок 8.1):

1) в секции «Implementation» выберем тип `External` и топик `SendToStage`;
2) в секции «Inputs» добавим новый параметр с именем `PublishDateVariableName` и типом `String or expression`. В значении указываем название переменной процесса, куда будет записана дата отправки.

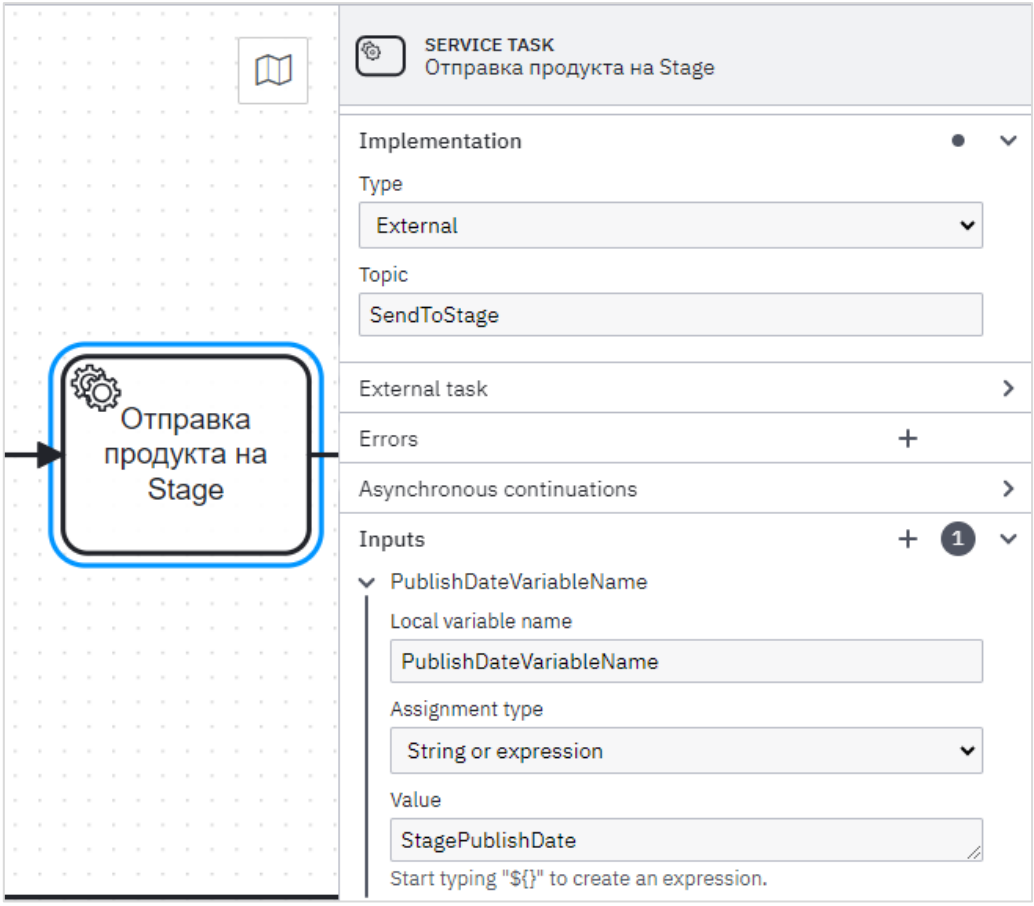


Рисунок 8.1. Создание задачи отправки продукта на Stage

Примечание: дата отправки в дальнейшем может быть использована для обновления статуса процесса и для проверки факта отправки.

8.2.2. Публикация продукта на Live

Для создания задачи публикации продукта на Live создадим на схеме задачу типа «Service Task» и присвоим ей следующие параметры (Рисунок 8.2):

1) в секции «Implementation» выберем тип `External` и топик `PublishProduct`;
2) в секции «Inputs» добавим новый параметр с именем `PublishDateVariableName` и типом `String or expression`. В значении указываем название переменной процесса, куда будет записана дата публикации.

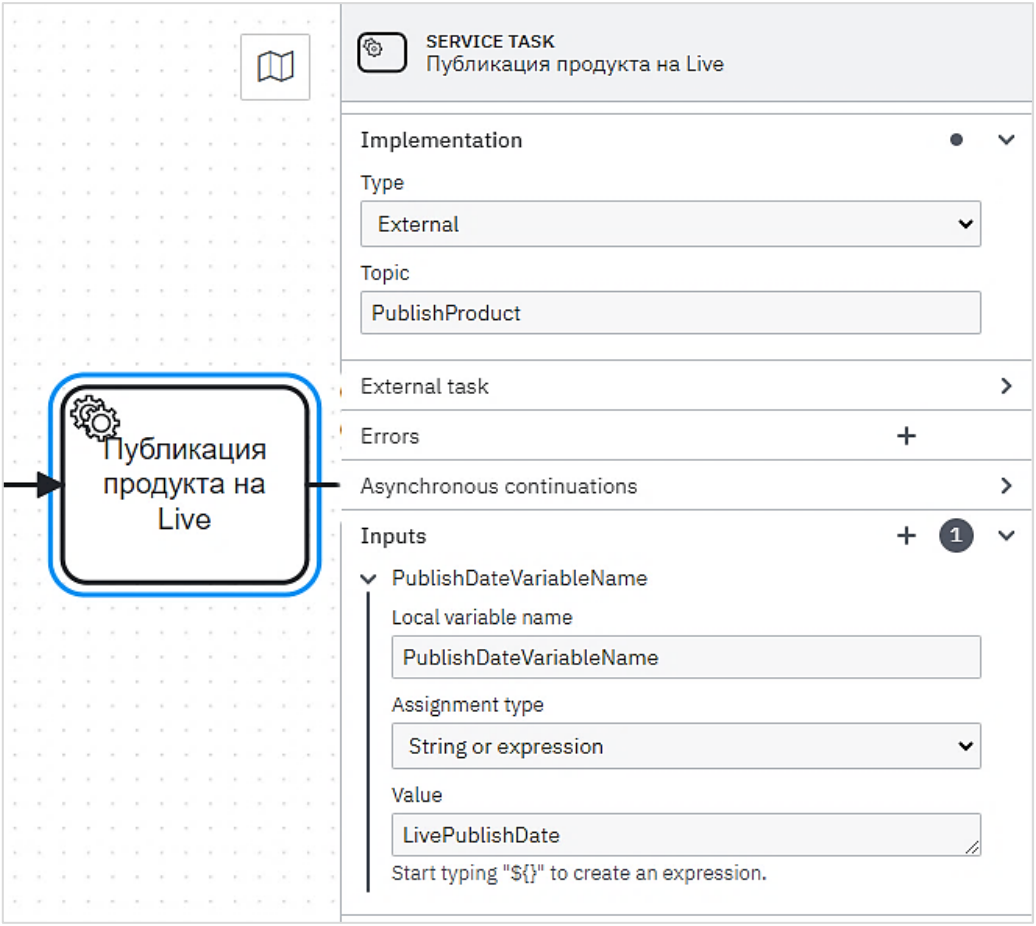


Рисунок 8.2. Создание задачи публикации продукта на Live

Примечание: дата публикации в дальнейшем может быть использована для обновления статуса процесса и для проверки факта публикации.

8.2.3. Проверка публикации на высокопроизводительной витрине

Примечание: подробнее высокопроизводительная витрина описана в п. «HighloadFront (Высокопроизводительная витрина)».

Для создания задачи проверки публикации на высокопроизводительной витрине создадим на схеме задачу типа «Service Task» и присвоим ей следующие параметры (Рисунок 8.3):

1) в секции «Implementation» выберем тип `External` и топик `CheckProductOnHighloadFront`;
2) в секции «Inputs» добавим следующие параметры (тип каждого параметра – `String or expression`):
   a. `Culture` – язык продукта, например `invariant`;
   b. `HighloadState` – состояние продукта, например `stage` или `live`;
   c. `HighloadVersion` – версия API, например `1.0`;
   d. `PublishDateVariableName` – название параметра процесса, в который была записана дата отправки или публикации на этапах отправки на Stage или публикации на Live (в зависимости о того, где требуется проверка);
   e. `ResultVariableName` – название переменной, в которую записан результат проверки (по результату проверки можно менять ветвление схемы);

f. `RetryCountVariableName` – название переменной, в которую записан номер попытки проверки (может использоваться для ограничения количества проверок перед остановкой процесса).

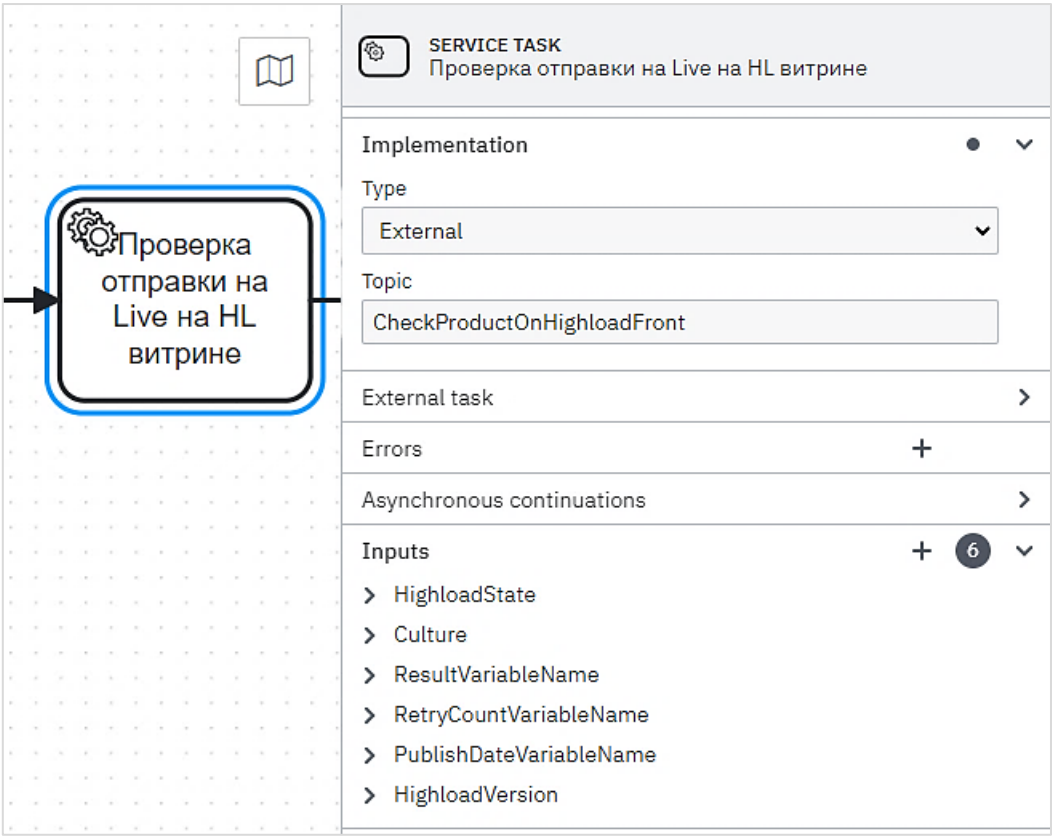


Рисунок 8.3. Создание задачи проверки публикации продукта на высокопроизводительной витрине

Пример использования возвращаемых параметров

После этапа проверки публикации можно в схеме поставить эксклюзивный шлюз (Exclusive gateway) и проверять результат.

Например, если в проверке результат был записан в параметр с именем `LiveRefResult`, то можно добавить условие `${LiveRefResult == true}` (Рисунок 8.4):

- Если условие выполняется, то токен процесса движется далее по схеме.
- Если условие не выполняется, то ставим ещё один эксклюзивный шлюз и в условии проверяем количество попыток.

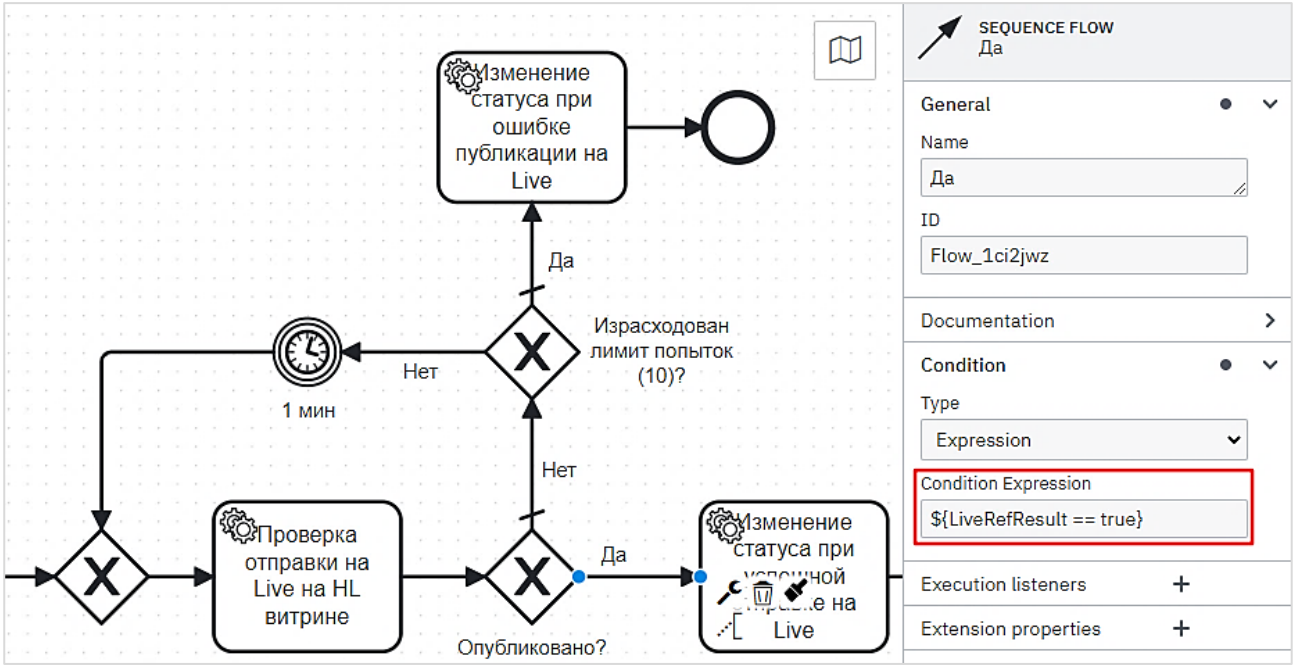


Рисунок 8.4. Проверка публикации продукта на высокопроизводительной витрине

Например, если на шаге проверки номер попытки записан в параметр `LiveRefRetryCount`, то добавляем проверку `${LiveRefRetryCount < 10}` (Рисунок 8.5):

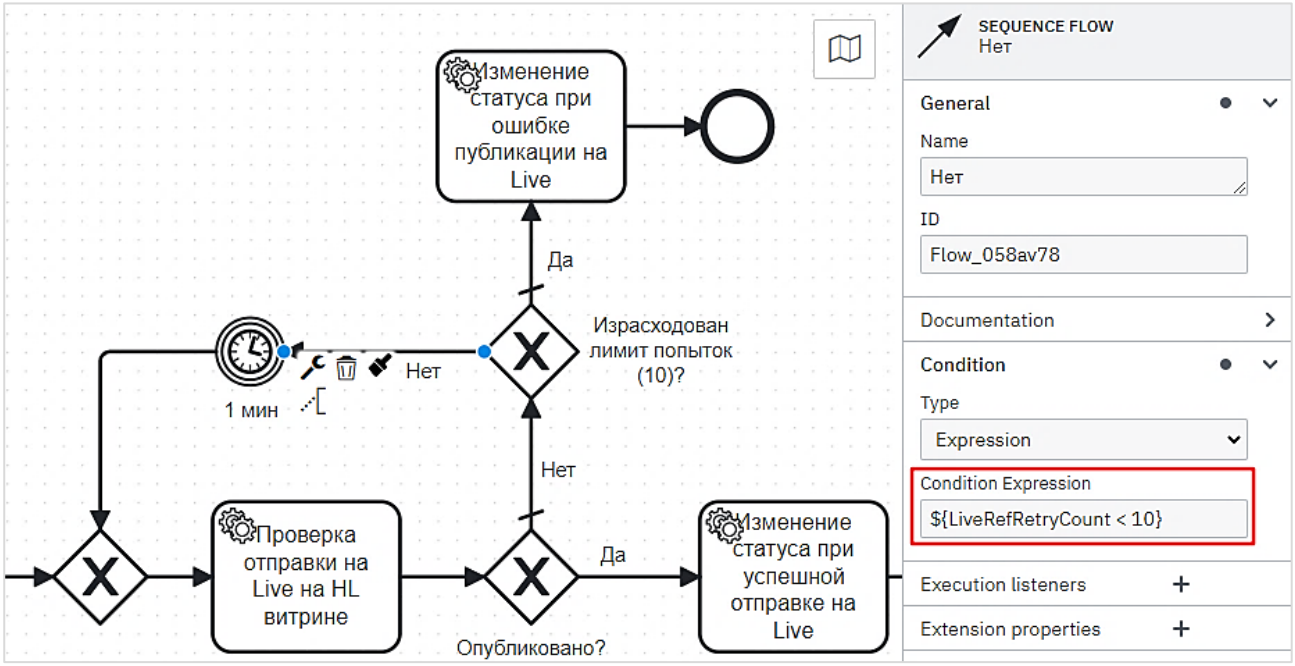


Рисунок 8.5. Проверка достижения лимита попыток

- Если условие выполняется (лимит попыток НЕ израсходован), то токен процесса попадает на таймер (1 мин., тип `Duration`), после чего процесс возвращается на этап проверки (Рисунок 8.6). Таким образом, проверка будет осуществляться до 10 раз с паузой в минуту между проверками.
- Если через 10 попыток (10 минут) публикация ещё не появилась, то можно обновить статус, разослать уведомление на почту через задачу уведомления (описано в Руководстве Разработчика QP8.CMS) и завершить процесс.

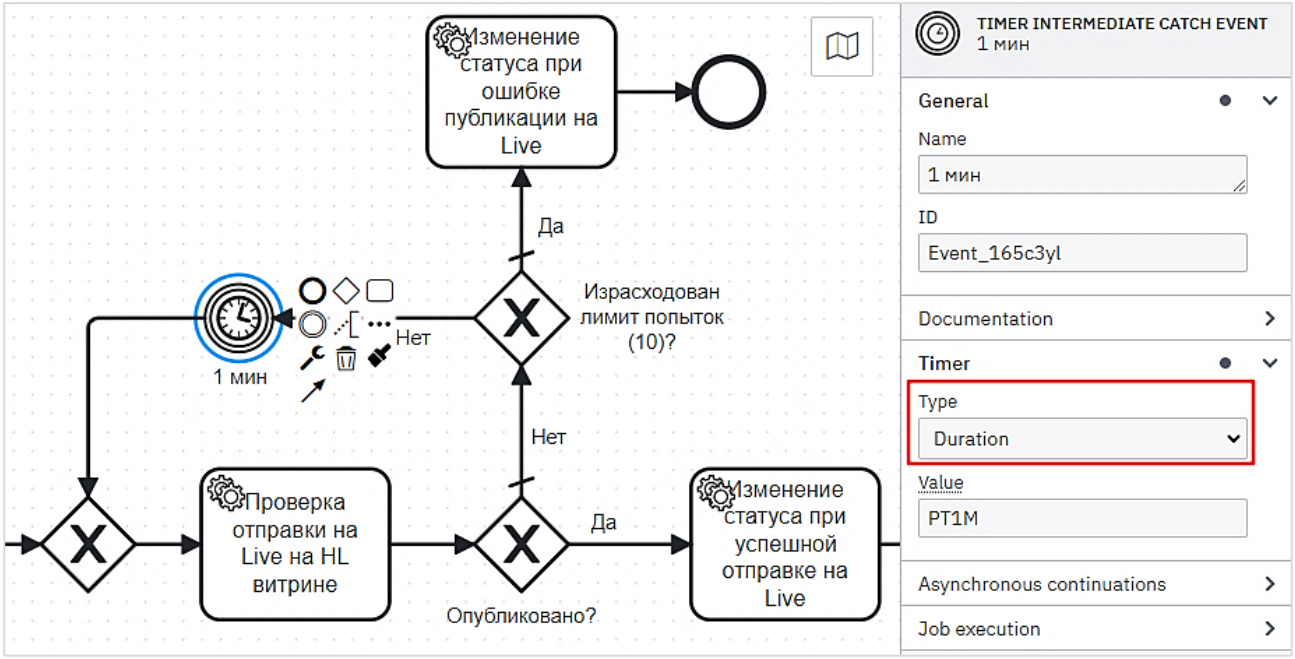


Рисунок 8.6. Использование таймера при проверке достижения лимита попыток

8.2.4. Проверка публикации на референсной витрине

Примечание: подробнее референсная витрина описана в п. «Референсная витрина (DPC.Front)».

Для создания задачи проверки публикации на референсной витрине создадим на схеме задачу типа «Service Task» и присвоим ей следующие параметры (Рисунок 8.7):

1) в секции «Implementation» выберем тип `External` и топик `CheckProductOnReferenceFront`;
2) в секции «Inputs» добавим следующие параметры (тип каждого параметра – `String or expression`):
   a. `Culture` – язык продукта, например `invariant`;
   b. `IsLive` – признак типа витрины (Live/Stage, `true` для Live);
   c. `PublishDateVariableName` – название параметра процесса, в который была записана дата отправки или публикации на этапах отправки на Stage или публикации на Live (в зависимости о того, где требуется проверка);
   d. `ResultVariableName` – название переменной, в которую записан результат проверки (по результату проверки можно менять ветвление схемы);
   e. `RetryCountVariableName` – название переменной, в которую записан номер попытки проверки (может использоваться для ограничения количества проверок перед остановкой процесса).

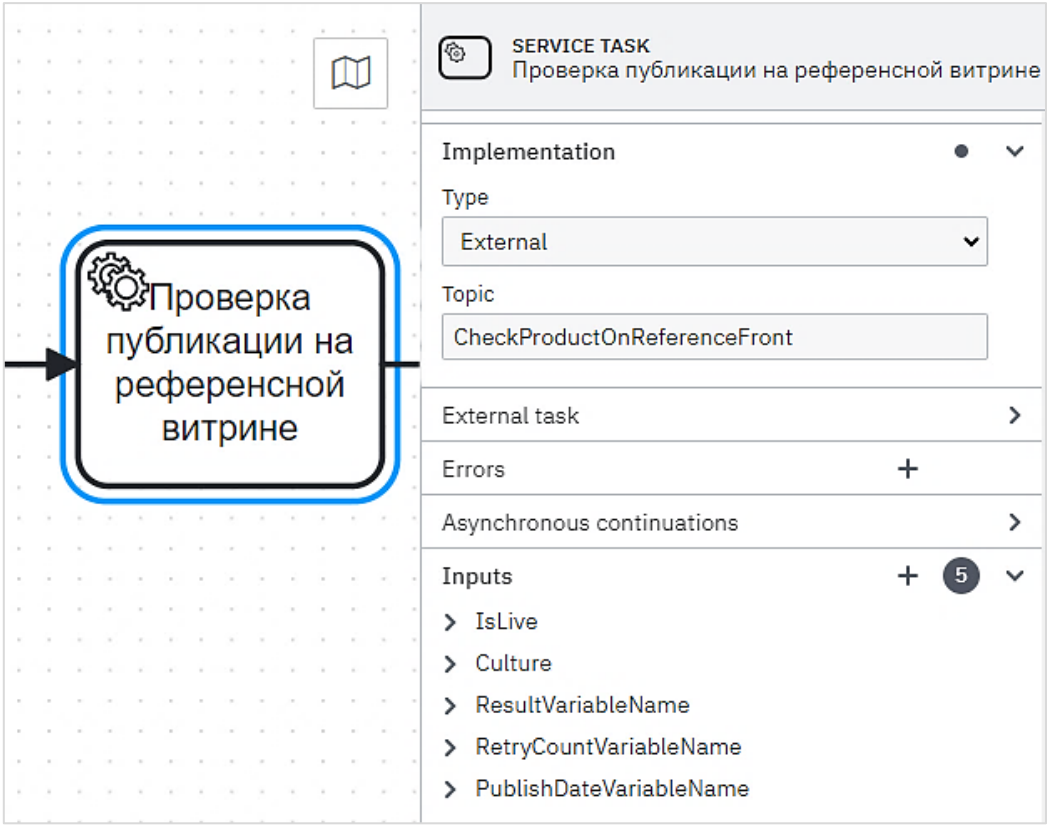


Рисунок 8.7. Создание задачи проверки публикации товара на референсной витрине

Пример использования возвращаемых параметров

После этапа проверки можно на схеме добавить эксклюзивный шлюз (Exclusive gateway) и проверять результат.

Например, если в проверке результат был записан в параметр с именем `LiveRefResult`, то можно добавить условие `${LiveRefResult == true}` (Рисунок 8.8):

- Если условие выполняется, то токен процесса движется далее по схеме.
- Если условие не выполняется, то ставим ещё один эксклюзивный шлюз и в условии проверяем количество попыток.

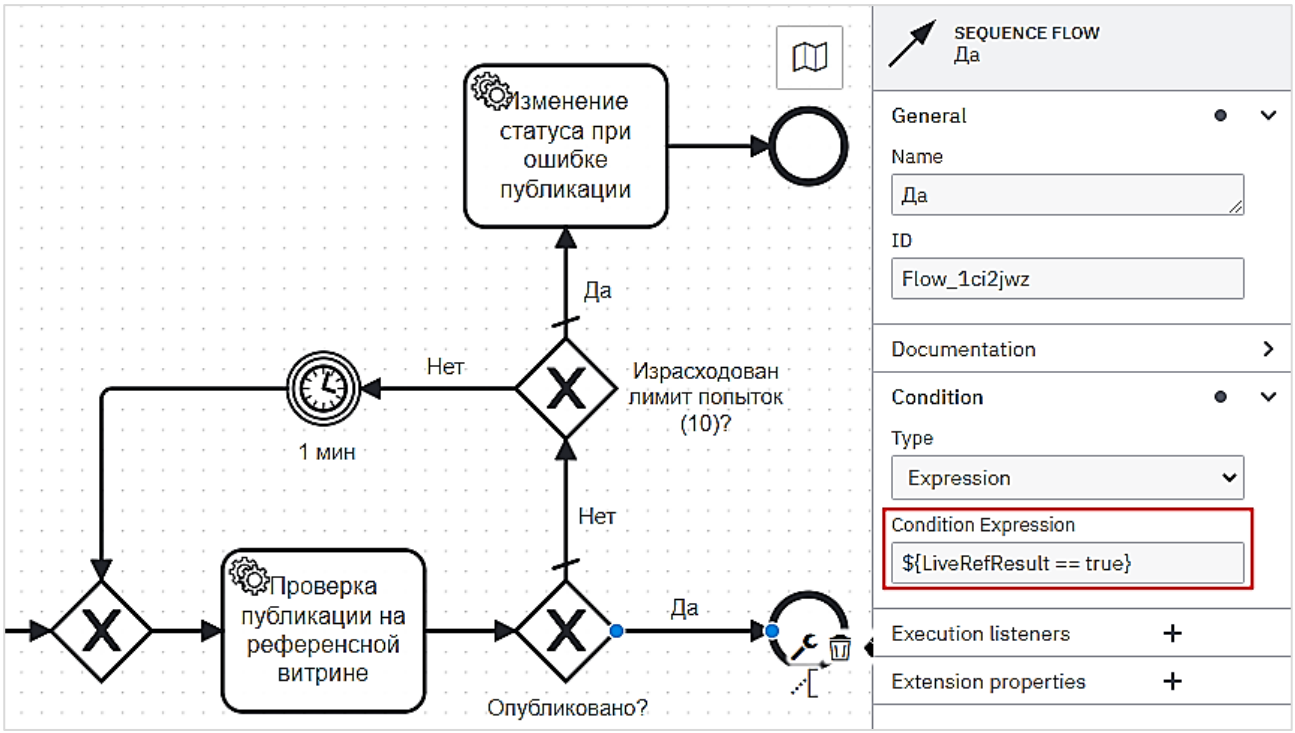


Рисунок 8.8. Проверка публикации продукта на референсной витрине

Например, если на шаге проверки номер попытки записан в параметр `LiveRefRetryCount`, то добавляем проверку `${LiveRefRetryCount < 10}` (Рисунок 8.9):

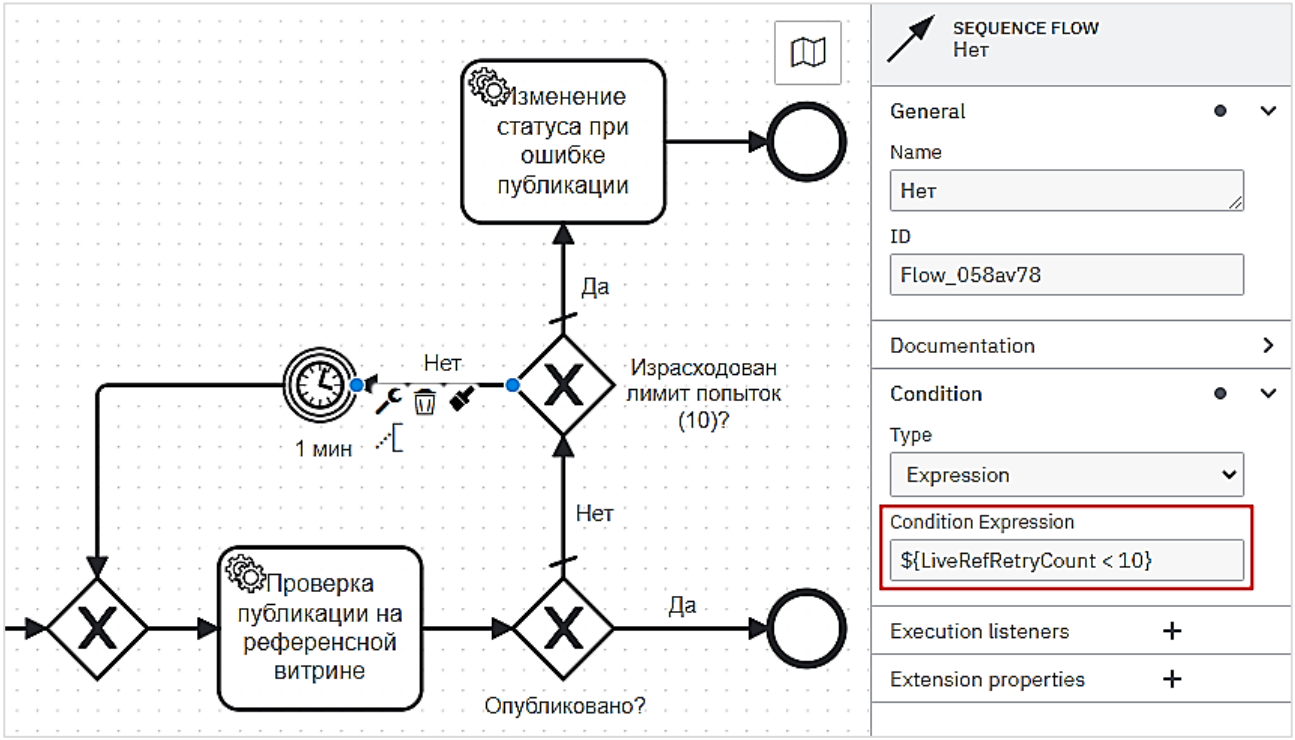


Рисунок 8.9. Проверка достижения лимита попыток

- Если условие выполняется (лимит попыток НЕ израсходован), то токен процесса попадает на таймер (1 мин., тип `Duration`), после чего процесс возвращается на этап проверки (Рисунок

8.10). Таким образом, проверка будет осуществляться до 10 раз с паузой в минуту между проверками.

- Если через 10 попыток (10 минут) публикация ещё не появилась, то можно обновить статус, разослать уведомление на почту через задачу уведомления (описано в Руководстве Разработчика QP8.CMS) и завершить процесс.

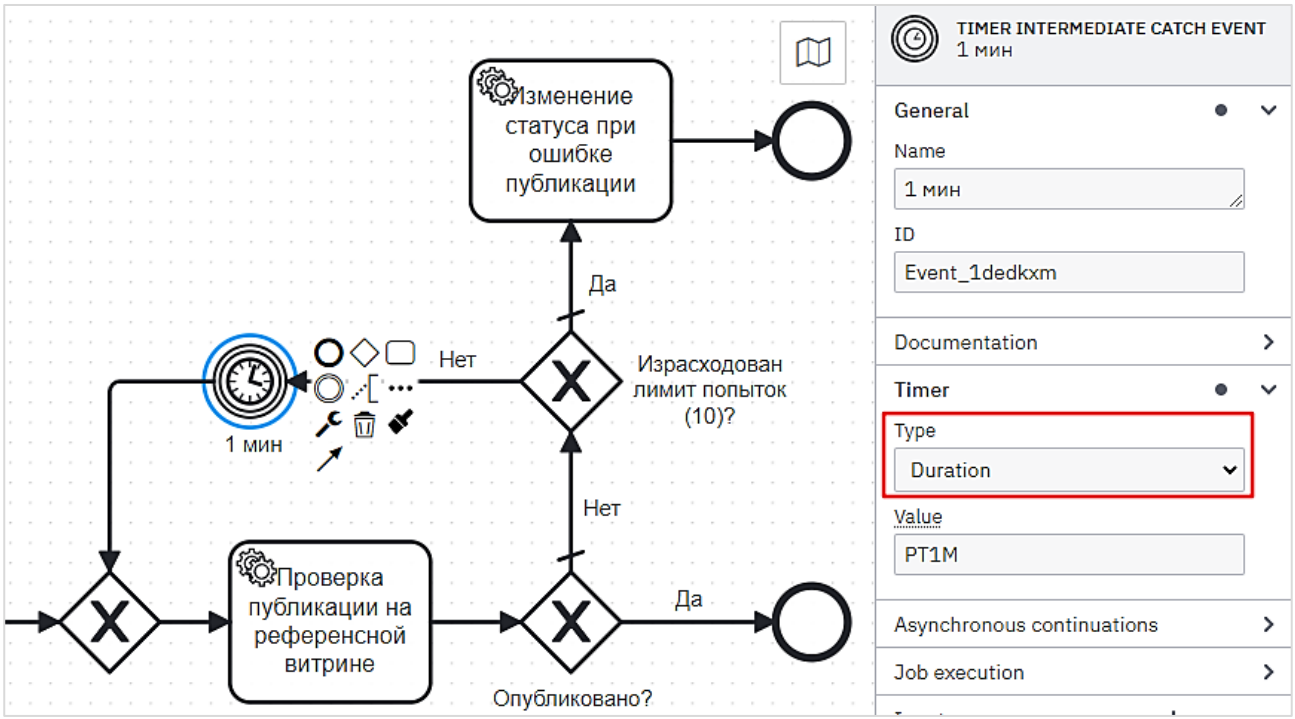


Рисунок 8.10. Использование таймера при проверке достижения лимита попыток

9. Виртуальные контенты

Виртуальные контенты – это данные, находящиеся вне QP8. Позволяют получить доступ к удаленным данным.

9.1. Версии продуктов

Функциональность «Версии продуктов» позволяет получить информацию:

- актуальное состояние витрины на заданную дату;
- состояния продукта до заданной даты, т.е. продукт удален или в него были внесены изменения, и он был опубликован. В контенте отображаются все действия с продуктом до указанной даты.

Функционал реализован в виде виртуального контента «Версии продуктов». Контент размещается по пути: «Сайт → Виртуальные контенты → Консолидация».

9.1.1. Источники данных функционала

Для хранения версий продуктов в БД QP заведена таблица `ProductVerions`, к которой обращается виртуальный контент «Версии продуктов». Подробнее см. DPC.Front

9.1.2. Первоначальная синхронизация истории продуктов и референсной витрины

Для того чтобы отслеживать историю изменений продуктов, необходимо выполнить первичную синхронизацию референсной витрины, таблица `Products`, с таблицей `ProductVerions` БД QP.

Примечание: в начальной версии DPC синхронизация уже проведена и не требуется повторное выполнение.

За первичную синхронизацию референсной витрины и таблицы БД отвечает процедура `SyncProductVersions`.

Примечание: если синхронизируется витрина, на которой существуют публикации, с таблицей, хранящей историю продуктов, то в контенте «Версии продуктов» будет отображаться история витрины начиная от даты синхронизации. Для синхронизации необходимо вызывать процедуру `SyncProductVersions`.

9.1.3. Фильтрация версий продукта

Для того чтобы получить **историю продуктов по** определенному **каналу**, необходимо:

1. Установить значения фильтрам:
   - `Language`;
   - `Format`;
   - `isLive`.
2. Запустить фильтрацию, кликнув по кнопке «Применить».

Для того чтобы получить **историю определенного продукта по определенному каналу**, необходимо:

1. Установить значения фильтрам:
   - `Language`;
   - `Format`;
   - `isLive`;
   - `ProductId`.

2. Запустить фильтрацию, кликнув по кнопке «Применить».

Для того чтобы получить **историю определенного продукта по определенному каналу** и заданному **времени**, необходимо:

1. Установить значения фильтрам:
   - `Language`;
   - `Format`;
   - `isLive`;
   - `ProductId`;
   - `modification`.
2. Запустить фильтрацию, кликнув по кнопке «Применить».

9.1.4. Структура данных контентов

Примечание: структура виртуального контента формируется из результата запроса, который задается в свойствах виртуального контента в группе настроек «Параметры виртуального контента».

Версии продуктов

В таблице 36 описана структура данных виртуального контента «Версии продуктов».

Таблица 36. Структура данных виртуального контента «Версии продуктов»

Параметр	Тип данных	Описание
modification	DateTime	Дата и время модификации продукта. Задается в формате: `MM/DD/YYYY hh:mm:ss AM/PM`, где: • MM – месяц; • DD– день; • YYYY – год; • hh - часы в 12-часовом формате • mm – минуты; • ss – секунды.
deleted	Boolean	Если флаг установлен, то продукт был удален
ProductId	Numeric	Идентификатор продукта
Type	String	Тип продукта. Например, тариф
Format	String	Формат, в котором представлено описание продукта. Например, JSON
Language	Sting	Язык, на котором публикуется продукт. Например, invariant – язык, установленный по умолчанию для региона
IsLive	Boolean	Если флаг установлен, то продукт опубликован на live – витрине
Title	String	Название продукта
Data	Textbox	Структура продукта в формате, который указан в параметре `Format`.

		Примечание: у продуктов с установленным флагом `deleted` выводится только идентификатор и тип продукта

Витрина

Контент отображает состояние консолидированной витрины на текущую дату. Т.е. витрина представлена одним веб-приложением с единой БД, в которой хранятся данные о публикации продуктов по всем каналам DPC.

Примечание: в предыдущих версиях DPC для каждого канала заводилась отдельная витрина, т.е. отдельное веб-приложение со своим адресом и БД.

В таблице 37 описана структура данных виртуального контента «Витрина».

Таблица 37. Структура данных виртуального контента «Витрина»

Параметр	Тип данных	Описание
ProductId	Numeric	Идентификатор продукта
Type	String	Тип продукта. Например, тариф
Format	String	Формат, в котором представлено описание продукта. Например, JSON
Language	Sting	Язык, на котором публикуется продукт. Например, invariant – язык, установленный по умолчанию для региона
IsLive	Boolean	Если флаг установлен, то продукт опубликован на live – витрине
Title	String	Название продукта
Data	Textbox	Структура продукта в формате, который указан в параметре `Format`

Витрины раньше

Отличается от функциональности «Витрина» тем, что в свойствах виртуального контента, в группе настроек «Параметры виртуального контента», задан SQL-запрос к функции `GetProducts`. Функция принимает в качестве входного параметра дату. В результате выполнения функция `GetProducts` возвращает продукты, актуальные на переданную дату.

Примечание: в функционале «Витрина» и «Витрины раньше» отображается только факт публикации продукта. Т.е. получить информацию об удалении продукта невозможно.

Статистика витрины

Отображает количество публикаций, удалений и сколько затронуто продуктов за день.

В таблице 38 описана структура данных виртуального контента «Статистика витрины».

Таблица 38. Структура данных виртуального контента «Статистика витрины»

Параметр	Тип данных	Описание
date	DateTime	Дата и время, за которые формируется статистика

put	Numeric	Количество публикаций
delete	Numeric	Количество удалений
products	Numeric	Количество затронутых продуктов

9.1.5. Версии локализованного продукта

При публикации нового локализованного продукта или изменении локализованных полей, с дальнейшей публикацией продукта, в контенте «Версии продуктов» создается запись под каждое действие. В контентах «Витрина» и «Витрина раньше», если продукт уже был опубликован, изменяется только время редактирования продукта (поле `modified`).

9.2. Неправильно удаленные продукты

В данном виртуальном контенте отображаются продукты на витрине, удовлетворяющие хотя бы одному из условий:

- Продукта нет в DPC;
- Продукт DPC заархивирован;
- Продукт имеет статус Invisible (Видимость \ Тип расписания: Не показывать).

Данные продукты автоматически удаляются с витрин, если отправить их с частичной отправкой по идентификатору.

9.3. Продукты только на Stage

Данные виртуальный контент отображает продукты, находящиеся на stage-витрине, которых нет на live-витрине.

10. Локализация DPC

Локализация – это представление продуктов на соответствующем языке и культуре. Например, используется для представления карточки продукта (вызов действия «Продукты») или при публикации продуктов на витрину.

Внимание! Локализация может не входить в DPC.

Если в DPC присутствует локализация, то ее настройки задаются в группе контентов «Локализация».

Примечание: настройки контента «Локализация» используются только в том случае, если каналы публикации настроены в БД QP.

В настройках клиентского кода параметрами `LOCALIZATION_CONTENT_ID`, `LOCALIZATION_MAP_CONTENT_ID` задаются идентификаторы соответственно контентов «Локализация» и «Мэппинг локализации». Если контенты не указаны, то локализация не применяется при сборке продуктов.

Примечание: подробное описание контентов «Локализация», «Мэппинг локализации».

Локализация позволяет задавать полю продукта соответствующие значения на соответствующем языке. При сборке продукта будет выведено то значение, которое соответствует указанному языку или культуре.

Если у продукта задана локализация, то в карточке продукта существует возможность выбора языка представления в группе настроек «Локализация» (рис. 10.1).

Рисунок 10.1. Группа настроек «Локализация»

Выбор языка в группе настроек «Локализация» меняет поля имеющие значения на выбранном языке. Поля не имеющие представления на выбранном языке не меняют свое состояние.

Настройка `Invariant Language (Invariant Country)` задает язык или культуру текущего региона.

Значения для полей на разных языках задаются в форме редактирования статьи, которая отвечает за представление продукта (см. приложение А). Например, для поля заголовок статьи (поле `Title`) задаются значения `Title_en`, `Title_ru` соответственно заголовок на английском языке и русском.

Для каждого языка заводится своя референсная витрина и если значения полей в DPC, имеющих локализацию, отличаются от продукта на витрине, то выводится таблица статусов локализации «Актуальность продуктов» (рис. 10.2).

Актуальность продукта

Продукт не актуален

Язык витрины	Статус на витрине	Дата последней публикации	Опубликовал
Invariant Language (Invariant Country)	**Содержит неотправленные изменения**	25.10.2017 17:24:19	
русский	Актуален	25.10.2017 17:24:18	
English	Актуален	25.10.2017 17:24:18	

Рисунок 10.2. Таблица статусов локализаций «Актуальность продукта»

Для того, чтобы продукт был обновлен на витрине необходимо вызвать пользовательское действие «Публиковать». На витрину публикуется значения поля соответствующее настройкам языка канала. Если возможность локализации отключена, то публикуются все поля.

10.1. Настройка локализации

Для того, чтобы задать локализацию публикуемого продукта необходимо:

1. Установить язык публикуемых продуктов в настройках канала (см. Добавление нового канала);
2. Зарегистрировать в контенте «Локализация» статью, в которой указать:
   - язык локализации. Языки регистрируются в контенте «Языки»;
   - суффикс для языка, зарегистрированного на первом шаге;
   - продукты, к которым необходимо применить локализацию.

Примечание: для того, чтобы применить локализацию к нескольким контентам существует связующий контент «Мэппинг локализации». Статьи контента хранят информацию о локализации, которая применяется к указанному контенту. Контент указывается виртуальный. Виртуальный контент содержит контенты, у которых есть описание продукта.

3. В статье продукта завести поля с зарегистрированным суффиксом. Например, `Title_en`, где _en – это суффикс.

При публикации продукта будет опубликовано поле с суффиксом, соответствующим языку, указанному в настройках канала публикации. Если локализация нет в DPC или отключена, то будут опубликованы все поля продукта.

Для того, чтобы карточка продукта была локализована необходимо выполнить шаги 2 и 3 вышеприведенного списка и задать параметр `Localize` пользовательскому действию «Продукт».

11. Конфигурация QP

Таблица 39. Настройки NotificationSender

Название настройки	Назначение
NOTIFICATION_SENDER_CHANNELS_CONTENT_ID	Идентификатор контента, в котором содержатся данные конфигурации для службы NotificationSender
NOTIFICATION_SENDER_CHECK_INTERVAL	Период времени между попытками публикации службой NotificationSender. **Примечание:** значение в секундах
NOTIFICATION_SENDER_ERROR_COUNT_BERORE_WAIT	Количество попыток публикации до паузы (служба NotificationSender)
NOTIFICATION_SENDER_FORMATTERS_CONTENT_ID	Идентификатор контента, в котором содержатся данные о форматах, в которых инструмент может передавать данные о продуктах на Витрину (служба NotificationSender)
NOTIFICATION_SENDER_PACKAGE_SIZE	Количество записей в очереди публикации, обрабатываемое службой за один раз (служба NotificationSender)
NOTIFICATION_SENDER_TIMEOUT	Время ожидания ответа от Витрины
NOTIFICATION_SENDER_WAIT_INTERVAL_AFTER_ERRORS	Длительность паузы (служба NotificationSender). **Примечание:** значение в секундах

12. Структура контентов

12.1. Индексы Elastic

Описание структуры данных контента «Индексы Elastic» приведено в таблице 40.

Таблица 40. Структура данных контента «Индексы Elastic»

Название поля	Дружественное название	Тип данных	Описание
Name	Имя индекса	String	Имя индекса
Address	Адрес Elastic	String	Адрес веб-приложения
State	Состояние	String	Окружение. Например, live или stage
IsDefault	По умолчанию	Boolean	Если флаг установлен, то текущая конфигурация применяется по умолчанию, если задается неполный адрес обращения к Elastic Sync API. Применяется для упрощенного обращения (роутинга)
DoTrace	Включить трассировку	Boolean	Если флаг установлен, то в файл логирования записываются все операции с Elasticsearch. В записи действия указывается адрес, на который был послан запрос, запрос к Elasticsearch и ответ Elasticsearch
ReindexChannel	Канал для переиндексации	One-to-Many Relation	
Date	-	Дата-Время	При задании этого параметра необходимо сначала выполнить переиндексацию (при этом данные будут получены из `ReindexChannel` на заданную дату и время), а затем можно получать состояние витрины обычным образом

12.2. Каналы

Контент содержит конфигурации каналов публикации. Структура контента приведена в таблице 41.

Таблица 41. Структура данных контента «Каналы»

Название поля	Дружественное название	Тип данных	Описание
Name		String	Названия канала
Language	Язык	One-to-Many Relation	Язык публикуемого продукта. **Примечание:** если в DPC нет локализации этого поля может не быть

Url		String	Ссылка на витрину, на которую публикуются продукты через канал
Format		One-to-Many Relation	Формат, в котором публикуются продукты
IsStage		Boolean	Если флаг установлен, то продукты публикуются на stage
DegreeOfParallelism		Numeric	Количество параллельно публикуемых продуктов. **Примечание:** если одновременно публикуются продукты с одинаковым идентификатором, то продукты публикуются по одному
Filter		String	Поле позволяет задать условия фильтрации продуктов публикации. Пример фильтрации: `[Type = Tariff']`

12.3. Лимиты Highload API

Описание структуры данных контента «Лимиты Highload API» приведено в таблице 42.

Таблица 42. Структура данных контента «Лимиты Highload API»

Название поля	Дружественное название	Тип данных	Описание
User	Пользователь	One-to-Many Relation	Пользователь Highload API. Пользователи регистрируются в контенте «Пользователи Highload API»
ApiMethod	Метод API	String	Методы Highload API. Методы регистрируются в контенте «Методы Highload API»
Seconds	Секунды	One-to-Many Relation	Интервал времени, в который можно выполнить запросы
Limit	Лимит	Numeric	Количество запросов, которые можно выполнить за указанный интервал времени

12.4. Методы Highload API

Описание структуры данных контента «Методы Highload API» приведено в таблице 43.

Таблица 43. Структура данных контента «Методы Highload API»

Название поля	Дружественное название	Тип данных	Описание
Title		String	Название метода

12.4.1. Дополнительные методы (Highload API Methods)

Название поля	Тип данных	Описание
Title	String	Короткое название в запросе, должно быть уникальным (алиас метода)
System	Булевое	Только для системных методов, которые по умолчанию - GetById, GetByType, Search
JSON	JSON	Body POST-запроса в формате DPC SearchAPI, который ограничит данные для определяемого метода.

12.5. Описания карточек продуктов

Описание структуры данных контента «Описания карточек продуктов» приведено в таблице 44.

Таблица 44. Структура данных контента «Описания карточек продуктов»

Название поля	Дружественное название	Тип данных	Описание
Title		String	Название карточки продукта
XmlDefinition	Описание контрола	Textbox	XML – разметка карточки продукта
ApplyToTypes		Many-to-Many Relation	Тип продукта
Content		One-to-Many Relation	Задает контент карточки продукта, которая описывается в статье.

12.6. Описания продуктов

Описание структуры данных контента «Описания продуктов» приведено в таблице 45.

Таблица 45. Структура данных контента «Описания продуктов»

Название поля	Дружественное название	Тип данных	Описание
Title		String	Название описываемого продукта
XmlDefinition		Textbox	XML – разметка продукта
ApplyToTypes		Many-to-Many Relation	Определяет тип продукта
Content		One-to-Many Relation	Задает контент продукта, который описывается в статье. **Примечание:** статьи этого контента могут являться корневыми статьями продукта

12.7. Пользователи Highload API

Описание структуры данных контента «Пользователи Highload API» приведено в таблице 46.

Таблица 46. Структура данных контента «Пользователи Highload API»

Название поля	Дружественное название	Тип данных	Описание
Name	Имя	String	Имя пользователя
AccessToken	Токен доступа	String	Ключ доступа **Примечание:** если ключ доступа не задан (пользователь Default), то осуществляется анонимный доступ

12.8. Сервисы

Описание структуры данных контента «Сервисы» приведено в таблице 47.

Таблица 47. Структура данных контента «Сервисы»

Название поля	Дружественное название	Тип данных	Описание
Title		String	Название сервиса
Slug		String	Название сервиса, используемое при вызове методов Web API
Definition		One-to-Many Relation	Описание продукта в XML-разметке. Описание продукта хранится в контенте «Описания продуктов»
Filter		String	Условия фильтрации продуктов
Version		String Enum	Версия сервиса. Позволяет создавать несколько статей одного и того же сервиса, но с разным описанием продуктов
Type	Тип продукта	One-to-Many Relation	Тип продукта

12.9. Форматеры

Описание структуры данных контента «Форматеры» приведено в таблице 48.

Таблица 48. Структура данных контента «Форматеры»

Название поля	Дружественное название	Тип данных	Описание
Name		String	Название форматера
Formatter		String	Имя интерфейса, который отвечает за формат продукта. Например, `JsonProductFormatter` позволяет получить продукт в JSON-разметке.
MediaType		String	http-заголовок запроса

12.10. Локализация

Описание структуры данных контента «Локализация» приведено в таблице 49.

Таблица 49. Структура данных контента «Локализация»

Название поля	Дружественное название	Тип данных	Описание
Language	Язык	One-to-Many Relation	Язык, к которому добавляется суффикс. Языки регистрируются в контенте «Языки»
Suffix	Суффикс	String	Суффикс добавляется к языку
Content	Применять к продуктам	Relation Many-to-One	Контент, к которому

12.11. Мэппинг локализации

Описание структуры данных контента «Мэппинг локализации» приведено в таблице 50.

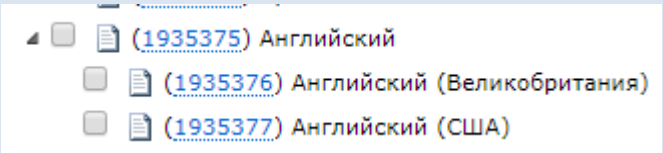
Таблица 50. Структура данных контента «Мэппинг локализации»

Название поля	Дружественное название	Тип данных	Описание
Localization	Локализация	One-to-Many Relation	Локализация
Content	Применять к продуктам	One-to-Many Relation	Виртуальный контент, к которому применяется локализация

12.12. Языки

Описание структуры данных контента «Языки» приведено в таблице 51.

Таблица 51. Структура данных контента «Языки»

Название поля	Дружественное название	Тип данных	Описание
Language	Язык	String	Язык
Code	Код	String	Код языка. Например, код русского языка – ru
Parent		One-to-Many Relation	Связь с локализацией по типу культуры. Например, параметр `Language` имеет значение `en-US` (Английский (США)), а поле `Parent` указывает на статью английский язык (рис. 12.1).  *Рисунок 12.1. Поле Parent контента «Языки»*

13. Структура данных таблиц

13.1. Таблица Messages БД dpc_notifications

В таблице 52 описана структура данных таблиц «Messages» БД dpc_notifications.

Таблица 52. Таблица «Messages» БД dpc_notifications

Название	Тип	Описание
Id	int	Идентификатор записи в очереди
Channel	nvarchar	Название канала, к которому относится запись в очереди
Created	Datetime	Дата и время создания записи в очереди
Data	nvarchar	Форматированные данные о продуктах для передачи на Витрину
Method	nvarchar	Метод HTTP, который необходимо использовать при обращении к Каналу
UserID	int	Идентификатор пользователя в Системе “ПК”, инициировавшего публикацию
UserName	varchar	Имя и фамилия пользователя в Системе “ПК”, инициировавшего публикацию
DataKey	int	Идентификатор продукта

14. Руководство по установке QP8.ProductCatalog

14.1. Установка на Windows

14.1.1. Требования

- Консоль PowerShell v5.1 или выше
- IIS 8.5 или выше с установленными средствами автоматизации (PS-модуль WebAdministration)
- В версии для SQL Server:
  - Установленный PS-модуль SqlServer или SqlPs
  - Доступный экземпляр SQL Server 2012 или выше
- В версии для PostgreSQL:
  - Установленный Postgres CLI 11 или 12 версии (psql, pg_restore)
  - Доступный экземпляр PostgreSQL / Postgres Pro 11 или выше

Примечание: для работы Postgres CLI может потребоваться дополнительная установка компонента (https://support.microsoft.com/ru-ru/help/2977003/the-latest-supported-visual-c-downloads), если он не был установлен в системе ранее.

Примечание: путь к установленному PostgreSQL / Postgres Pro необходимо добавить в переменную PATH.

- ASP.NET Core Runtime 3.1.13 (Hosting Bundle)

Примечание: если ASP.NET Core Runtime устанавливается в первый раз, необходима перезагрузка компьютера, иначе не будут работать Windows-службы.

- Доступный кластер или одиночный сервер Elasticsearch 5, 6 или 8 версии
- Установленный QP8

Примечание: при установке QP8.ProductCatalog для PostgreSQL требуется установка версии QP8 для ASP.NET Core с поддержкой PostgreSQL.

14.1.2. Необходимые права

- Права локального администратора (для копирования файлов на локальном хосте, настройки IIS, создания и запуска Windows служб)
- В версии для SQL Server:
  - Копирования файлов на сетевую шару SQL Server
  - Права администратора на SQL Server (для восстановления базы из бэкапа и запуска скриптов).

Примечание: если не задаются имя пользователя и пароль для SQL Server, предполагается наличие Windows-аутентификации, настроенной для пользователя, запускающего скрипт инсталляции

- В версии для PostgreSQL:
  - Права администратора на PostgreSQL (для восстановления базы из бэкапа и запуска скриптов).

14.1.3. Автоматическая установка

1. Скачать и распаковать дистрибутив (перед распаковкой необходимо разблокировать архив (рис. 14.1), чтобы в дальнейшем извлекаемые из него файлы были также разблокированы):

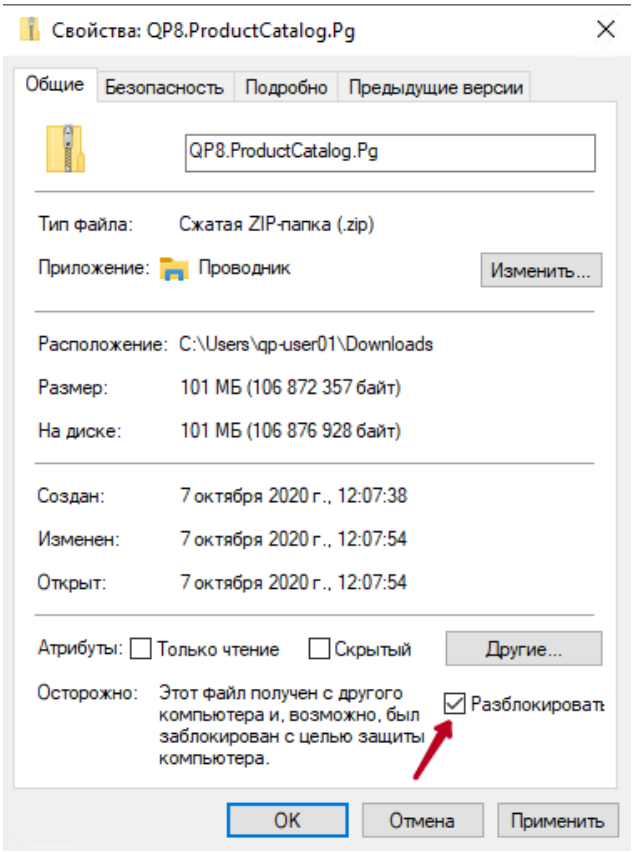


Рисунок 14.1. Разблокировка архива с дистрибутивом продукта

2. Запустить Install.bat (для SQL Server) или install_pg.bat (для PostgreSQL) от имени Администратора
3. Откроется консоль, в которой будут запрошены параметры:
   - `CustomerCode` - кастомер код каталога (уникальное наименование клиента/проекта)
   - `DB server` - имя хоста SQL Server или PostgreSQL, где будет установлена база каталога
   - `Db login` - логин администратора (только для PostgreSQL)
   - `Db password` - пароль администратора (только для PostgreSQL)
   - `Elasticsearch` - адрес Elasticsearch. В случае кластера можно задавать в виде http://host1:port; http://host2:port
   - `Backend port` - порт по которому доступен бэкэнд QP, обычно это 89

Примечание: Параметр `CustomerCode` также используется как имя создаваемой базы данных. При повторном запуске нужно указать новый кастомер код, если требуется сохранить текущие изменения в базе данных, иначе она будет пересоздана в процессе инсталляции

4. На время установки откроется консоль PowerShell. Все действия установки сохранятся в файл Install.log в папке C:\QA\Logs. Проверить файл логов, что там нет ошибок. Если были ошибки, то устранить их причину и повторно запустить Install.bat (install_pg.bat)
5. После установки зайти в бэкэнд каталога по URL http://<хост установки>:<порт установки>, для авторизации выбрать установленный кастомер код каталога, логин и пароль по-умолчанию – admin/admin.

Примечание: В качестве имени хоста вместо localhost нужно использовать название компьютера, которое можно узнать, например, через команду hostname, иначе не будут сохраняться cookies и часть функциональности будет недоступна.

6. В секции `Product Catalog` вызвать контекстное меню `HighloadFront`. В открывшейся вкладке нажать кнопки `индексировать`. Это создаст индексы в Elasticsearch.

14.1.4. Ручная установка

Служит альтернативой Install.bat (install_pg.bat). Позволяет использовать дополнительные параметры для управления процессом установки.

1. Запустить консоль PowerShell от имени администратора
2. Перейти в каталог .\Install\
3. Команда `Get-Help .\InstallConsolidationCatalog.ps1 -detailed` вернет подробное описание скрипта уснановки, его параметров и примеры использования.
4. Запустить `.\InstallConsolidationCatalog.ps1` с нужными параметрами

Скрипт установки работает по следующему сценарию:

1. Проводится валидация параметров на существующий путь к файлу/каталогу:
   7. `sourceBackupPath`
   8. `currentSqlPath`
   - `installRoot`
2. Проверка окружения:
   - Установлен .NET Core Runtime нужной версии
   - Установлен QP8
   - Доступен SQL Server или PostgreSQL по указанному хосту
3. Опционально (в зависмости от параметра `cleanUp`)
   - Удаляются все компоненты каталога
   - Удаляются их файлы
   - Удаляется кастомер код каталога из QP

Таким образом, при повторном запуске, можно обновлять каталог, делать новый биндинг портов и т.п.

4. Проверяется доступность портов, передаваемых в параметрах:
9. `actionsPort`
10. `notifyPort`
11. `frontPort`
12. `searchApiPort`
13. `syncApiPort`
14. `webApiPort`.

Они будут зарезервированы за компонентами каталога и должны быть доступны на момент установки.

5. Устанавливаются компоненты каталога:
   - Dpc.Admin: Бэкэнд каталога
   - DPC.ActionsService: Сервис выполнения задач
   - DPC.NotificationSender: Сервис публикации продуктов
   - Dpc.Front: Референсная витрина
   - Dpc.SyncApi: Витрина индексации продуктов в Elasticsearch
   - Dpc.SearchApi: Поиск продуктов по индексам Elasticsearch
   - Dpc.WebApi: API каталога
6. Создается новый кастомер код для каталога

7. Копируется бэкап на сервер БД (только для SQL Server)
8. Восстанавливается база данных из бэкапа
9. База данных обновляется до актуального состояния
10. В базе обновляются настройки каталога, зависящие от параметров скрипта
11. Регистрируется в QP кастомер код каталога

14.2. Установка на Linux

14.2.1. Требования

- PostgreSQL / PostgresPro версии 12 или выше (но для восстановления нужен pg_restore от 12 версии)
- ElasticSearch версий 5.x, 6.x или 8.x / OpenSearch
- Установленный продукт QP8.CMS с поддержкой PostgreSQL (в том же варианте, в котором планируется разворачивать QP8.ProductCatalog: с использованием Docker, без использования Docker, с использованием Kubernetes)

14.2.2. Установка БД

1. Скачать БД демо-сайта QP8.ProductCatalog, а также пример БД для PostgreSQL (для получения актуального скрипта current.sql)
2. Создать новую пустую БД в PostgreSQL, например, с именем *qmobile_catalog*
3. Восстановить БД с помощью утилиты `pg_restore` из файла дампа, распакованного на шаге 1, в пустую БД *qmobile_catalog*, созданную на шаге 2. Пример команды (необходимо использовать реальные логин, пароль и имя сервера вместо *dbuser*, *dbpass* и *dbserver*, вместо DUMP_PATH указывается путь к распакованному файлу дампа):

```
pg_restore -Fc -d 'postgresql://dbuser:dbpass@dbserver/qmobile_catalog' -j 4 'DUMP_PATH'
```

4. Выполнить файл `current.sql`, распакованный на шаге 1, на БД *qmobile_catalog,* предварительно переключив текущий сеанс на схему qp с помощью команды:

```
SET search_path TO qp;
```

5. Создать пользователя, например, с именем *qmobile* и дать ему права на базу данных. Список прав:

```
GRANT CONNECT,TEMP ON DATABASE qmobile_catalog TO qmobile;
GRANT USAGE,CREATE ON SCHEMA qp TO qmobile;
GRANT SELECT,INSERT,UPDATE,DELETE,REFERENCES,TRIGGER ON ALL TABLES IN SCHEMA qp TO qmobile;
GRANT USAGE,SELECT,UPDATE ON ALL SEQUENCES IN SCHEMA qp TO qmobile;
GRANT EXECUTE ON ALL ROUTINES IN SCHEMA qp TO qmobile;
ALTER ROLE qmobile SET search_path TO qp,public;
ALTER DEFAULT PRIVILEGES IN SCHEMA qp GRANT SELECT,INSERT,UPDATE,DELETE,REFERENCES,TRIGGER ON
TABLES TO qmobile;
ALTER DEFAULT PRIVILEGES IN SCHEMA qp GRANT USAGE,SELECT,UPDATE ON SEQUENCES TO qmobile;
ALTER DEFAULT PRIVILEGES IN SCHEMA qp GRANT EXECUTE ON ROUTINES TO qmobile;
CALL qp_change_contents_ownership('qmobile');
```

Замечание: если в дальнейшем под созданным пользователем не будет выполняться обновление БД при переходе на новую версию, а вместо этого будет использована административная учётная запись, то права REFERENCES и TRIGGER для таблиц и UPDATE для последовательностей можно не задавать.

6. Добавить в конфигурационный файл QP `config.xml` новую запись `customer`, либо модифицировать дефолтную запись `customer`, указав актуальную строку подключения к БД *qmobile_catalog* с созданным пользователем *qmobile*

Примечание: если необходимо запускать приложение под стандартным пользователем *postgres,* нужно добавить в строку подключения дополнительный параметр для поиска схемы `;Search Path=qp,public;`

7. Настроить адрес сервера ElasticSearch/OpenSearch, если он отличается от localhost:9200:

```
update content_data set data = replace(data, 'localhost:9200', 'elastic_host:9200') where content_item_id in (1935953, 1935952) and attribute_id = 2143

update content_item set modified = now() where content_item_id in (1935953, 1935952);
```

8. В случае использования внешних DNS для разворачиваемых сервисов DPC необходимо выполнить скрипт замены localhost на DNS (при этом вместо `dpc-admin.test`, `qp-storage.test` нужно подставить реальные DNS соответствующих компонентов):

```
update custom_action set url = replace(url, 'localhost:7000', 'dpc-admin.test'), icon_url = replace(icon_url, 'localhost:7000', 'dpc-admin.test'), modified = now();

update site set upload_url_prefix = replace(upload_url_prefix, 'localhost:5000', 'qp-storage.test'), modified = now();
```

9. Как альтернатива предыдущему варианту: если приложение используется внутри сети, то вместо DNS можно использовать сетевое имя компьютера, на котором выполняется развёртывание (в формате **имя компьютера:порт**). В этом случае порты остаются теми же и необходимо выполнить скрипт замены `localhost` на имя компьютера (вместо `hostname` нужно подставить реальное имя компьютера):

```
update custom_action set url = replace(url, 'localhost', 'hostname'), icon_url = replace(icon_url, 'localhost', 'hostname'), modified = now();

update site set upload_url_prefix = replace(upload_url_prefix, 'localhost', 'hostname'), modified = now();
```

10. При вариантах разворачивания, отличных от Kubernetes (нативном разворачивании или использовании docker) нужно выполнить замену DNS `dpc-admin.dpc` на `hostname:7000` (вместо `hostname` нужно подставить реальное имя компьютера):

```
update custom_action set url = replace(url, 'dpc-admin.dpc', 'hostname:7000'), modified = now();

update site set xaml_dictionaries = replace(xaml_dictionaries, 'dpc-admin.dpc', 'hostname:7000'), modified = now();
```

14.2.3. Установка на Linux (с использованием Docker)

1. Скачать архив qp8-product-catalog-manifests.tar.gz и распаковать его содержимое в `/etc/dpc-config.`
2. Перейти в папку `/etc/dpc-config/compose` и выполнить команду: `sudo docker-compose up -d`

Примечание: При необходимости можно обновить версии докер-образов и задать параметры подключения к конфигурационному сервису QP в файле .env

14.2.4. Установка на Linux (без использования Docker)

Базовые действия

- При сборке из исходного кода:
  - вытягиваем исходники из репозитория на github;
- При использовании готовых бинарных файлов:
  - Выкачиваем архив qp8-product-catalog-linux.tar.gz, содержащий бинарные файлы продуктового каталога.
  - Распаковываем полученный архив, в домашний каталог пользователя, созданного в рамках установки QP8.CMS (по умолчанию - `/home/qp`). Должна получиться такая структура каталогов:
    - DPC.ActionsService
    - DPC.Admin
    - DPC.API
    - DPC.Front
    - DPC.NotificationsSender
    - DPC.Search
    - DPC.Sync

Установка DPC.Admin

- Только при сборке из исходников:
  - в папке `QA.ProductCatalog.Admin.WebApp` выполняем установку npm-пакетов командой `npm ci`;
  - собираем фронт командой `npm run build`;
  - в папке проекта `QA.ProductCatalog.Admin.WebApp` (там должен быть файл `QA.ProductCatalog.Admin.WebApp.csproj`) выполняем команду на публикацию:

```
dotnet publish "QA.ProductCatalog.Admin.WebApp.csproj" -c Release -o bin/release/publish/ -r
linux-x64 --self-contained=false
```

  - в папке `/home/qp` создаём подпапку `DPC.Admin` и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.Admin`.
- Переходим в папку `DPC.Admin`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.

  - В значении параметра `RestNotificationUrl` указываем адрес и порт, на котором будет работать компонент **DPC.NotificationsSender**. В случаях использования приложения только внутри сети можно задать значение `http://<host_name>:7200`, где *host_name* - имя комьютера. Если все сервисы разворачиваются на одном компьютере и пользователь будет открывать приложение через браузер также на этом компьютере, то можно оставить значение по умолчанию `http://localhost:7200`.
  - В значении параметра `HighloadFrontSyncUrl` указываем адрес и порт, на котором будет работать компонент **DPC.Sync.** В случаях использования приложения только внутри сети можно задать значение `http://<host_name>:7600`, где *host_name* - имя комьютера. Если все сервисы разворачиваются на одном компьютере и пользователь будет открывать приложение через браузер также на этом компьютере, то можно оставить значение по умолчанию `http://localhost:7600`.
- Внесём изменения в файл `NLogClient.config:`
  - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-admin/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-admin/internal-nlog.txt.`
  - В секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменяем его на `fileStructured`.
- Создаём на сервере директорию `/var/log/dpc-admin/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-admin.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Admin
After=qp.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.Admin/
ExecStart=/bin/dotnet QA.ProductCatalog.Admin.WebApp.dll --urls http://*:7000

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-admin.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-admin.service
```

Установка DPC.ActionsService

- Только при сборке из исходников:
  - в папке проекта `QP8.ProductCatalog.ActionsService` (там должен быть файл `QP8.ProductCatalog.ActionsService.csproj`) выполняем команду на публикацию:

```
dotnet publish "QP8.ProductCatalog.ActionsService.csproj" -c Release -o bin/release/publish/ -r
linux-x64 --self-contained=false
```

  - в папке `/home/qp` создаём подпапку `DPC.ActionsService` и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.ActionsService`.
- Переходим в папку `DPC.ActionsService`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - В значении параметра `RestNotificationUrl` указываем адрес и порт, на котором будет работать компонент **DPC.NotificationsSender**. В случаях использования приложения только внутри сети можно задать значение `http://<host_name>:7200`, где *host_name* - имя комьютера. Если все сервисы разворачиваются на одном компьютере и пользователь будет открывать приложение через браузер также на этом компьютере, то можно оставить значение по умолчанию `http://localhost:7200`.
- Внесём изменения в файл `NLogClient.config`:
  - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-actions-service/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-actions-service/internal-nlog.txt.`
  - В секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменяем его на `fileStructured`.
- Создаём на сервере директорию `/var/log/dpc-actions-service/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-actions-service.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Actions Service
```

```
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.ActionsService/
ExecStart=/bin/dotnet QP8.ProductCatalog.ActionsService.dll --urls http://*:7300

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-actions-service.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-actions-service.service
```

Установка DPC.API

- Только при сборке из исходников:
  - в папке проекта `QP8.ProductCatalog.WebApi` (там должен быть файл `QP8.ProductCatalog.WebApi.csproj`) выполняем команду на публикацию:

```
dotnet publish "QP8.ProductCatalog.WebApi.csproj" -c Release -o bin/release/publish/ -r linux-x64
--self-contained=false
```

  - в папке `/home/qp` создаём подпапку `DPC.API` и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.API`.
- Переходим в папку `DPC.API`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - В значении параметра `RestNotificationUrl` указываем адрес и порт, на котором будет работать компонент **DPC.NotificationsSender**. В случаях использования приложения только внутри сети можно задать значение `http://<host_name>:7200`, где *host_name* -

имя комьютера. Если все сервисы разворачиваются на одном компьютере и пользователь будет открывать приложение через браузер также на этом компьютере, то можно оставить значение по умолчанию `http://localhost:7200`.

- Внесём изменения в файл `NLogClient.config`:
  - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-api/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-api/internal-nlog.txt.`
  - В секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменяем его на `fileStructured`.
- Создаём на сервере директорию `/var/log/dpc-api/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-api.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Web API
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.API/
ExecStart=/bin/dotnet QP8.ProductCatalog.WebApi.dll --urls http://*:7100

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-api.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-api.service
```

Установка DPC.NotificationsSender

- Только при сборке из исходников:
  - в папке проекта `QA.Core.DPC.NotificationSender` (там должен быть файл `QA.Core.DPC.NotificationSender.csproj`) выполняем команду на публикацию:

```
dotnet publish "QA.Core.DPC.NotificationSender.csproj" -c Release -o bin/release/publish/ -r linux-x64 --self-contained=false
```

        - в папке `/home/qp` создаём подпапку `DPC.NotificationsSender`
    - и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.NotificationsSender`.
- Переходим в папку `DPC.NotificationsSender`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
    - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
    - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
- В файле `apsettings.json` в секции `Properties` можно изменить значение параметра `InstanceId`. Важно, чтобы у разных экземпляров приложения `InstanceId` различался, а у одного экземпляра приложения он должен совпадать у компонентов **DPC.NotificationSender**, **DPC.Sync** и **DPC.Front**, что требуется для механизма публикации.
- Внесём изменения в файл `NLogClient.config`:
    - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-notification-sender/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-notification-sender/internal-nlog.txt.`
    - В секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменяем его на `fileStructured`.
- Создаём на сервере директорию `/var/log/dpc-notification-sender/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-notification-sender.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Notification Sender
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.NotificationSender/
ExecStart=/bin/dotnet QA.Core.DPC.NotificationSender.dll --urls http://*:7200

[Install]
```

```
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-notification-sender.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-notification-sender.service
```

Установка DPC.Front

- Только при сборке из исходников:
  - в папке проекта `QA.ProductCatalog.Front.Core.API` (там должен быть файл `QA.ProductCatalog.Front.Core.API.csproj`) выполняем команду на публикацию:

```
dotnet publish " QA.ProductCatalog.Front.Core.API.csproj" -c Release -o bin/release/publish/ -r
linux-x64 --self-contained=false
```

    - в папке `/home/qp` создаём подпапку `DPC.Front`
  - и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.Front`.
- Переходим в папку `DPC.Front`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
- В файле `apsettings.json` в секции `Data` можно изменить значение параметра `InstanceId`. Важно, чтобы у разных экземпляров приложения `InstanceId` различался, а у одного экземпляра приложения он должен совпадать у компонентов **DPC.NotificationSender**, **DPC.Sync** и **DPC.Front**, что требуется для механизма публикации.
- Внесём изменения в файл `NLog.config`:
  - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-front/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-front/internal-nlog.txt.`
  - В секции `rules` во всех логгерах, в которых в параметре `writeTo` задано значение `console`, заменяем его на `fileStructured`.
- Создаём на сервере директорию `/var/log/dpc-front/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-front.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Front
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.Front/
ExecStart=/bin/dotnet QA.ProductCatalog.Front.Core.API.dll --urls http://*:7400

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-front.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-front.service
```

Установка DPC.Sync

- Только при сборке из исходников:
  - в папке проекта `QA.ProductCatalog.HighloadFront.Core.API` (там должен быть файл `QA.ProductCatalog.HighloadFront.Core.API.csproj`) выполняем команду на публикацию:

```
dotnet publish " QA.ProductCatalog.HighloadFront.Core.API.csproj" -c Release -o
bin/release/publish/ -r linux-x64 --self-contained=false
```

    - в папке `/home/qp` создаём подпапку `DPC.Sync`
  - и копируем туда всё содержимое каталога `bin/release/publish/`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.Sync`.
- Переходим в папку `DPC.Sync`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в

параметре QpConfigToken конфигурационного файла /home/qp/QP/appsettings.json приложения QP.

- В файле apsettings.json в секции Data можно изменить значение параметра InstanceId. Важно, чтобы у разных экземпляров приложения InstanceId различался, а у одного экземпляра приложения он должен совпадать у компонентов **DPC.NotificationSender**, **DPC.Sync** и **DPC.Front**, что требуется для механизма публикации.
- Внесём изменения в файл NLog.config:
  - Находим находим internalLogFile и <variable name="logDirectory" value= и прописываем там путь /var/log/dpc-sync/. При этом для internalLogFile должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: /var/log/dpc-sync/internal-nlog.txt.
  - В секции rules во всех логгерах, в которых в параметре writeTo задано значение console, заменяем его на fileStructured.
- Создаём на сервере директорию /var/log/dpc-sync/ и выдаём qp пользователю права на владение директорией.
- Создаём файл dpc-sync.service в папке /usr/lib/systemd/system/ и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Sync
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.Sync/
ExecStart=/bin/dotnet QA.ProductCatalog.HighloadFront.Core.API.dll --urls http://*:7600

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка /usr/lib/systemd/system/, то все изменения вносим в папку /lib/systemd/system/.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-sync.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-sync.service
```

Установка DPC.Search

- Только при сборке из исходников:

  - в папке `/home/qp` создаём подпапку `DPC.Search` и копируем туда всё содержимое соседней подпапки `DPC.Sync`. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `DPC.Search`.
- Переходим в папку `DPC.Search`.
- Внесём изменения в файл `appsettings.json` в секцию `Integration`:
  - Задаём значение параметра `ConfigurationServiceUrl`. В качестве значения указываем адрес сервиса конфигурации QP, который мы развернули ранее. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigUrl` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
  - Задаём значение параметра `ConfigurationServiceToken`. В качестве значения указываем JWT-токен. Значение должно совпадать с тем, которое уже прописано в параметре `QpConfigToken` конфигурационного файла `/home/qp/QP/appsettings.json` приложения QP.
- В файле `apsettings.json` в секции `Data` необходимо установить значение параметра `CanUpdate` в `false`.
- В файле `apsettings.json` в параметр `CorsDomains` внесём значение `["*"]`
- Внесём изменения в файл `NLog.config`:
  - Находим находим `internalLogFile` и `<variable name="logDirectory" value=` и прописываем там путь `/var/log/dpc-search/`. При этом для `internalLogFile` должен сохранится конечный файл, т.е. правильно строка будет выглядеть так: `/var/log/dpc-search/internal-nlog.txt.`
- Создаём на сервере директорию `/var/log/dpc-search/` и выдаём `qp` пользователю права на владение директорией.
- Создаём файл `dpc-search.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=DPC Search
After=dpc-admin.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/DPC.Search/
ExecStart=/bin/dotnet QA.ProductCatalog.HighloadFront.Core.API.dll --urls http://*:7500

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-search.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-search.service
```

14.2.5. Установка на Linux (с использованием Kubernetes)

1. Скачать архив qp8-product-catalog-manifests.tar.gz и распаковать его содержимое в `/etc/dpc-config.`
2. Перейти в папку `/etc/dpc-config/kubernetes`
3. При необходимости можно изменить параметры подключения к конфигурационному сервису QP в параметрах `CONFIGURATION_SERVICE_HOST` и `CONFIGURATION_SERVICE_TOKEN`, а также идентификатор экземпляра приложения в параметре `INSTANCE_ID` в файле `configmap.yaml`
4. Применить манифесты в следующем порядке:

```
kubectl apply -f namespace.yaml
kubectl apply -f configmap.yaml
kubectl apply -f deployment.yaml
kubectl apply -f deployment-front.yaml
kubectl apply -f service.yaml
kubectl apply -f service-front.yaml
```

5. При необходимости использовать внешние URL вместо заданных по умолчанию в домене *.test* можно определить их в файлах `ingress.yaml` и `ingress-front.yaml` и применить манифесты:

```
kubectl apply -f ingress.yaml
kubectl apply -f ingress-front.yaml
```

14.2.6. Настройка nginx

Для корректной работы `DPC.Admin` необходимо решить вопрос безопасности, связанный с CORS-ограничениями. Для их обхода следует делать запросы на тот же домен, на котором располагается QP, осуществить это можно с помощью nginx-а.

Необходимые секции представлены в файле `/dpc-config/nginx/nginx.conf`. Детально ознакомится с настройкой nginx, а так же общей механикой – можно по этой ссылке.

Nginx с использованием docker

Если nginx при установке QP8.CMS был запущен в docker, то следует:

1) выполнить команду `docker-compose down`;
2) открыть на редактирование файл nginx.conf предположительно расположенный по пути `/etc/qpconfig/nginx/nginx.conf`;
3) в конфигурационный файл добавить секции из файла `/etc/dpc-config/nginx/nginx.conf`;
4) задать свои DNS вместо заданных в домене `.test`
5) выполнить команду `docker-compose up –d`.

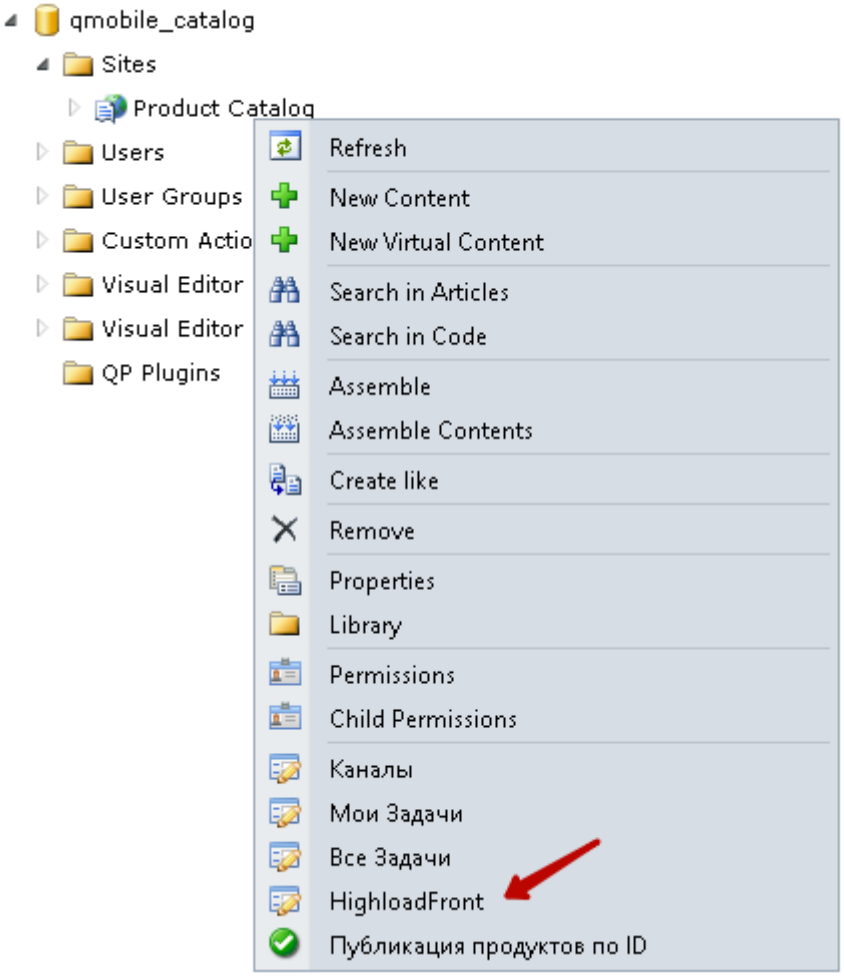
Nginx без использования docker

Если nginx при установке QP8.CMS был запущен не в docker, то следует:

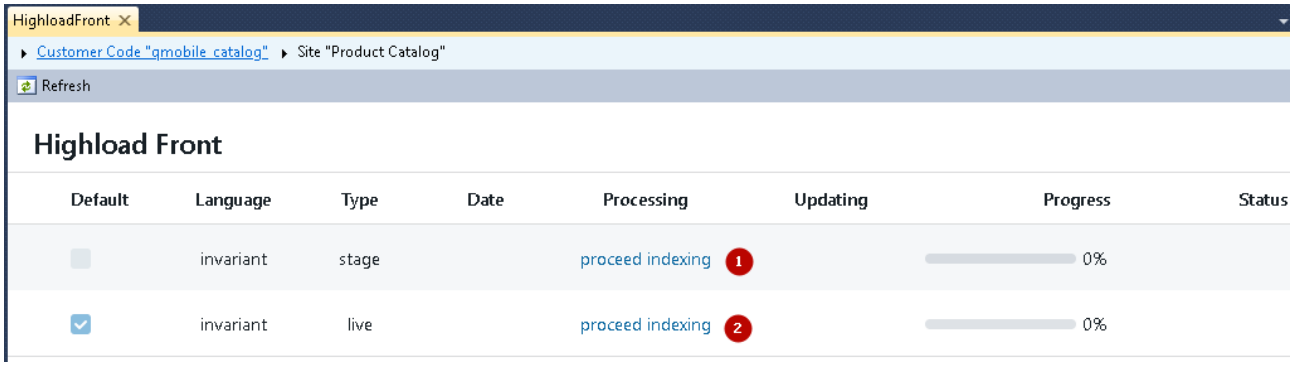
1) открыть на редактирование файл nginx.conf предположительно расположенный по пути `/etc/nginx/nginx.conf`;
2) в конфигурационный файл добавить секции из файла `/home/qp/dpc-config/nginx/nginx.conf`;
3) задать свои DNS вместо заданных в домене `.test`;
4) после чего проверить корректность конфигурации командой `nginx -t`;
5) если всё корректно, то перечитать конфигурацию nginx командой `nginx -s reload`.

14.2.7. Заполнение индексов ElasticSearch / OpenSearch

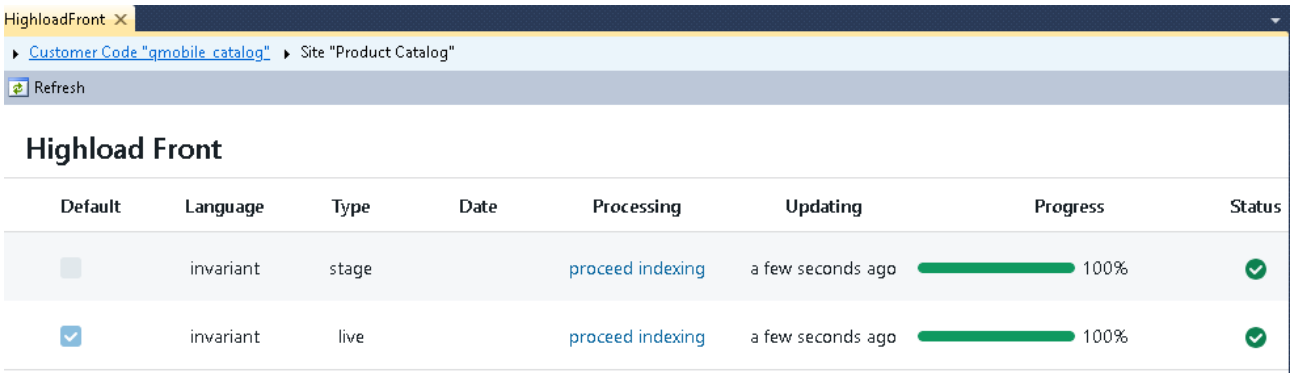
- Залогиниться в систему
- В контекстном меню сайта выбрать пункт **HighloadFront**



- Выполнить поочередную переиндексацию обоих индексов нажатием на соответствующие ссылки



При этом процесс должен завершиться без ошибок:



После этого компонент QP8.ProductCatalog.Search готов к использованию (например, демо-сайтом)

14.3. Установка демо-сайта на Linux

14.3.1. Установка демо-сайта (с использованием Docker)

Внимание! Для установки демо-сайта требуется предварительная установка модулей **Impact** и **PdfGenerator**.

1. Скачать архив qp8-product-catalog-demo-manifests.tar.gz и распаковать его содержимое в `/etc/dpc-demo-config`.
2. Перейти в папку `/etc/dpc-demo-config/compose`
   - При использовании внешних URL исправить в файле *demo.js* адреса компонентов **DPC.Search**, **DPC.Impact** и **DPC.PdfServer** в соответствии с заведёнными DNS (уточнить какой адрес к какому сервису относится можно по файлу *demohost.js*)
   - при испольовании приложения внутри сети заменить в файле *demo.js* `localhost` на сетевое имя компьютера
3. Выполнить команду: `sudo docker-compose up –d`

Примечание: При необходимости можно обновить версии докер-образов и задать параметры подключения к конфигурационному сервису QP в файле .env

14.3.2. Установка демо-сайта (без использования Docker)

- При установке готовых бинарных файлов:
  - Выкачиваем архив qp8-product-catalog-demo.tar.gz, содержащий бинарные файлы демосайта.

  - Распаковываем полученный архив, в домашний каталог пользователя, созданного в рамках установки QP8.CMS (по умолчанию - `/home/qp`). Должен появиться каталог `QMobile.Demo`.
- При сборке из исходников:
  - вытягиваем исходники из репозитория на github;
  - в корневой папке проекта выполняем установку npm-пакетов командой `npm ci`;
  - собираем фронт командой `npm run build`;
  - в папке `/home/qp` создаём подпапку `QMobile.Demo` и копируем туда всё содержимое каталога `build`, в который осуществлялась публикация. Нужно проверить, что у пользователя `qp` есть права на чтение и исполнение содержимого папки `QMobile.Demo`.
- Переходим в папку `QMobile.Demo`.
- В файле `env.js` необходимо задать значения переменных:
  - в качестве значения DPC_HOST нужно указать URL компонента **DPC.SearchApi**.
  - в качестве значения IMPACT_HOST нужно указать URL компонента **DPC.Impact** из модуля **Impact**.
  - в качестве значения DOWNLOAD_PDF_HOST нужно указать URL компонента **DPC.PdfServer** из модуля **PdfGenerator**
- Создаём на сервере директорию `/var/log/dpc-admin/` и выдаём `qp` пользователю права на владение директорией.
- Установить компонент `http-server` командой:

```
npm install --global http-server
```

- Создаём файл `dpc-qmobile-demo.service` в папке `/usr/lib/systemd/system/` и заполняем его следующим содержимым:

```
[Unit]
Description=QMobile Demo
After=dpc-search.service
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=5
User=qp
WorkingDirectory=/home/qp/QMobile.Demo/
ExecStart=http-server -d false -p 8200

[Install]
WantedBy=multi-user.target
```

Внимание: если в конкретном дистрибутиве Linux отсутствует папка `/usr/lib/systemd/system/`, то все изменения вносим в папку `/lib/systemd/system/`.

- Прописываем сервис в автозапуск, выполнив команду:

```
systemctl enable dpc-qmobile-demo.service
```

- Запускаем сервис, выполнив команду:

```
systemctl start dpc-qmobile-demo.service
```

14.3.3. Установка демо-сайта (с использованием Kubernetes)

1. Скачать архив qp8-product-catalog-demo-manifests.tar.gz и распаковать его содержимое в `/etc/dpc-demo-config.`
2. Перейти в папку `/etc/dpc-demo-config/kubernetes`
3. В файле `k8s.yaml` необходимо задать значения:
   - в качестве значения DPC_HOST нужно указать внешний URL компонента **DPC.SearchApi**. URL должен быть доступен из браузера пользователя демо-сайта.
   - в качестве значения IMPACT_HOST нужно указать внешний URL компонента **DPC.Impact** из модуля **Impact**. URL должен быть доступен из браузера пользователя демо-сайта.
   - в качестве значения DOWNLOAD_PDF_HOST нужно указать внешний URL компонента **DPC.PdfServer** из модуля **PdfGenerator**. URL должен быть доступен из браузера пользователя демо-сайта.
   - в ingress в качестве хоста вместо `dpc-qmobile-demo.test` нужно задать внешний DNS демо-сайта, по которому к нему сможет обращаться пользователь.
4. Применить манифест:

```
kubectl apply -f k8s.yaml
```

14.3.4. Настройка nginx

Для корректной работы `DPC.Admin` необходимо решить вопрос безопасности, связанный с CORS-ограничениями. Для их обхода следует делать запросы на тот же домен, на котором располагается QP, осуществить это можно с помощью nginx-а.

Необходимые секции представлены в файле `/dpc-config/nginx/nginx.conf`. Детально ознакомится с настройкой nginx, а так же общей механикой – можно по этой ссылке.

Nginx с использованием docker

Если nginx при установке QP8.CMS был запущен в docker, то следует:

6) выполнить команду `docker-compose down`;
7) открыть на редактирование файл nginx.conf предположительно расположенный по пути `/etc/qpconfig/nginx/nginx.conf`;
8) в конфигурационный файл добавить секции из файла `/etc/dpc-demo-config/nginx/nginx.conf`;
9) задать свои DNS вместо заданных в домене `.test`
10) выполнить команду `docker-compose up –d`.

Nginx без использования docker

Если nginx при установке QP8.CMS был запущен не в docker, то следует:

1) открыть на редактирование файл nginx.conf предположительно расположенный по пути `/etc/nginx/nginx.conf`;

2) в конфигурационный файл добавить секции из файла `/home/qp/dpc-demo-config/nginx/nginx.conf`;
3) задать свои DNS вместо заданных в домене `.test`;
4) после чего проверить корректность конфигурации командой `nginx -t`;
5) если всё корректно, то перечитать конфигурацию nginx командой `nginx -s reload`.