ELMA

Интеграция системы ELMA с внешними системами

Краткое руководство

Оглавление

Введение

В рамках данного краткого руководства описываются основные возможности, которые предоставляет **Платформа ELMA BPM** для интеграции с другими системами. В данной книге представлены примеры наиболее распространенных на практике способов реализации обмена данными с системой ELMA. Данный список не является исчерпывающим, так как система может реализовать любой нетиповой протокол обмена данными, используя всю широту возможностей .NET.

Основная задача данного краткого руководства – сформировать понимание реализации интеграции систем на наиболее распространенных примерах, показать типовые проблемы и способы их решения, предоставить готовые к адаптации и переиспользованию примеры.

Описание способов интеграции в данном руководстве приведено от простого к сложному, с целью сформировать понимание у тех пользователей, которые ранее не сталкивались с подобными задачами. При этом важно отметить, что наиболее частые и распространенные на практике способы интеграции представлены в **Глава 4** (передача данных через веб-сервисы со стороны внешней системы в ELMA) и **Глава 7** (передача данных из ELMA во внешнюю систему).

Данная книга предназначена для тех пользователей, которые имеют базовые знания и опыт в написании кода и планируют самостоятельно интегрировать систему ELMA с другими системами. Также предполагается, что пользователь уже знаком с архитектурой системы ELMA и владеет базовыми навыками работы с ней, которые описаны в кратком руководстве по Платформе ELMA BPM.

Полное описание функционала системы ELMA приведено в справке по системе. Справка входит в поставку системы, а также всегда доступна в Базе знаний ELMA: http://www.elma-bpm.ru/kb/help/.

Решения многих технических вопросов описаны в Базе знаний ELMA по адресу http://www.elma-bpm.ru/kb/. База знаний постоянно пополняется специалистами компании.

Глава 1. Использование системы ELMA в инфраструктуре компании

1.1. Общие сведения об интеграции

Как правило, все используемые в компании корпоративные системы тесно взаимосвязаны друг с другом. Например, используемая в интернет-магазине программа не будет работать эффективно без взаимодействия установленной на складе с программой, а мобильный календарь – без синхронизации с сервером расписаний. Иными словами, для максимально продуктивной работы требуется непрерывный обмен данными между приложениями на различных устройствах.

Эту проблему эффективно решает интеграция.

Рис. 1. Интеграция между системами

Под **интеграцией** следует понимать двусторонний обмен данными между системами. В нашем случае обменом данными будем считать взаимодействие между системой ELMA и различными внешними системами: сервисами, веб-сайтами, готовыми программными продуктами и базами данных.

Обмен данными осуществляется по определенным каналам связи в виде обмена так называемыми сообщениями. В качестве сообщений могут выступать веб-запросы, файлы (документы), поток данных.

Интеграция служит неотъемлемой частью любой системы, которая «планирует развиваться», так как с чем большим количеством систем она сможет работать, тем более универсальной и востребованной она становится.

Приведем пример. Система ELMA не всегда имела механизм интеграции с системой «1С: Предприятие». Однако наличие данного механизма открывало перед нами новые возможности и новых заказчиков, которые ранее работали только с 1С и хотели бы продолжать использовать эту систему, но в силу некоторых факторов нуждались в системе ELMA. Таким образом, появление дополнительного механизма интеграции позволило выйти системе ELMA на новый уровень.

1.2. Когда требуется интеграция?

Не стоит забывать, что процесс интеграции с любой внешней системой является очень трудоемким и перед тем, как приступить к интеграции, необходимо ответить на ряд вопросов. Так ли необходима интеграция? На сколько это целесообразно? Можно ли ее как-то избежать?

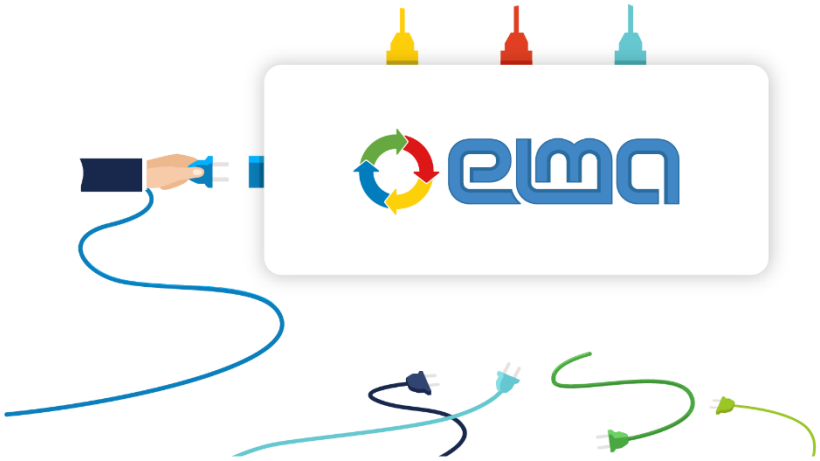


Рис. 2. Сложный процесс интеграции с внешними системами

Ниже приведены основные доводы, которыми стоит руководствоваться перед принятием решения об интеграции системы ELMA с другой системой.

1. Для чего нужна интеграция?

Что будет являться результатом интеграции с системой ELMA и есть ли возможность ее избежать? Возможно, уже есть практика решения стоящих перед системой ELMA задач и необходимость в интеграции отпадает, либо подобная интеграция уже разработана и описана, что сводит повторение интеграции лишь к привидению к системе ELMA.

2. Является ли результат интеграции разовой потребностью или будет использоваться постоянно/периодически?

Перед любой интеграцией следует определить потребность в ней, а именно, будет ли потребность разовой, периодической или же постоянной. В зависимости от этого стоит принимать решении о ее необходимости. Конечно, согласиться на интеграцию можно при любом ее использовании, решающим фактором здесь выступит результат следующего пункта.

3. Определите соотношение затрат к результату.

В данном пункте раскрывается именно экономическая составляющая интеграции, которая должна дать понять, насколько эффективна разница затраченных на интеграцию ресурсов к полученной от нее выгоде.

Например, ради нескольких отчетов можно затратить колоссальные усилия на разработку решения интеграции и огромные ресурсы (человеческие/денежные/машинные) на ее реализацию, а на выходе получить интеграцию с некой самописной программой. Естественно, в данном случае и речи быть не может об интеграции, т.к. затраты многократно превышают потенциальный доход.

Отдельно можно выделить интеграцию с легаси-системами (устаревшими), которые еще используются, но в перспективе будут заменены на более современные. Как правило, трудоемкость интеграции с такими системами выше среднего, а использование ограничено. В таких случаях бывает целесообразно применить RPA-подходы (обмен данными с использованием роботов) для быстрой интеграции.

Глава 2. Подход к интеграции ИТ-систем

2.1. Общие требования к интеграции

Проектирование любых интеграционных решений далеко не тривиальная задача, и разработка модели интеграции является весьма ответственным занятием. Убедившись в необходимости интеграции и определившись с ее целью, разработчик интеграционного решения, как правило, может столкнуться со следующими «вызовами»:

1. **Ненадежность сети передачи данных.** Все интеграционные решения предполагают передачу информации между устройствами. В отличие от процессов, выполняющихся в пределах одного компьютера, распределенной вычислительной среде присущ целый ряд недостатков. Зачастую общающиеся системы разделены континентами, что вынуждает передавать данные по телефонным линиям, сегментам локальных сетей, через маршрутизаторы, коммутаторы, общедоступные сети и спутниковые каналы связи. Доставка информации на каждом из этих этапов связана с задержкой и риском потери.
2. **Низкая скорость передачи данных.** Время доставки данных через компьютерную сеть на порядок больше времени вызова локального метода. Таким образом, создание распределенного приложения требует применения иных принципов проектирования, чем создание приложения, выполняющегося в пределах одного компьютера.
3. **Различия между приложениями.** Интеграционное решение должно учитывать все различия (язык программирования, платформу, формат данных), существующие между объединяемыми системами.
4. **Неизбежность изменений.** Интеграционное решение должно иметь возможность адаптации к изменению объединяемых им приложений. Зачастую преобразования в одной системе влекут за собой непредсказуемые последствия для других систем. Поэтому при интеграции приложений важно уменьшить их взаимозависимость за счет так называемого слабого связывания.

Для преодоления описанных выше трудностей можно воспользоваться четырьмя основными подходами.

1. **Передача файла.** Одно приложение создает файл, а другое приложение считывает его. Приложения должны согласовать имя файла, его расположение, формат, время записи/считывания, а также процедуру удаления.

2. **Общая база данных.** Несколько приложений используют общую логическую структуру данных, которой соответствует одна физическая база данных. Наличие единого хранилища данных устраняет проблему передачи информации между приложениями.
3. **Удаленный вызов процедуры.** Приложение предоставляет доступ к части своей функциональности посредством удаленного вызова процедуры. Взаимодействие между приложениями осуществляется синхронно в режиме реального времени.
4. **Обмен сообщениями.** Приложение размещает сообщение в общем канале, которое затем считывается другим приложением. Приложения должны согласовать канал, а также формат сообщения. Взаимодействие между приложениями осуществляется в асинхронном режиме.

Каждый из предложенных подходов имеет собственные преимущества и недостатки. На практике приложения могут быть интегрированы несколькими способами таким образом, чтобы использовать только сильные стороны того или иного подхода.

Глава 3. Запуск бизнес-процессов ELMA с веб-страницы

Для организации взаимодействия пользователя с сетью Интернет, разработчик сайта должен предусмотреть передачу запросов от пользователя сайта на сервер, где расположен данный сайт. Запросы бывают двух видов: GET и POST-запросы.

3.1. GET-запросы

3.1.1. О GET-запросах и их применимости

GET-запросы представляют собой передачу данных непосредственно в адресной строке браузера и имеют следующий синтаксис: **http://домен/страница?[параметр1=значение1][&параметр2=значение2]...**

Набор передаваемых на сервер данных начинается с символа «?» и разделяется символом «&», а сами данные представляют собой пары **«параметр»=«значение»**. В GET-запросах рекомендуется передавать исключительно служебную информацию в виде чисел и слов с использованием букв латинского алфавита.

Например, если требуется передать имя и фамилию пользователя на странице регистрации (например, **reg.html** сайта **http://elma.someorg.ru**), то это будет выглядеть так: **http://elma.someorg.ru/reg.html?fname=Иван&lname=Иванов** Стоит отметить, что для корректной обработки кириллицы и дальнейшей передачи символов русского алфавита рекомендуется использовать последнюю версию веб-браузера.

GET-запросы имеют несколько недостатков:

1. Ограниченность передаваемых данных. На стороне сервера строка запроса ограничивается заданным максимальным значением. Например, если максимальный размер запроса может составлять 1024 символа, то все, что превышает данное значение, будет удалено. Следовательно, часть передаваемой информации не будет обработана указанной страницей сайта.
2. Возможность передачи строго определенных наборов символов. Например, символы «?» и «&» уже зарезервированы и передавать их как значения параметров нельзя. Однако это правило можно обойти, если в строке запроса передавать не сам символ, а его кодовое значение. Для этого используется символ «%», за которым следует код символа. Например, так: **http://elma.someorg.ru/reg.html?fname=%CC%DF%AD%1F%DS&lname=%DD** В

данном случае с целью экономии длины запроса кодовые значения указаны в шестнадцатеричном виде.

Несмотря на указанные недостатки, создание сайтов без GET-запросов было бы крайне затруднительным. Например, они незаменимы в случаях начальной инициализации страницы сайта для определенного пользователя, когда в запросе указывается не только сайт и текущая страница, но и id пользователя. Например, так было ранее сделано в социальной сети Вконтакте: **http://vk.com/profile.php?id=01234567**

Также GET-запросы часто применяются при регистрации пользователя для проверки корректности указанного e-mail адреса. В этом случае на указанный пользователем e-mail приходит письмо со ссылкой активации, которая представляет собой GET-запрос.

Использование GET-запроса – простейший способ запуска бизнес-процесса из внешней системы. Все, что нужно для создания такого запроса – это составить ссылку вида (Рис. 3):

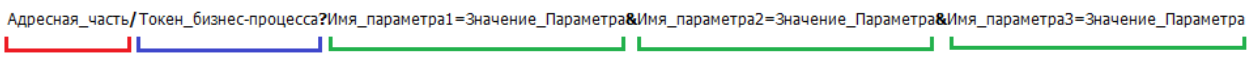


Рис. 3. Схема GET-запроса, при помощи которого выполняется запуск бизнес-процесса ELMA

К адресу сервера добавляется стандартный путь **/Processes/ProcessHeader/RunByWebQuery/**. После последнего символа **/** указываются параметры запроса. Первое, что требуется указать – это токен бизнес-процесса. По сути, правильно указанные адресная часть и токен уже позволяют использовать эту ссылку для запуска бизнес-процесса. В этом случае он будет запущен со стандартными, определенными по умолчанию параметрами. То есть результат будет такой же, как при запуске процесса через веб-приложение системы ELMA.

Если же требуется указать некоторые дополнительные значения параметров бизнес-процесса, то их нужно перечислить в тексте ссылки после токена. Токен и параметры в ссылке отделяются вопросительным знаком (?); между собой параметры отделяются знаком амперсанда (&). Параметр указывается в следующем формате: **Имя_параметра=Значение_параметра**.

Полученная ссылка может быть запущена с любого устройства. В простейшем случае, ее достаточно вручную ввести в адресной строке веб-браузера компьютера.

3.1.2. Описание примера

Разберем небольшой пример использования GET-запроса для запуска процесса с передачей нескольких параметров.

Компания, специализирующаяся на доставке, имеется бизнес-процесс «Оформление заказа». Суть данного процесса заключается в получении и дальнейшей обработке информации о заказе, который оставил посетитель сайта. Необходимо организовать работу таким образом, чтобы вся информация с сайта попала в систему, запустив при этом бизнес-процесс «Оформление заказа».

3.1.3.1. Реализация

Составим простой бизнес-процесс (Рис. 4), состоящий из начального и конечного событий, а также пользовательской задачи для представления результата.

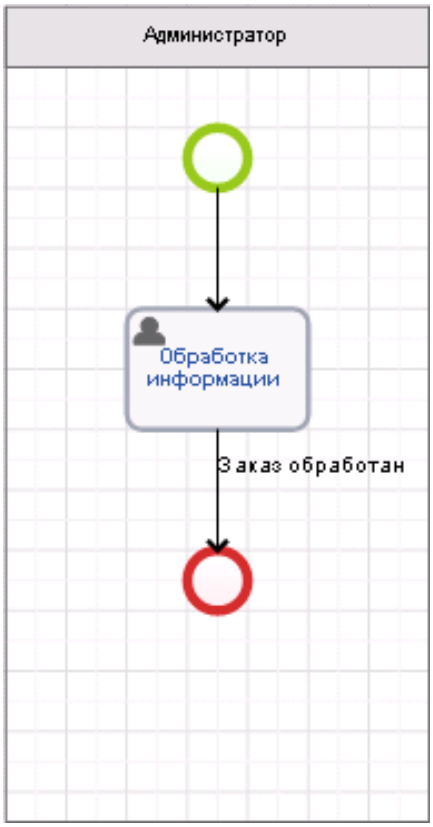


Рис. 4. Бизнес-процесс «Оформление заказа»

Также необходимо задать контекст процесса (Рис. 5) в виде трех переменных строкового типа: **Name**, **Contact** и **Adress**. В эти переменные будут переданы значения с формы сайта.

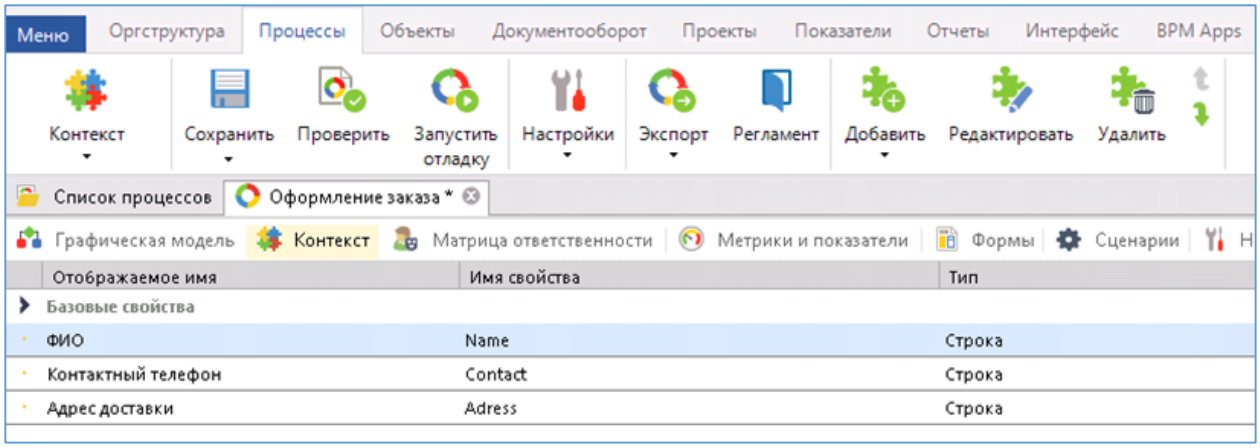


Рис. 5. Контекстные переменные бизнес-процесса

Опубликуем процесс (Рис. 6) с генерацией токена и установленным флажком **Веб-запрос (GET)**.

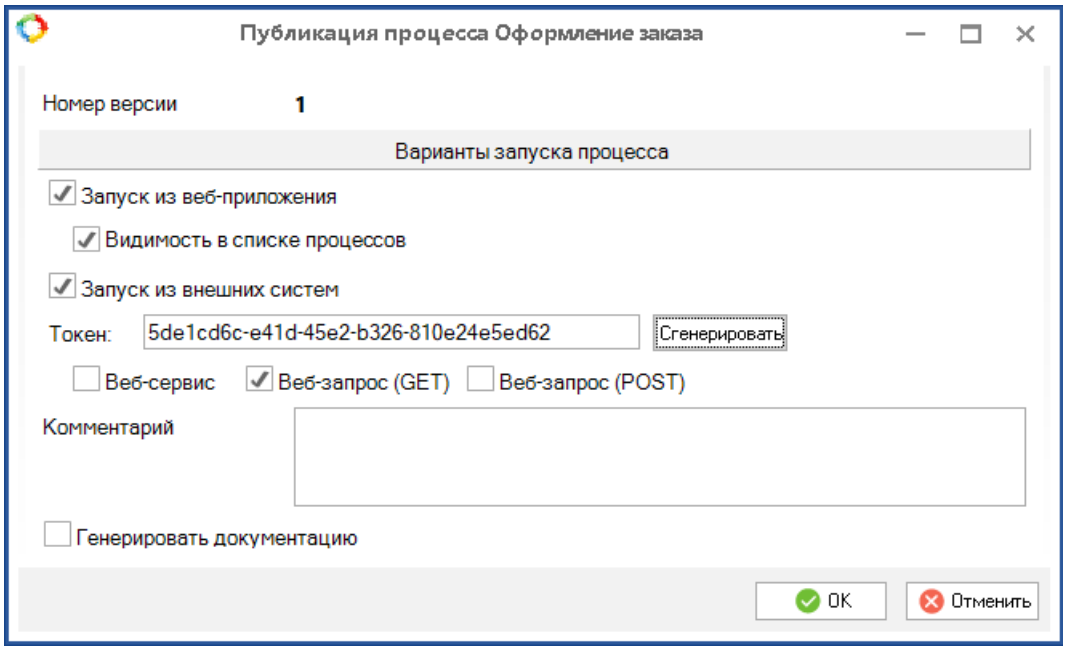


Рис. 6. Диалоговое окно настройки бизнес-процесса при публикации

Для запуска процесса с помощью GET-запроса создадим HTML-форму, в которой пользователь будет заполнять параметры перед запуском процесса.

Пример кода формы:

```
<html>
<head>
<meta charset="utf-8">
</head>
<form ACTION="http://127.0.0.1:8000/Processes/ProcessHeader/RunByWebQuery/5de1cd6c-e41d-45e2-
b326-810e24e5ed62" method="GET">
    <table width="60%" border="0" cellspacing="1" cellpadding="4">
        <tr>
```

```
            <td><b>ФИО:</b></td>
            <td><input type="text" name="Name" size="100"></td>
        </tr>
        <tr>
            <td><b>Адрес:</b></td>
            <td><input type="text" name="Adress" size="100"></td>
        </tr>
        <tr>
            <td><b>Контактный телефон:</b></td>
            <td><input type="text" name="Contact" size="15"></td>
        </tr>
        <tr>
            <td><input type="Submit" value="Отправить" name="Run_Workflow"></td>
        </tr>
    </table>
</form>
</html>
```

При нажатии на кнопку **Отправить** (Рис. 7) на сервер будет отправлен GET-запрос, который запустит процесс (Рис. 8) с соответствующим токеном и передаст ему ряд параметров.

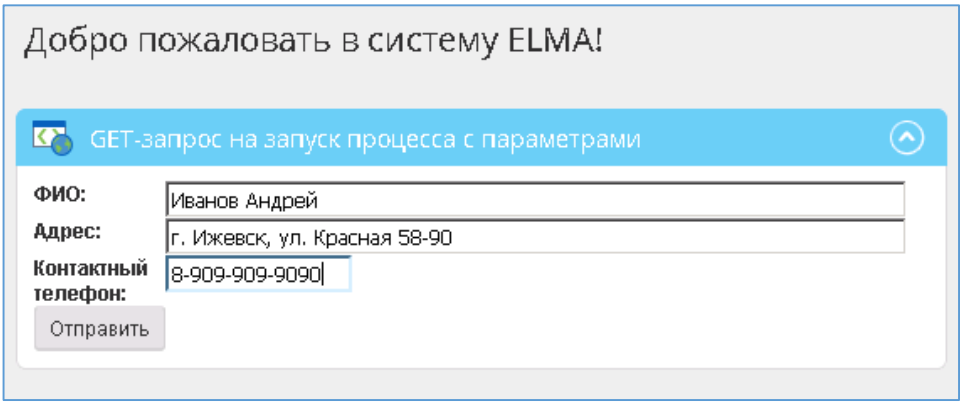


Рис. 7. Портлет для запуска бизнес-процесса с помощью GET-запроса

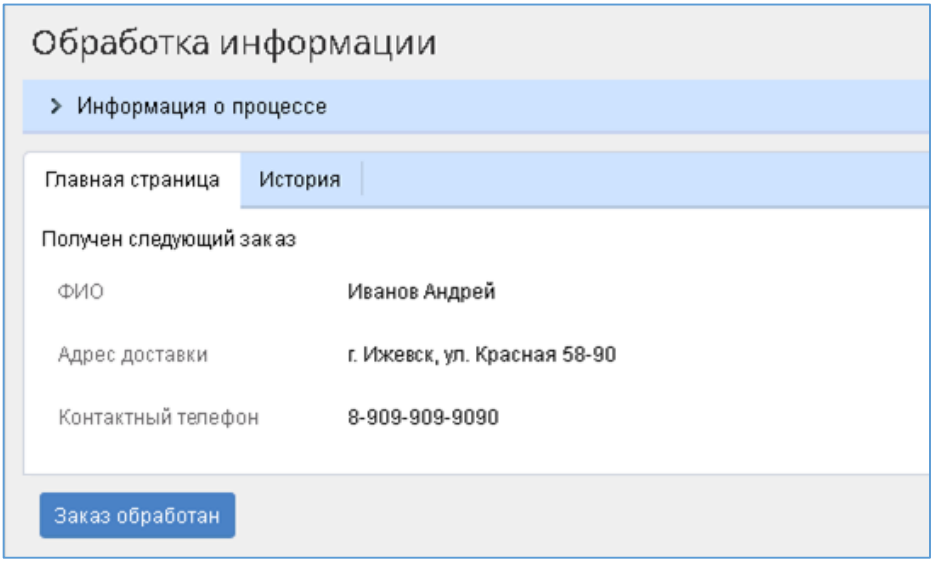


Рис. 8. Задача «Обработка информации». Результат выполнения GET-запроса с параметрами

Ключевым недостатком запуска процесса по GET-запросу напрямую с формы можно назвать необходимость авторизации в системе ELMA. То есть реализовать данные запросы внутри системы можно без каких-либо ограничений, но, если предполагается запуск с внешнего сайта, то пользователю потребуется ввести логин и пароль в системе ELMA.

3.2. POST-запросы

3.2.1. О POST-запросах и их применимости

POST-запрос по своей структуре сложнее GET-запроса: он имеет заголовок и тело. Нельзя передать POST-запрос через адресную строку браузера – требуется специализированный механизм отправки запросов.

POST-запрос является наиболее предпочтительным при необходимости отправки в бизнес-процесс большого количества параметров. При помощи POST-запроса можно передать неограниченное количество параметров. Максимальное количество параметров, которое можно передавать GET-запросом, ограничено максимальной длиной заголовка запроса – от двух тысяч символов (в зависимости от используемых веб-сервера и веб-клиента). В большинстве случаев этого более чем достаточно, но, если требуется, например, передать в переменную некоторый текст, этого лимита может не хватить.

Кроме того, HTTP-POST-запросы не имеют ограничений на тип передаваемых параметров. Возможность использования POST-запросов будет полезна при интеграции системы ELMA с простыми веб-приложениями в случае, если требуется передавать в систему ELMA большой объем данных или файлы.

Рассмотрим на примере передачу параметров с помощью POST-запроса.

3.2.2. Описание примера

В качестве примера будем использовать ранее рассмотренный процесс «Оформление заказа» с небольшим исключением – будем использовать метод POST и добавим передачу файла в процесс.

3.1.3.2. Реализация

Добавим к уже существующим параметрам процесса еще один параметр – **Вложение** типа «Вложение (Объект)» (Рис. 9).

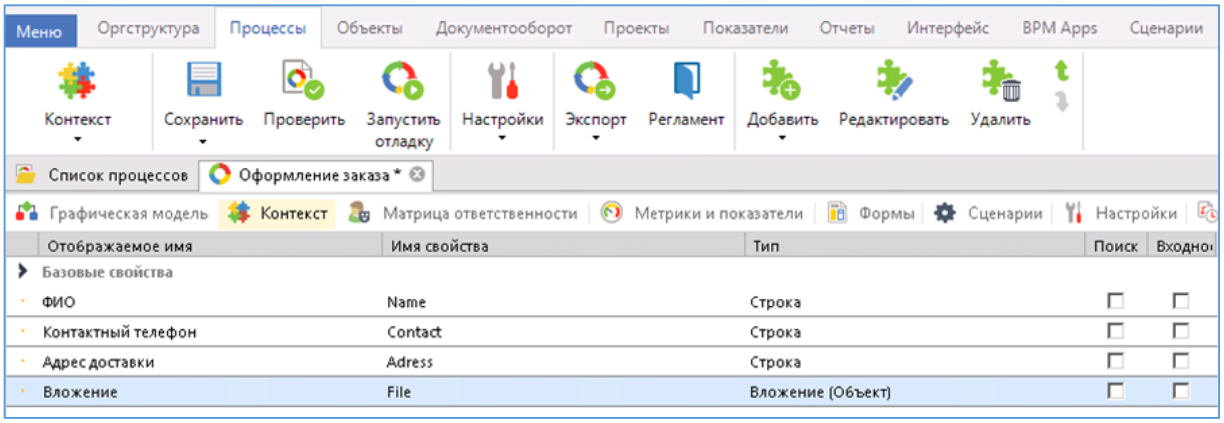


Рис. 9. Контекстные переменные бизнес-процесса «Оформление заказа»

Опубликуем процесс (Рис. 10) с уже существующим токеном, установив флажок на параметре **Веб-запрос (POST)**.

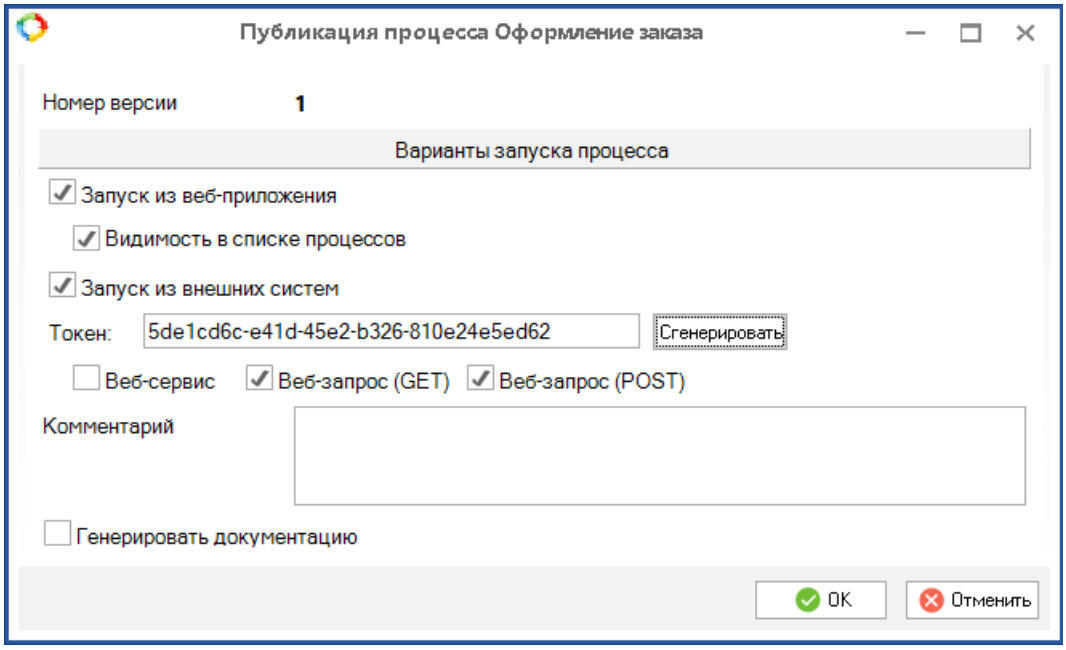


Рис. 10. Настройки бизнес-процесса при публикации

Для запуска процесса с помощью POST-запроса воспользуемся уже существующей формой, внеся ряд изменений:

1. В теге «form» изменим значение параметра «method» с GET на POST.
2. Добавим в тег «form» еще один параметр «enctype="multipart/form-data"».
3. Добавим на форму следующую конструкцию:

```
<tr>
    <td><b>Вложение:</b></td>
    <td><input type="file" name="File.File" size="15"></td>
</tr>
```

Здесь стоит отметить один нюанс: если переменная в контексте процесса является типом «Вложение» (как в нашем случае), то необходимо добавить через точку параметр «File». Если же используется тип «File», то дополнение не понадобится.

Пример кода формы:

```
<html>
<head>
    <meta charset="utf-8">
</head>
<form enctype="multipart/form-data"
ACTION="http://127.0.0.1:8000/Processes/ProcessHeader/RunByWebQuery/5de1cd6c-e41d-45e2-b326-
810e24e5ed62" method="POST">
    <table width="60%" border="0" cellspacing="1" cellpadding="4">
        <tr>
            <td><b>ФИО:</b></td>
            <td><input type="text" name="Name" size="100"></td>
        </tr>
        <tr>
            <td><b>Адрес:</b></td>
            <td><input type="text" name="Adress" size="100"></td>
        </tr>
        <tr>
            <td><b>Контактный телефон:</b></td>
            <td><input type="text" name="Contact" size="15"></td>
        </tr>
        <tr>
            <td><b>Вложение:</b></td>
            <td><input type="file" name="File.File" size="15"></td>
        </tr>
        <tr>
            <td><input type="Submit" value="Отправить" name="Run_Workflow"></td>
        </tr>
    </table>
</form>
</html>
```

При нажатии на кнопку **Отправить** (Рис. 11) на сервер будет отправлен POST-запрос, который запустит процесс с соответствующим токеном и передаст ему ряд параметров, включая файл (Рис. 12).

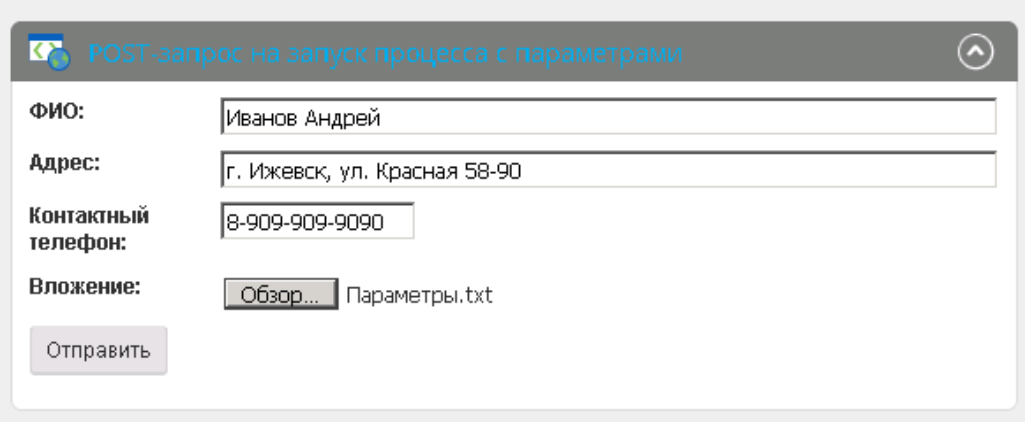


Рис. 11. Портлет для запуска бизнес-процесса с помощью GET-запроса

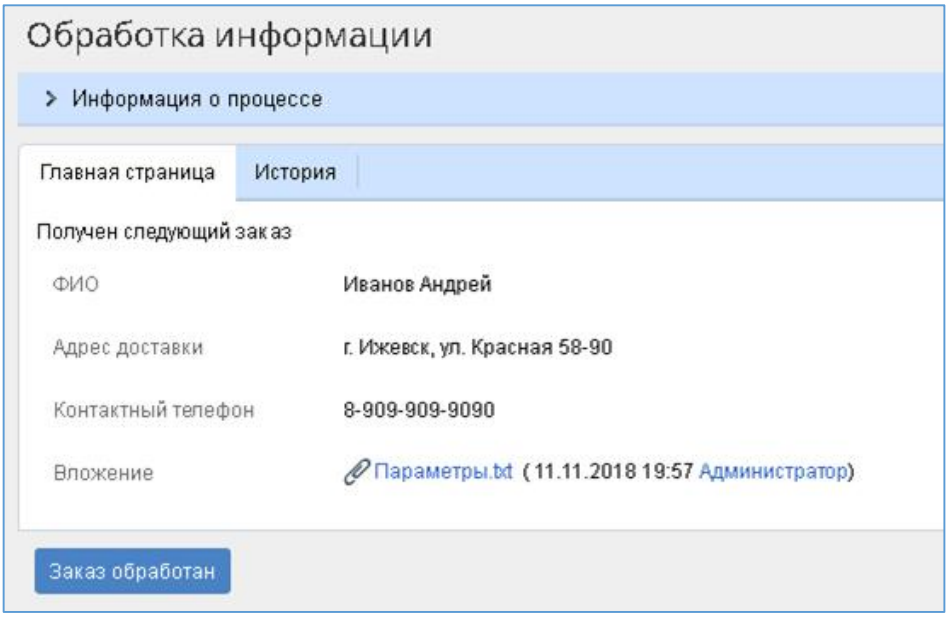


Рис. 12. «Обработка информации» – результат выполнения POST-запроса с параметрами

Глава 4. Запуск бизнес-процессов ELMA из внешних систем

В предыдущих разделах мы рассмотрели возможности запуска процесса с передачей параметров посредством POST и GET-запросов. Однако нам не всегда нужно передавать какие-либо данные при запуске процесса. На практике часто возникает потребность обмениваться информацией в ходе уже запущенного бизнес-процесса. Так, например, сторонняя программа/сайт может запросить от системы ELMA список текущих контрагентов, существующих в системе, а также внести в них какие-либо изменения и вернуть их в систему ELMA.

Для такого взаимодействия в системе ELMA существует справка по публичным веб-службам ELMA. Данная справка всегда доступна по ссылке **http://<адрес сервера ELMA>:<порт>/Api/Help**, а также в Базе знаний ELMA.

В системе ELMA описаны публичные сервисы системы с помощью языка описания веб-сервисов WSDL и архитектурного стиля REST API. Далее коротко дадим определение, разберем на одном и том же примере оба подхода, а также сделаем выводы.

4.1. Архитектурный стиль REST

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем (World Wide Web (WWW)), который, как правило, используется для построения веб-служб. В общем случае REST является очень простым интерфейсом управления информацией без использования каких-либо дополнительных внутренних прослоек.

Управление информацией сервиса целиком и полностью основывается на протоколе передачи данных. Наиболее распространенным протоколом является HTTP, для которого действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Перед тем, как переходить к рассмотрению примера, необходимо понимать, что для работы с системой ELMA извне необходимо иметь токены доверенных внешних приложений (**ApplicationToken**) и токены сессии (**SessionToken**). Токен внешнего приложения требуется для идентификации доверенного приложения (ELMA for IPad, ELMA Agent, Outlook и др.) и имеет срок истечения. При первой авторизации

необходимо обязательно передавать в заголовке токен приложения: ApplicationToken. Подробнее о токене см. в статье Базы знаний ELMA.

Настройка токенов внешних приложений осуществляется в веб-приложении ELMA в разделе ***Администрирование – Система – Внешние приложения***. При этом есть ряд системных токенов, срок истечения которых устанавливается отдельно для каждого приложения.

Токен сессии – уникальный ключ, используемый для авторизации приложения в системе ELMA. В данном случае «сессия» обозначает уникальную пару «Приложение + Пользователь». Токен сессии требуется для идентификации именно этой пары на сервере. В случае, если в приложении для текущего пользователя уже есть токен сессии, то дополнительно при первой авторизации (по логину и паролю) необходимо также передавать его в заголовке **SessionToken**. Это необходимо для обеспечения отслеживания изменений на сервере в связке с клиентом.

4.2. Создание нового внешнего приложения и токена

Создадим внешнее приложение и токен для него. Для этого нужно перейти в раздел ***Администрирование – Система – Внешние приложения*** и нажать на кнопку **Добавить** (Рис. 13).

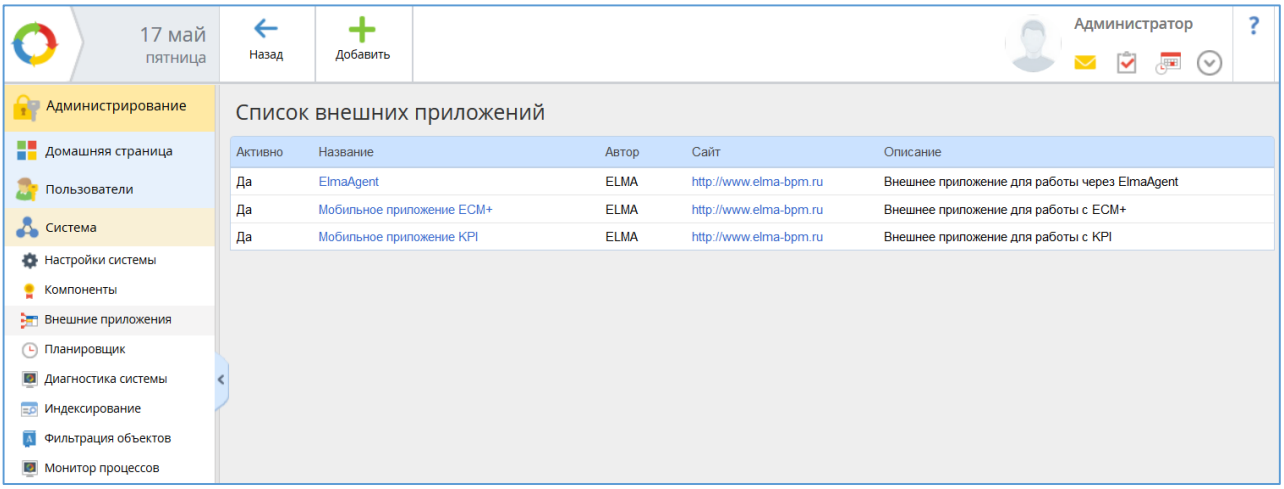


Рис. 13. Раздел «Администрирование – Система – Внешние приложения»

Далее заполняем требуемые для нашего приложения поля (Рис. 14) и нажимаем на кнопку **Сохранить**.

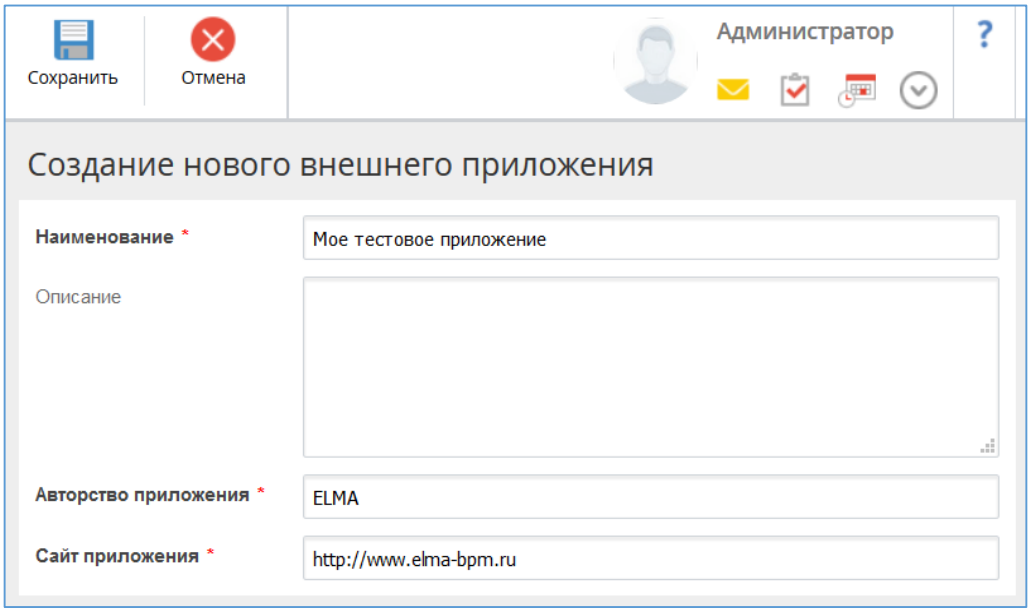


Рис. 14. Форма создания нового внешнего приложения

После этого необходимо перейти в созданное приложение и включить его нажатием на кнопку **Включить** (Рис. 15).

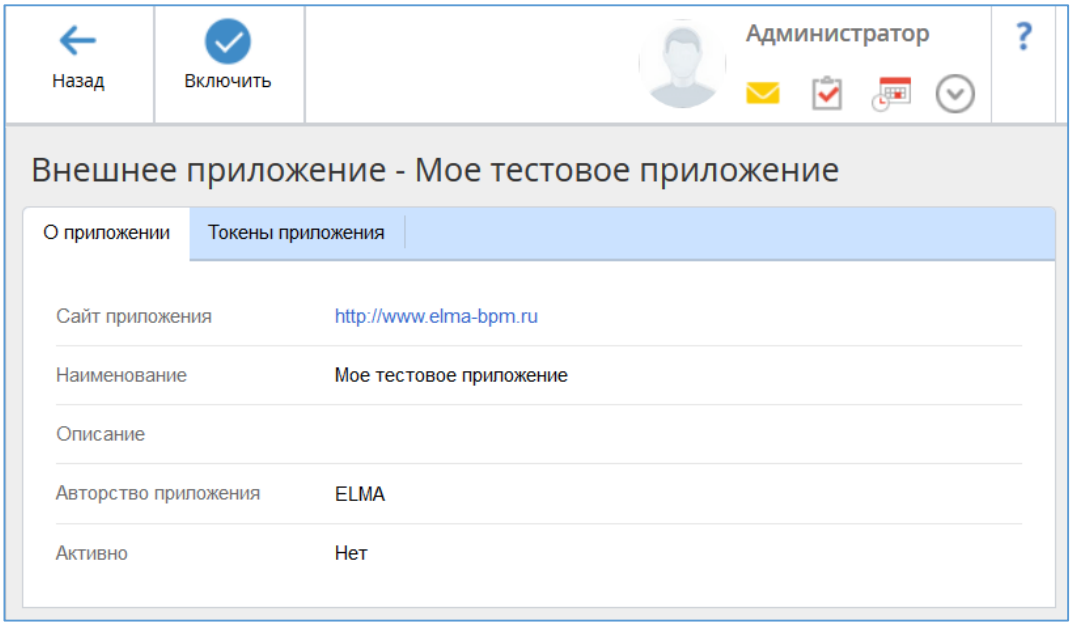


Рис. 15. Карточка внешнего приложения. Вкладка «О приложении»

Теперь можно добавить токен. Для этого переходим на вкладку **Токены приложения** и нажимаем на кнопку **Добавить токен** (Рис. 16).

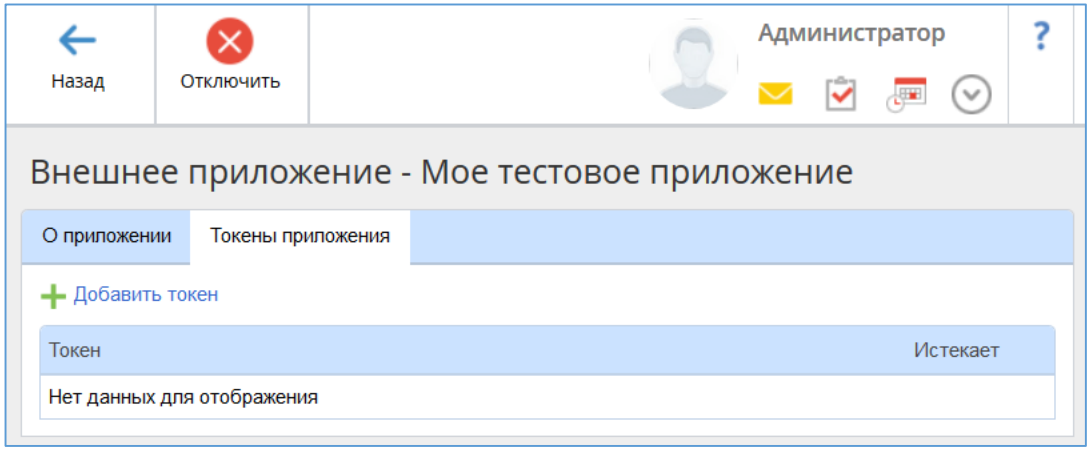


Рис. 16. Карточка внешнего приложения. Вкладка «Токены приложения»

В открывшемся диалоговом окне (Рис. 17) необходимо выбрать дату истечения нового токена и нажать на кнопку **Создать токен**.

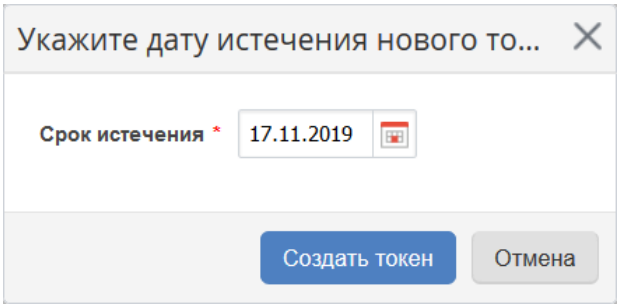


Рис. 17. Диалоговое окно выбора срока истечения токена

Новое внешнее приложение готово, теперь с ним можно работать.

4.3. Пример подключения к REST API системы ELMA

Условно наш пример можно разделить на 3 этапа:

1. Авторизация в системе.
2. Получение данных.
3. Изменение данных.

Для работы создадим консольное приложение в Visual Studio и подключим следующие пространства имен:

```
using System.Net;
using System.IO;
using Newtonsoft.JSON;
```

Для последнего пространства имен необходимо добавить в решение пакет Newtonsoft.JSON через Сервис – Диспетчер пакетов NuGET – Управление пакетами NuGET для решения.

В ходе примера мы авторизуемся в системе, после чего получим данные о искомой сущности (платежном поступлении) по ее типу и идентификатору в системе. После этого изменим статус платежного поступления на «получено». Данный пример подходит для проставления статуса оплаты по сделкам контрагентов в системе ELMA из некоторой платежной системы, получившей денежные средства по данной сделке.

Рассмотрим в качестве примера контрагента «ООО РИК» (Рис. 18). Данный контрагент имеет единственную сделку «Поставка оборудования» с объёмом продаж 207000,00 условных единиц.

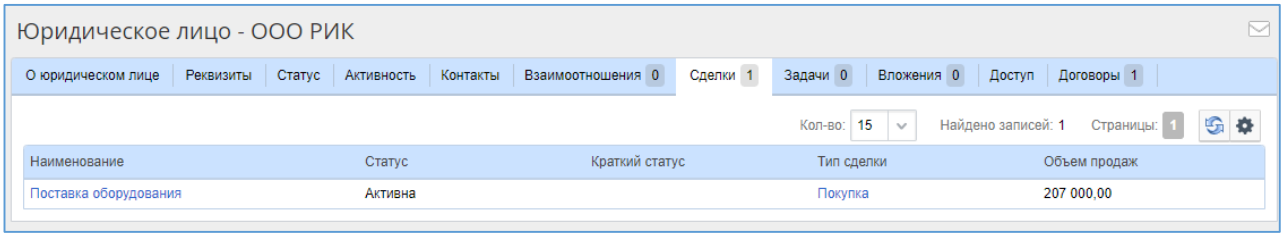


Рис. 18. Карточка контрагента. Вкладка "Сделки"

Перейдем непосредственно к карточке данной сделки (Рис. 19). Сделка содержит в себе три поступления: «Орг. техника и комплектующие», «Партия ноутбуков», «Поставка и установка измерительной техники».

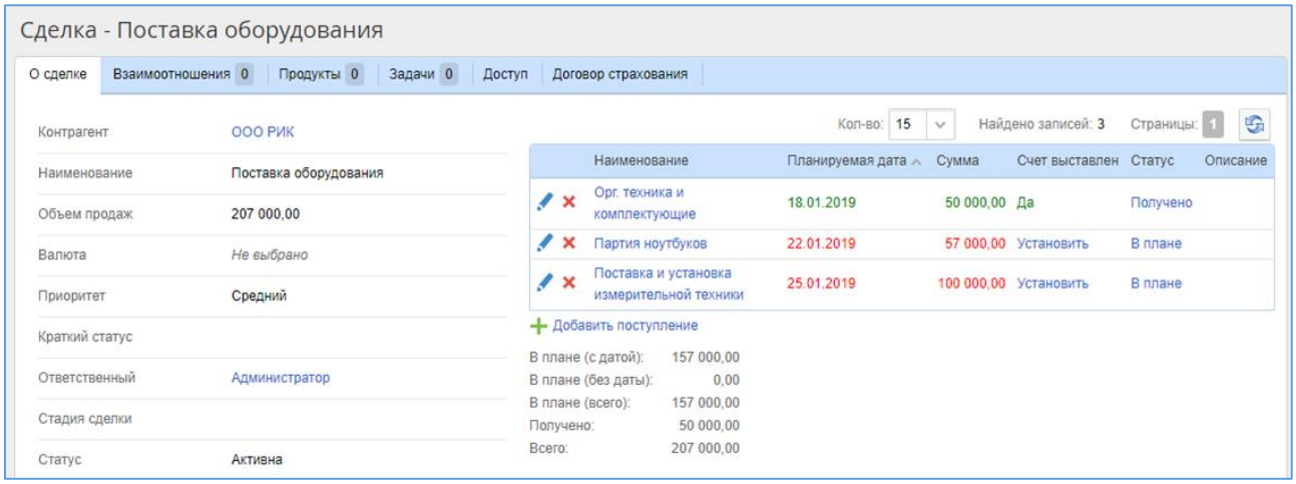


Рис. 19. Карточка сделки. Вкладка «О сделке»

Поступление «Орг. техника и комплектующие» имеет статус «Получено», а остальные два – «В плане».

Статусы всех поступлений можно изменять вручную (при наличии определенных прав доступа). Однако возникает вопрос: как узнать, в какой момент времени следует изменить статус поступления на «Получено»? Безусловно, статус меняется при поступлении денежных средств, однако система ELMA не является системой бухгалтерского учета и не получает подобных уведомлений. Получается, что оповещение о поступлении средств получает сторонняя система, которая может отправить сообщение о поступлении таких средств в систему ELMA. Именно эта возможность и рассматривается в данном примере, где в качестве «сторонней системы» выступает консольное приложение.

4.3.1. Авторизация в системе

Первое, что необходимо сделать перед началом работы – пройти авторизацию на сервере. Для этого переходим в справку по публичным веб-службам ELMA (**http://<адрес сервера ELMA>:<порт>/Api/Help**) и выбираем **Список доступных публичных веб-сервисов** (Рис. 20).

Список доступных сервисов системы

IAuthorizationService	Сервис для авторизации пользователей • Перейти к описанию WSDL • Перейти к описанию REST запросов к методам
IBatchOperationService	Сервис для получения записей сущностей • Перейти к описанию WSDL • Перейти к описанию REST запросов к методам
IEntityChangesService	Сервис для отслеживания изменений сущностей • Перейти к описанию WSDL • Перейти к описанию REST запросов к методам

Рис. 20. Список доступных публичных веб-сервисов системы

В данном списке необходимо выбрать Сервис для авторизации пользователей – Перейти к описанию REST запросов к методам (Рис. 21).

Операции в http://localhost:8000/API/REST/Authorization

Данная страница описывает операции службы в этой конечной точке.

Uri	Метод	Описание
/ApiVersion	GET	Получить текущую версию API сервера
/CheckPermissions	POST	Проверить привилегии
/CheckToken	GET	Проверить токен авторизации и получить ИД пользователя
/LoginWith	GET	Авторизовать пользователя по протоколу Basic
	POST	Авторизовать пользователя по логину и паролю
/LoginWithSSPI	POST	Авторизовать пользователя по тикету сквозной авторизации SSPI
/ServerTime	GET	Получить текущее серверное время
/ServerTimeUTC	GET	Получить текущее серверное время в формате UTC
/TemporaryGuid	GET	Получить временный токен авторизации. Можно использовать его в запросе как параметр TemporaryGuid
/Version	GET	Получить текущую версию сервера

Рис. 21. Сервис для авторизации пользователей. REST запросы к методам

В данной таблице выберем вариант **/LoginWith (POST)** – данный метод основан на авторизации по паре login/password.

На странице метода (Рис. 22) отображается строка для авторизации **http://localhost:8000/API/REST/Authorization/LoginWith?username={USERNAME}**, где **localhost:8000** – это локальный адрес нашего сервера. Вместо **{USERNAME}** необходимо будет передать логин пользователя.

Авторизовать пользователя по логину и паролю

Url: http://localhost:8000/API/REST/Authorization/LoginWith?username={USERNAME}

HTTP Метод: POST

Направление сообщения	Формат	Текст сообщения
Запрос	Xml	Пример
Запрос	Json	Пример
Ответ	Xml	Пример,Схема
Ответ	Json	Пример

Рис. 22. Строка для авторизации пользователя по логину и паролю

Данный метод дает возможность работать как с XML-форматом, так и с JSON.

Из-за удобства и простоты формата будем работать с JSON. Ниже приведен пример текста ответа для данного формата (Рис. 23):

```
{
        "AuthToken":"1627aea5-8e0a-4371-9022-9b504344e724",
        "CurrentUserId":"Строковое содержимое",
        "Lang":"Строковое содержимое",
        "SessionToken":"Строковое содержимое"
}
```

Рис. 23. Пример текста ответа JSON

Из всего ответа нас будут интересовать только токены авторизации и сессии.

В консольном приложении в методе **Main()** необходимо задать токен приложения, логин/пароль, токены авторизации и сессии, которые мы получим от сервера для дальнейшей работы, а также некоторые другие параметры, суть которых будет раскрыта позднее.

```
static void Main(string[] args)
{
    //адрес сервера ELMA
    var elma = "127.0.0.1:8000";
    //токен приложения
    var appToken =
"1734AFBBE8E7559AA08508C9056E6C2C96D0BE21D6F05BBB6652890A9C1EA19E28230AF1F0B41702125BFAB9670E6039
52998BAD6D5C8C6AFA9EFFA9714933A2";
    //данные для авторизации
    var login = "admin";
    var password = "";
    //токены авторизации и сессии
    var authToken = "";
    var sessionToken = "";
    //Параметры поступления
    var typeUid = "da9ce014-0446-4a85-bdd0-b6fbece14ec2";
    var entityId = "1";
    //Признак найденного платежного поступления
    bool? paymentResult;

    //Реализация
```

```
    //Авторизация в системе ELMA по логину и паролю
    Authorization(elma, appToken, login, password, out authToken, out sessionToken);

    //Получение данных из ELMA, используя токены авторизации и сессии
    GetData(elma, authToken, sessionToken, typeUid, entityId, out paymentResult);

    //Изменение статуса
    if (paymentResult.HasValue && paymentResult.Value)
    {
        SetStatus(elma, authToken, sessionToken, typeUid, entityId);
    }

    Console.ReadKey();
}
```

Метод авторизации состоит из создания веб-запроса, в котором указана строка подключения:

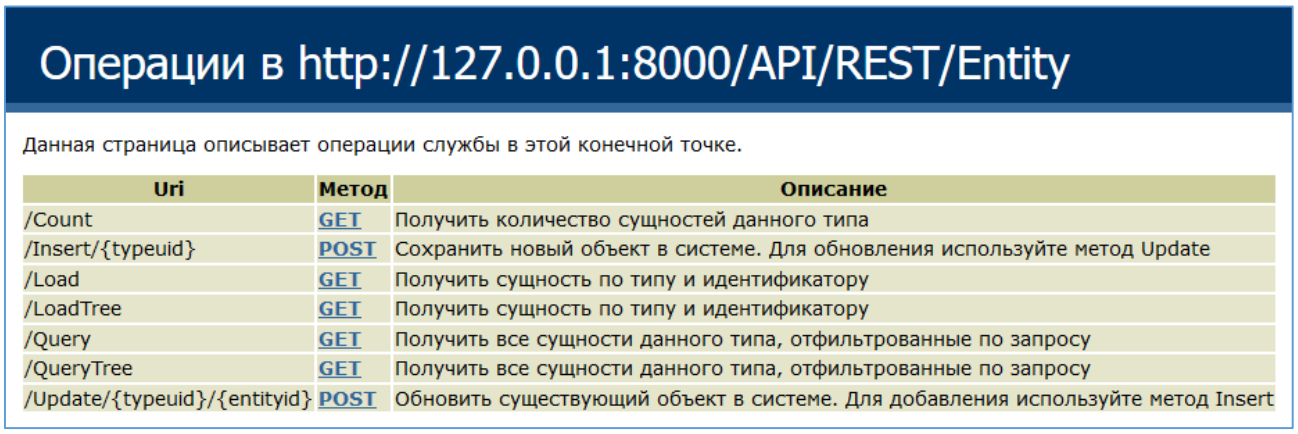
```
//авторизация
private static void Authorization(string elma, string appToken, string login, string password,
    out string authToken, out string sessionToken)
{
    //создаем веб запрос
    var url = string.Format("http://{0}/API/REST/Authorization/LoginWith?username={1}", elma,
login);
    var authRequest = (HttpWebRequest)WebRequest.Create(url);
    authRequest.Headers.Add("ApplicationToken", appToken);
    authRequest.Headers.Add("WebData-Version", "2.0");
    authRequest.Method = "POST";
    authRequest.Timeout = 10000;
    authRequest.ContentType = "application/json; charset=utf-8";

    //данные для отправки. используется для передачи пароля.
    var sentData = Encoding.UTF8.GetBytes(password);
    req.ContentLength = sentData.Length;
    using (var sendStream = authRequest.GetRequestStream())
    {
        sendStream.Write(sentData, 0, sentData.Length);
    }
    //получение ответа
    using (var authResponse = (HttpWebResponse)authRequest.GetResponse())
    {
        using (var sr = new StreamReader(authResponse.GetResponseStream(), Encoding.UTF8))
        {
            //достаем необходимые данные
            var dict = JsonConvert.DeserializeObject<Dictionary<string, string>>(sr.ReadToEnd());
            authToken = dict["AuthToken"];
            sessionToken = dict["SessionToken"];
        }
    }
}
```

В заголовке запроса указан токен приложения. Также в запрос передается метод (в нашем случае это POST и тип JSON). Обратите внимание на дополнительный заголовок **WebData-Version** – он позволяет значительно упростить работу с данными, подробнее можно прочитать в статье.

Далее формируем данные для передачи пароля, а также получения и парсинга ответа. В ответе нас интересует токен авторизации и токен сессии. Данные токены необходимы нам для дальнейшей работы (их будем передавать каждый раз, обращаясь на сервер).

4.3.2. Получение данных

Для работы с данными нам необходимо обратиться к списку доступных веб-сервисов и выбрать **Сервис для получения записей сущностей – Перейти к описанию REST запросов к методам** (Рис. 24).



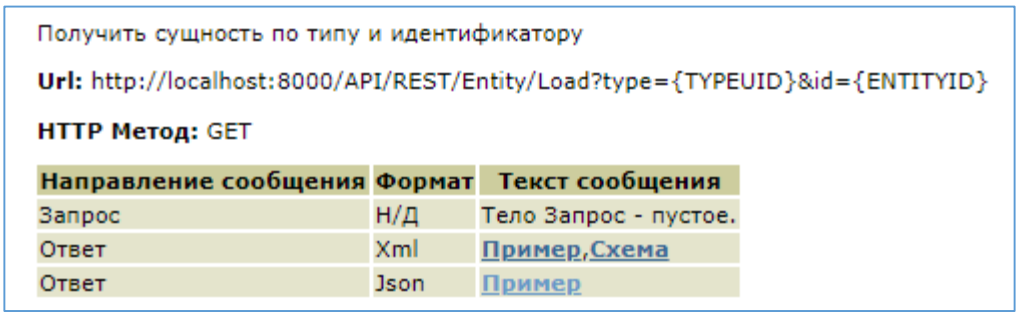
Операции в http://127.0.0.1:8000/API/REST/Entity

Данная страница описывает операции службы в этой конечной точке.

Uri	Метод	Описание
/Count	GET	Получить количество сущностей данного типа
/Insert/{typeuid}	POST	Сохранить новый объект в системе. Для обновления используйте метод Update
/Load	GET	Получить сущность по типу и идентификатору
/LoadTree	GET	Получить сущность по типу и идентификатору
/Query	GET	Получить все сущности данного типа, отфильтрованные по запросу
/QueryTree	GET	Получить все сущности данного типа, отфильтрованные по запросу
/Update/{typeuid}/{entityid}	POST	Обновить существующий объект в системе. Для добавления используйте метод Insert

Рис. 24. Сервис для получения записей сущностей. REST запросы к методам

Из всех методов выберем метод **/Load (GET)** (Рис. 25).



Получить сущность по типу и идентификатору

Url: http://localhost:8000/API/REST/Entity/Load?type={TYPEUID}&id={ENTITYID}

HTTP Метод: GET

Направление сообщения	Формат	Текст сообщения
Запрос	Н/Д	Тело Запрос - пустое.
Ответ	Xml	Пример,Схема
Ответ	Json	Пример

Рис. 25. Строка получения сущности по типу и идентификатору

Обратим внимание, что в запросе необходимо указать тип и идентификатор искомой сущности. В данном случае может возникнуть логичный вопрос: откуда внешнему приложению знать о типе и, тем более, об идентификаторе сущности?

Однако в реальных примерах интеграционных решений при создании сущности в системе ELMA будет настроена передача данных в соинтегрируемую систему для возможной дальнейшей работы. В случае, если данные необходимо передать на сайт,

можно воспользоваться методом **/Query (GET)**, который позволит выбрать все сущности по заданному фильтру.

В нашем случае будем искать по типу и идентификатору. Вернемся немного назад (см. выше) и еще раз взглянем на метод **Main()**, который содержит в себе тип и идентификатор искомой сущности. В нашем примере мы будем искать платежное поступление (Рис. 26 и Рис. 27), в котором в дальнейшем изменим статус (Рис. 28 и Рис. 29) на «Получено».

Соответствующие объекты в Дизайнере ELMA выглядят следующим образом (Рис. 26 – Рис. 29):

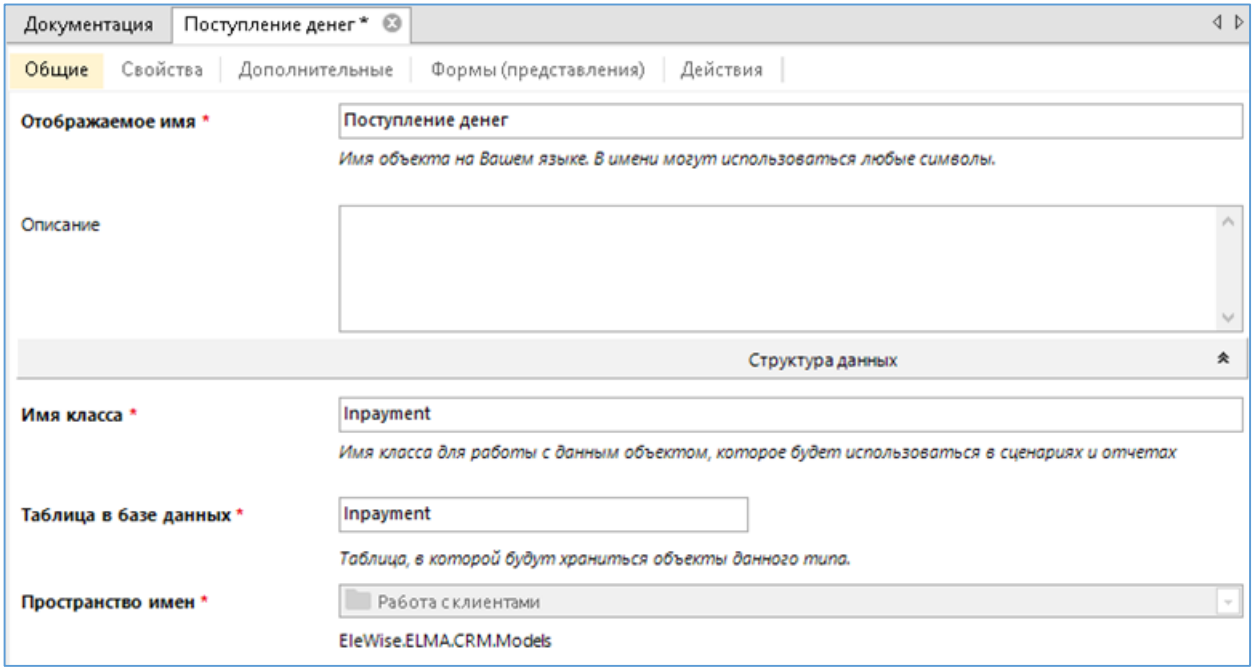
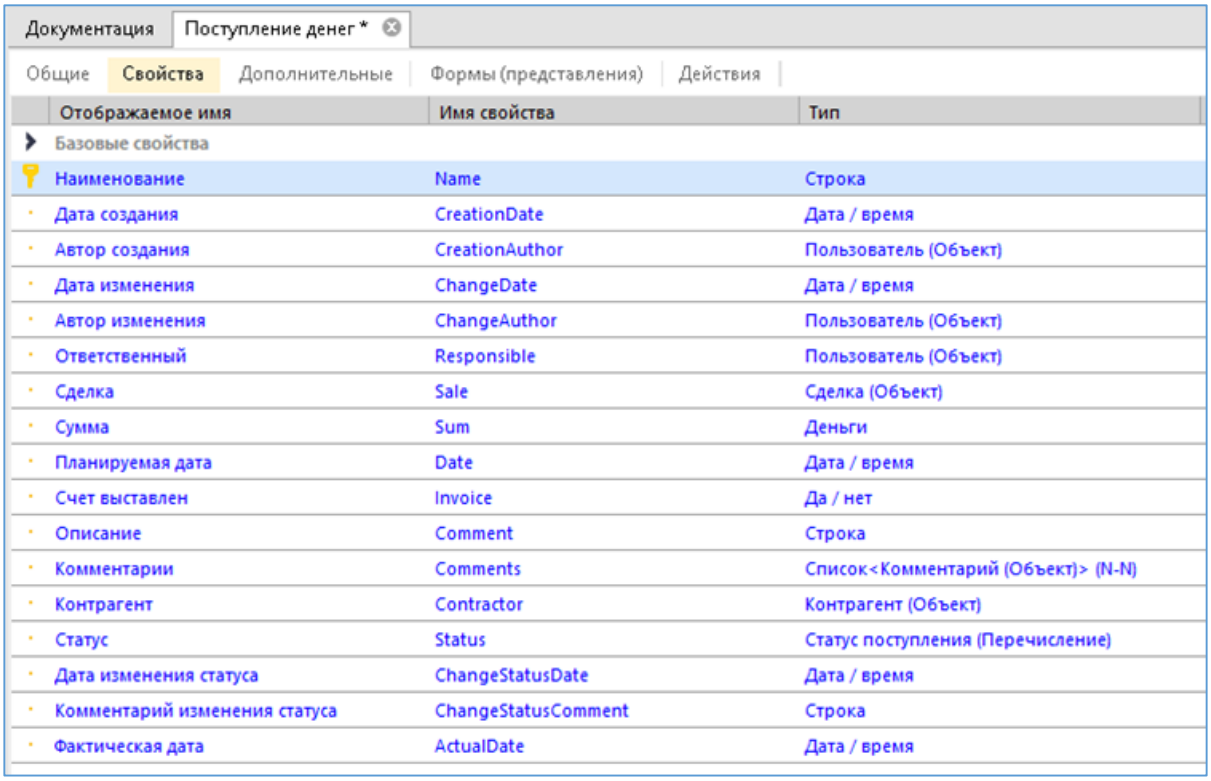


Рис. 26. Дизайнер ELMA. Карточка объекта «Поступление денег». Вкладка «Общие»



Отображаемое имя	Имя свойства	Тип
Базовые свойства		
Наименование	Name	Строка
Дата создания	CreationDate	Дата / время
Автор создания	CreationAuthor	Пользователь (Объект)
Дата изменения	ChangeDate	Дата / время
Автор изменения	ChangeAuthor	Пользователь (Объект)
Ответственный	Responsible	Пользователь (Объект)
Сделка	Sale	Сделка (Объект)
Сумма	Sum	Деньги
Планируемая дата	Date	Дата / время
Счет выставлен	Invoice	Да / нет
Описание	Comment	Строка
Комментарии	Comments	Список<Комментарий (Объект)> (N-N)
Контрагент	Contractor	Контрагент (Объект)
Статус	Status	Статус поступления (Перечисление)
Дата изменения статуса	ChangeStatusDate	Дата / время
Комментарий изменения статуса	ChangeStatusComment	Строка
Фактическая дата	ActualDate	Дата / время

Рис. 27. Дизайнер ELMA. Карточка объекта «Поступление денег». Вкладка «Свойства»

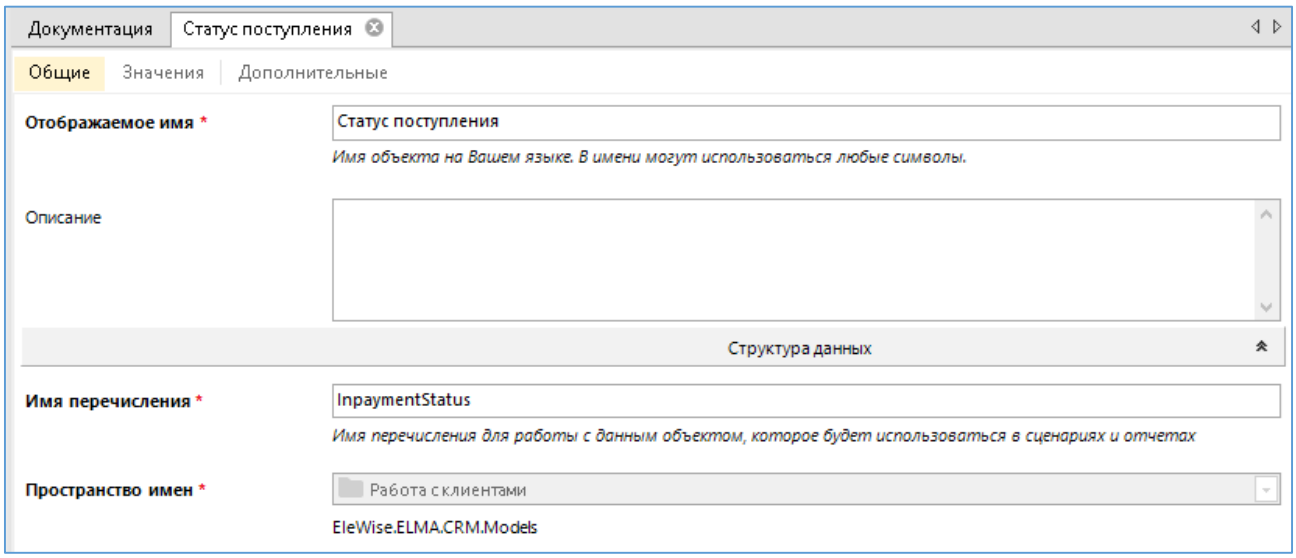


Рис. 28. Дизайнер ELMA. Карточка объекта «Статус поступления». Вкладка «Общие»

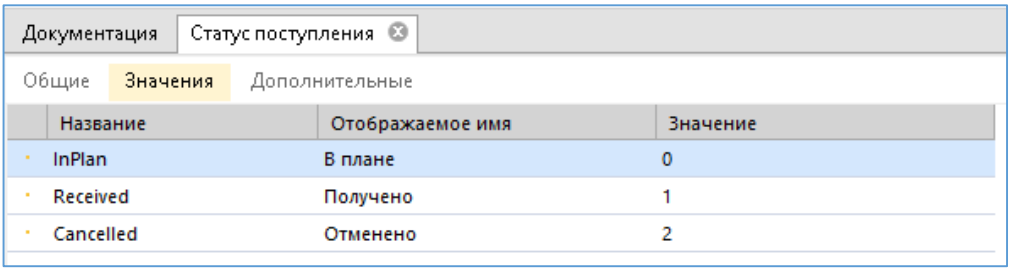


Рис. 29. Дизайнер ELMA. Карточка объекта «Статус поступления». Вкладка «Значения»

Они же в разделе веб-сервисов (Рис. 30 и Рис. 31):

Идентификатор типа объекта: da9ce014-0446-4a85-bdd0-b6fbece14ec2

Поступление денег

Id	Идентификатор объекта (Int64)
TypeUid	Идентификатор типа объекта (Guid)
Uid	Уникальный идентификатор. (тип: Уникальный идентификатор (GUID))
Name	Наименование. (тип: Строка)
CreationDate	Дата создания. (тип: Дата / время)
CreationAuthor	Автор создания. (тип: Пользователь (Объект))
	Id — Значение идентификатора сущности
	Name — Строковое представление сущности
	Uid — Уникальный идентификатор сущности
ChangeDate	Дата изменения. (тип: Дата / время)
ChangeAuthor	Автор изменения. (тип: Пользователь (Объект))
	Id — Значение идентификатора сущности
	Name — Строковое представление сущности
	Uid — Уникальный идентификатор сущности
Responsible	Ответственный. (тип: Пользователь (Объект))
	Id — Значение идентификатора сущности
	Name — Строковое представление сущности
	Uid — Уникальный идентификатор сущности

Рис. 30. Раздел веб-сервисов. «Поступление денег»

Статус поступления

InPlan	В плане (0)
Received	Получено (1)
Cancelled	Отменено (2)

Список перечислений

{ "InPlan": "В плане (0)","Received": "Получено (1)","Cancelled": "Отменено (2)" }

Рис. 31. Раздел веб-сервисов. «Статус поступления»

В результате выполнения получим следующий JSON-ответ (Рис. 32):

```
{
        "Items":[{
                "Data":{
                        "Items":[{
                                "Data":{
                                        "Items":null,
                                        "Value":"Строковое содержимое"
                                },
                                "DataArray":[{
                                        "Items":null,
                                        "Value":"Строковое содержимое"
                                }],
                                "Name":"Строковое содержимое",
                                "Value":"Строковое содержимое"
                        }],
                        "Value":"Строковое содержимое"
                },
                "DataArray":[{
                        "Items":[{
                                "Data":{
                                        "Items":null,
                                        "Value":"Строковое содержимое"
                                },
                                "DataArray":[{
                                        "Items":null,
                                        "Value":"Строковое содержимое"
                                }],
                                "Name":"Строковое содержимое",
                                "Value":"Строковое содержимое"
                        }],
                        "Value":"Строковое содержимое"
                }],
                "Name":"Строковое содержимое",
                "Value":"Строковое содержимое"
        }],
        "Value":"Строковое содержимое"
}
```


Рис. 32. Пример текста ответа JSON

Из примера видно, что результатом будет древовидная структура всего платежного поступления. Данную структуру можно посмотреть в Дизайнере ELMA в разделе **Объекты**.

Рассмотрим метод **GetData()**, в котором аналогично создадим запрос со строкой определенного формата, передав тип сущности и ее идентификатор. Далее передадим метод (GET), токены авторизации и сессии, а также тип (JSON).

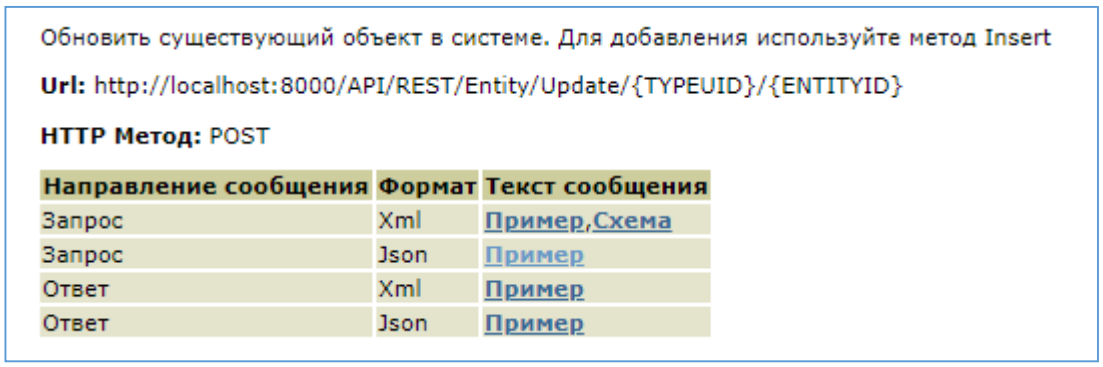
```
//получение данных
private static void GetData(string elma,
    string authToken,
    string sessionToken,
    string typeUid,
    string entityId,
    out bool? paymentResult)
{
    var url = string.Format("http://{0}/API/REST/Entity/Load?type={1}&id={2}", elma, typeUid,
entityId);
    var entityRequest = (HttpWebRequest)WebRequest.Create(url) as;
    entityRequest.Method = "GET";
    entityRequest.Headers.Add("AuthToken", authToken);
    entityRequest.Headers.Add("SessionToken", sessionToken);
    entityRequest.Headers.Add("WebData-Version", "2.0");
    entityRequest.Timeout = 10000;
```

```
    entityRequest.ContentType = "application/json; charset=utf-8";

    using (var response = (HttpWebResponse)entityRequest.GetResponse())
    {
        using (var streamReader = new StreamReader(response.GetResponseStream()), Encoding.UTF8))
        {
            var entity = streamReader.ReadToEnd();
            paymentResult = !string.IsNullOrWhiteSpace(entity);
        }
    }
}
```

В качестве положительного ответа нас интересует ответ, который гарантирует наличие на сервере данных по указанному типу сущности и ее идентификатору. Также и сам ответ мы вернем в метод **Main()**, поскольку для дальнейшего изменения статуса нам потребуется изменить его значение в структуре ответа и вернуть обратно на сервер. Для этого перейдем к следующему пункту нашего примера.

4.3.3. Изменение данных

Для изменения сущности останемся в текущем сервисе, но обратимся к другому методу (Рис. 33) – **/Update/{TYPEUID}/{ENTITYID} (POST)**.



Обновить существующий объект в системе. Для добавления используйте метод Insert

Url: http://localhost:8000/API/REST/Entity/Update/{TYPEUID}/{ENTITYID}

HTTP Метод: POST

Направление сообщения	Формат	Текст сообщения
Запрос	Xml	Пример,Схема
Запрос	Json	Пример
Ответ	Xml	Пример
Ответ	Json	Пример

Рис. 33. Строка обновления существующего в системе объекта

Для изменения одного поля достаточно правильно передать только изменяемые поля. Рассмотрим последний метод нашего приложения **SetStatus()**.

```
// изменение статуса в случае если данные найдены
private static void SetStatus(string elma,
    string authToken,
    string sessionToken,
    string typeUid,
    string entityId)
{
    var url = string.Format("http://{0}/API/REST/Entity/Update/{1}/{2}", elma, typeUid,
entityId);
    var updateRequest = (HttpWebRequest)WebRequest.Create(url);
    updateRequest.Method = "POST";
    updateRequest.Headers.Add("AuthToken", authToken);
    updateRequest.Headers.Add("SessionToken", sessionToken);
    updateRequest.Headers.Add("WebData-Version", "2.0");
```

```
    updateRequest.Timeout = 10000;
    updateRequest.ContentType = "application/json; charset=utf-8";

    // при формировании json-строки укажем параметры,
    // которые нам необходимо изменить: статус и комментарий
    var json = "{\"Status\": \"1\",\"ChangeStatusComment\":\"Получена оплата\"}";
    var sentData = Encoding.UTF8.GetBytes(json);
    updateRequest.ContentLength = sentData.Length;
    using (var sendStream = updateRequest.GetRequestStream())
    {
        sendStream.Write(sentData, 0, sentData.Length);
    }

    using (var response = (HttpWebResponse)updateRequest.GetResponse())
    {
        using (var streamReader = new StreamReader(response.GetResponseStream(), Encoding.UTF8))
        {
            var responseData = streamReader.ReadToEnd();
            // обработка полученных данных
        }
    }
}
```

По аналогии с предыдущими методами сформируем запрос, передавая строку с параметрами сущности, метод (POST), токены авторизации и сессии и метод (JSON). Для изменения статуса необходимо будет составить JSON-строку следующего вида:

```
{
    "Status ": "1",
    "ChangeStatusComment": " Получена оплата "
}
```

В строке передаем новые данные по смене статуса: изменяем значение на «1», а также добавляем некоторый комментарий (последнее в данном случае не является обязательным). В результате работы метода статус платежного поступления будет сразу же изменен.

Все описанные нами методы поочередно вызовем в методе **Main()**. При этом последний метод **SetStatus()** будем вызывать только в том случае, если предыдущий метод **GetData()** вернет не пустой результат. В противном случае его запуск будет не нужен.

4.4. Пример WSDL

Выполним эту же задачу, используя WSDL-сервисы. Работать также будет с помощью консольного приложения, выполненного в Visual Studio.

Для их использования необходимо добавить их в приложение. Добавление осуществляется в обозревателе решений, по умолчанию расположенном в правой части окна (Рис. 34). Для добавления сервиса необходимо щелкнуть правой кнопкой мыши на каталоге **Service References** и выбрать пункт **AddServiceReference**. В открывшемся окне в поле **Адрес** необходимо ввести требуемую ссылку и нажать на кнопку **Перейти**.

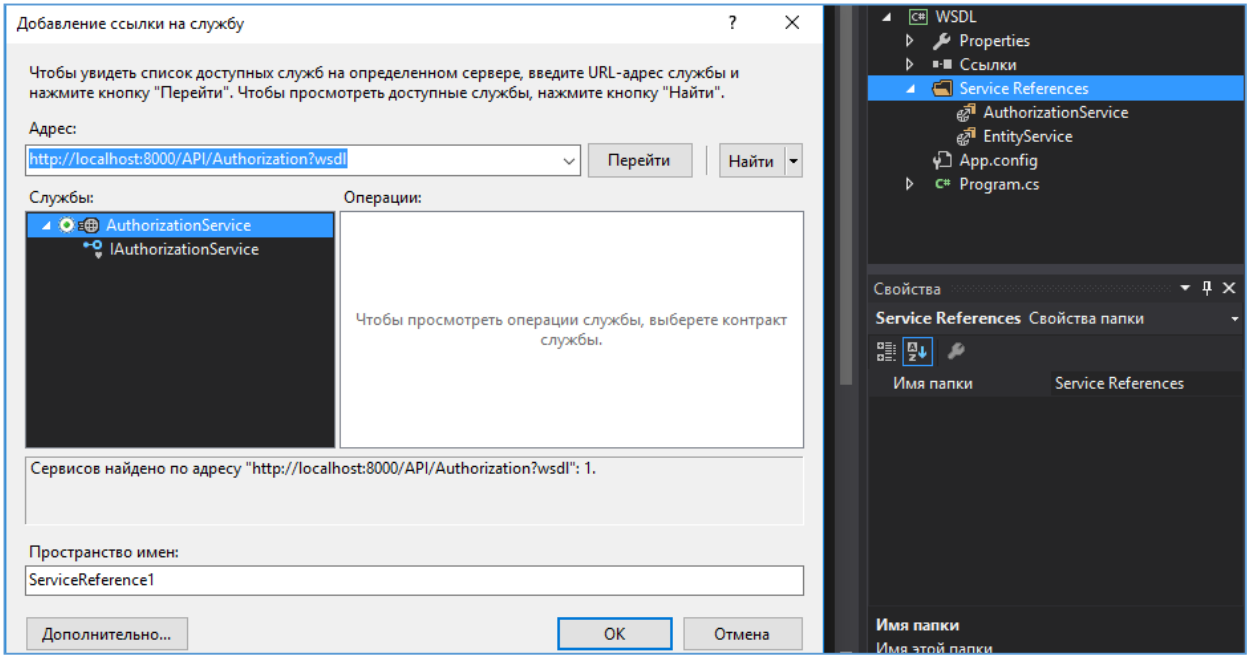


Рис. 34. Обозреватель решений. Добавление сервиса

В случае успешного перехода необходимо нажать на кнопку **ОК**.

Для получения требуемой ссылки (по аналогии с предыдущим примером) воспользуемся справкой по публичным веб-службам ELMA (**http://<адрес сервера ELMA>:<порт>/Api/Help/Services**) и выберем WSDL в сервисе авторизации. В результате этого получим ссылку следующего вида: http://localhost:8000/API/Authorization?wsdl.

Аналогичным образом добавим ссылку на **IEntityService.**

Перейдем непосредственно к приложению. Для удобства работы создадим статичный метод **Start()**.

```
/// <summary>
/// Фабрика создания каналов связи авторизации
/// </summary>
static ChannelFactory<AuthorizationService.IAuthorizationServiceChannel> authServiceFactory {
get; set; }

/// <summary>
/// Фабрика создания каналов связи для работы с сущностями
/// </summary>
static ChannelFactory<EntityService.IEntityServiceChannel> entityServiceFactory { get; set; }

static void Start()
{
    //адрес сервера ELMA
    var elma = "127.0.0.1:8000";
    //токен приложения
    var appToken =
"1734AFBBE8E7559AA08508C9056E6C2C96D0BE21D6F05BBB6652890A9C1EA19E28230AF1F0B41702125BFAB9670E6039
52998BAD6D5C8C6AFA9EFFA9714933A2";
    //данные для авторизации
    var login = "admin";
    var password = "";
    //токены авторизации и сессии
    Guid authToken = Guid.Empty;
    string sessionToken = "";
    //Параметры поступления
    var typeUid = "da9ce014-0446-4a85-bdd0-b6fbece14ec2";
    var entityId = "1";
    //Признак найденного платежного поступления
    bool? paymentResult;

    //Данные
    EntityService.WebData item;

    authServiceFactory = new ChannelFactory<AuthorizationService.IAuthorizationServiceChannel>(
        new BasicHttpBinding(),
        new EndpointAddress(string.Format("http://{0}/API/Authorization", elma))
    );

    entityServiceFactory = new ChannelFactory<EntityService.IEntityServiceChannel>(
        new BasicHttpBinding() { MaxReceivedMessageSize = Int32.MaxValue },
        new EndpointAddress(string.Format("http://{0}/API/Entity", elma))
    );

    //авторизация
    Authorization(appToken, login, password, out authToken, out sessionToken);

    //получение данных
    GetData(authToken, sessionToken, typeUid, entityId, out paymentResult, out item);
    //изменение данных
    if (paymentResult.HasValue && paymentResult.Value)
    {
        SetStatus(authToken, sessionToken, typeUid, entityId, item);
    }

    authServiceFactory.Dispose();
    entityServiceFactory.Dispose();

    Console.ReadKey();
}
```

В данном случае (как и в прошлом примере) требуется указать токен приложения, данные для авторизации, токены авторизации и параметры сущности, с которой

будем работать. В этом методе будут последовательно вызываться методы авторизации, получения и изменения данных.

Переменные **authServiceFactory** и **entityServiceFactory** служат для создания каналов связи с сервером. Непосредственное создание каналов будет происходить в последующих методах перед самым их использованием.

4.4.1. Авторизация в системе

Перейдем к методу авторизации в системе:

```
private static void Authorization(string appToken,
    string login,
    string password,
    out Guid authToken,
    out string sessionToken)
{
    using (var authorizationService = authServiceFactory.CreateChannel())
    {
        using (new OperationContextScope(authorizationService))
        {
            WebOperationContext.Current.OutgoingRequest.Headers.Add("ApplicationToken",
appToken);
            var token = authorizationService.LoginWithUserName(login, password);
            authToken = token.AuthToken;
            sessionToken = token.SessionToken;
        }
    }
}
```

В данный метод мы передаем токен приложения и пару логин/пароль для авторизации в системе ELMA. В результате выполнения данного метода для дальнейшей работы мы получим от сервера токены авторизации и сессии.

В первую очередь создается канал связи с сервером методом **CreateChannel()** переменной **authServiceFactory**.

Вся основная работа будет производиться в **using(….){…}**. В круглых скобках мы инициализируем новый экземпляр класса **OperationContextScope**, использующий вновь созданный канал **authorizationService**. В фигурных скобках по заданному токену приложения, а также данным авторизации в системе ELMA мы получаем в ответ от сервера токены авторизации и сессии, необходимые для работы.

4.4.2. Получение данных

Метод получения данных:

```
private static void GetData(Guid authToken,
    string sessionToken,
    string typeUid,
    string entityId,
    out bool? paymentResult,
    out EntityService.WebData webDataItem)
```

```
{
    using(var entityService = entityServiceFactory.CreateChannel())
    {

        using(new OperationContextScope(entityService))
        {
            WebOperationContext.Current.OutgoingRequest.Headers.Add("AuthToken",
authToken.ToString());
            WebOperationContext.Current.OutgoingRequest.Headers.Add("SessionToken",
sessionToken);

            var tempData = entityService.Load(typeUid, entityId);
            if (tempData != null && tempData.Items.Count > 0)
            {
                Console.WriteLine("Указанная платежка найдена! Изменяем статус на 'Оплачено'.");
                paymentResult = true;
                webDataItem = tempData;
                foreach (var item in tempData.Items)
                {
                    Console.WriteLine(item.Name + ": " + item.Value);
                }
            }
            else
            {
                Console.WriteLine("Указанная платежка не найдена, необходимо уточнить поиск");
                paymentResult = false;
                webDataItem = null;
            }
        }
    }
}
```

В методе **GetData()** мы получаем необходимые данные для решения поставленной задачи. Разберемся подробнее в работе метода.

1. В качестве параметров передаем токены авторизации и сессии, а также тип искомой сущности и его идентификатор.
2. В результате работы метода получим сущность, если она найдена по заданным параметрам, или null, если нет. Найденная сущность будет храниться в параметре **webDataItem** типа **EntityService.WebData**.

Первой же строкой (как и в предыдущем методе) используется переменная **entityServiceFactory**, которая создает канал с сервером и позволяет инициализировать дальнейшую работу. Также для работы необходимо указать токены авторизации и сессии, без валидности которых дальнейшая работа просто невозможна.

Далее методом **Load()**, созданного канала с сервером (**entityService**), загружаем необходимую сущность по его типу **typeUid** и идентификатору **entityId**. В случае найденной сущности передаем ее в параметр **webDataItem**, иначе – записываем туда null.

Посмотрим на структуру сущности:

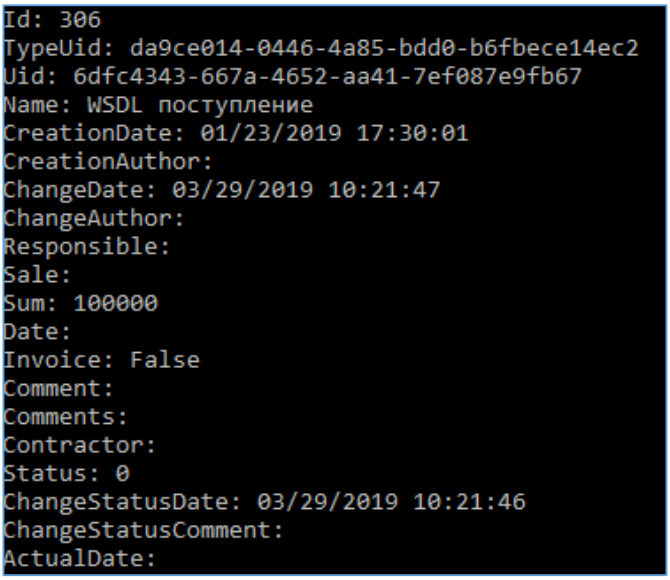


Рис. 35. Структура сущности

Для понимания значения полей, их типа и значений, которые они могут принимать, необходимо обратиться к Дизайнеру ELMA и на вкладке **Объекты** перейти к объекту **Поступление денег**. В данном случае нас интересует поле **Статус** (**Status: 0**), которое является перечислением и может принимать следующие значения:

- 0 – В плане;
- 1 – Получено;
- 2 – Отменено.

В нашем случае установлен статус «В плане», а нам необходимо выбрать «Получено». Для изменения статуса перейдем к следующему методу нашего приложения.

4.4.3. Изменение данных

Метод изменения данных:

```
private static void SetStatus(Guid authToken,
    string sessionToken,
    string typeUid,
    string entityId,
    EntityService.WebData webDataItem)
{
    using(var entityService = entityServiceFactory.CreateChannel())
    {
        using(new OperationContextScope(entityService))
        {
            WebOperationContext.Current.OutgoingRequest.Headers.Add("AuthToken",
authToken.ToString());
            WebOperationContext.Current.OutgoingRequest.Headers.Add("SessionToken",
sessionToken);

            var webData = new EntityService.WebData();
```

```
            var statusWebData = new EntityService.WebDataItem
            {
                Name = "Status",
                Value = "1"
            };
            webData.Items = new EntityService.WebDataItem[]
            {
                statusWebData
            };
            webDataItem.Items = webData.Items;
            Console.WriteLine("Статус изменен на 'Оплачено'!");

            var tempData = entityService.Update(typeUid, entityId, webDataItem);
        }
    }
}
```

Метод **SetStatus** необходим для установки статуса «Получено» у найденной сущности. Кроме того, данный метод принимает в качестве параметров токены авторизации и сессии, тип и идентификатор сущности, а также саму сущность.

Как и в предыдущем методе создается канал, указываются токены авторизации и сессии. Далее создаем новое свойство «Status» в формате **EntityService.WebData** и устанавливаем значение «1».

Последней строкой метода обновляем данные на сервере, используя метод **Update()** с параметрами сущности (типом и идентификатором), а также самой сущностью.

В результате работы приложения статус платежного поручения изменится на «Получено».

Глава 5. Отправка e-mail в ELMA

Один из важнейших моментов избавления от рутинной работы – автоматическая отправка сообщений по электронной почте.

Например, применительно к работе интернет-магазина – это может быть отправка оповещений клиенту о статусе обработки заказа. После формирования заказа на сайте происходит запуск бизнес-процесса «Обработка заказа».

В личном кабинете сайта доступна возможность включения или отключения опции «Получать уведомления по электронной почте о статусе заказа». Если данная настройка включена, то после успешного запуска процесса клиенту приходит уведомление о смене статуса заказа на «В обработке». Далее в ходе бизнес-процесса клиент также получает уведомления о смене статусов. Эти простые действия позволяют быть в курсе всех изменений по заказу.

Для реализации данного функционала в системе ELMA доступны различные возможности. Можно:

1. Использовать дополнительный компонент **Оповещение на E-mail** (доступный для получения в магазине готовых решений ELMA Store), который реализует блок для его дальнейшего размещения на карте бизнес-процесса. Подробнее см. в **Разделе 5.1**.
2. В бизнес-процессе написать сценарий на языке C# по отправке сообщения, а затем использовать его в блоках **Сценарий**. Подробнее см. в **Разделе 5.2**.
3. Написать функцию по отправке почты на языке C# и разместить ее в глобальных модулях, чтобы в дальнейшем была возможность вызывать ее в любом бизнес-процессе. Подробнее см. в **Разделе 5.3**.
4. Использовать собственное пользовательское расширение, которое реализует блок для размещения на карте бизнес-процесса.
5. Подготовить внешний бизнес-процесс, в который поместить сценарий C# и вызывать такой бизнес-процесс в качестве подпроцесса.

5.1. Использование дополнительного компонента «Оповещение на E-mail»

После приобретения компонента **Оповещение на E-mail** из ELMA Store его требуется установить.

После этого в Дизайнере ELMA на вкладке **Графическая модель** на боковой панели инструментов карточки бизнес-процесса в блоке **Элементы BPMN** в группе **Операции** появится дополнительный блок **Сообщение на E-mail** (Рис. 36).

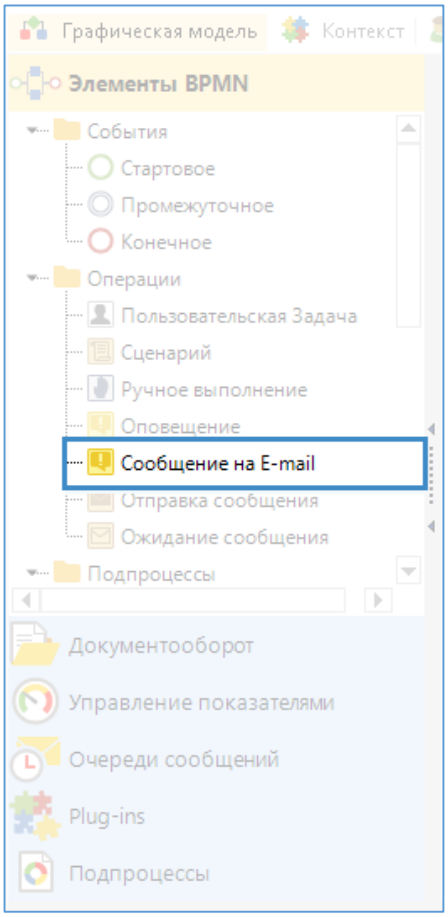
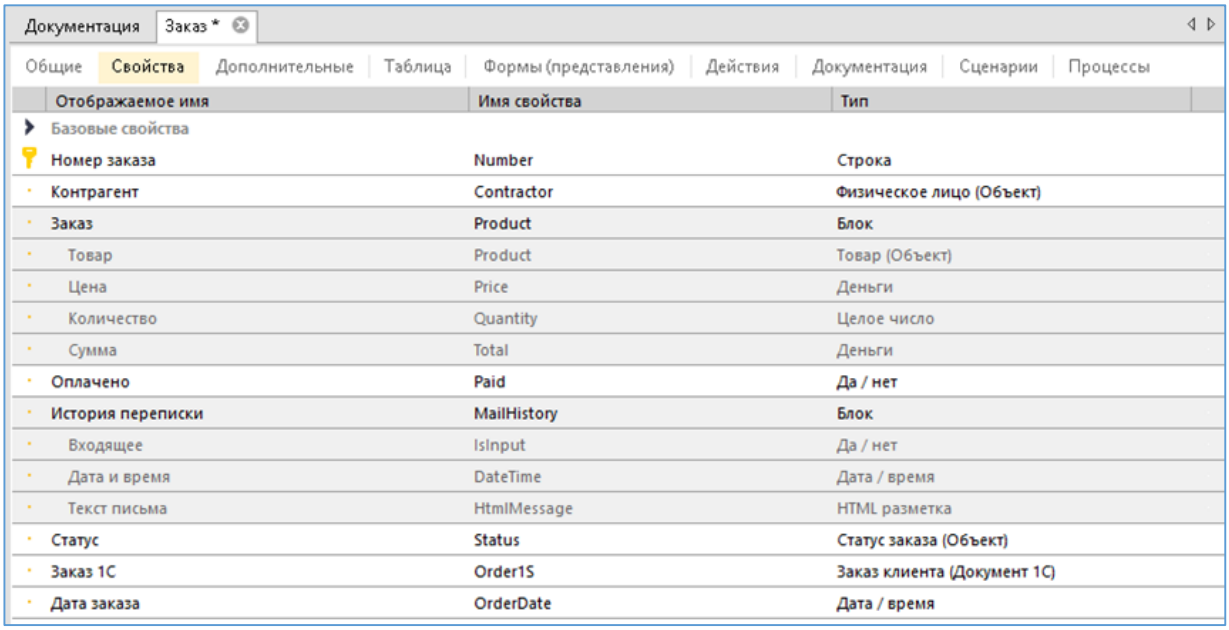


Рис. 36. Блок «Сообщение на E-mail»

Для реализации данного примера был создан объект **Заказ** со следующими контекстными переменными (Рис. 37):



Отображаемое имя	Имя свойства	Тип
Базовые свойства		
Номер заказа	Number	Строка
Контрагент	Contractor	Физическое лицо (Объект)
Заказ	Product	Блок
Товар	Product	Товар (Объект)
Цена	Price	Деньги
Количество	Quantity	Целое число
Сумма	Total	Деньги
Оплачено	Paid	Да / нет
История переписки	MailHistory	Блок
Входящее	IsInput	Да / нет
Дата и время	DateTime	Дата / время
Текст письма	HtmlMessage	HTML разметка
Статус	Status	Статус заказа (Объект)
Заказ 1С	Order1S	Заказ клиента (Документ 1С)
Дата заказа	OrderDate	Дата / время

Рис. 37. Объект «Заказ». Список контекстных переменных

Блок **Сообщение на E-mail** (Рис. 36) требуется переместить на карту бизнес-процесса «Обработка заказа» в те места, где по логике данного процесса клиенту должно быть отправлено письмо о смене статуса заказа.

Далее следует произвести настройку данной операции. Для настройки блока в контекст процесса требуется добавить дополнительные атрибуты:

1. Адрес электронной почты контрагента (Рис. 38):
   - **Имя свойства** – «EmailAddressContragent»;
   - Тип переменной – «Строка».

Данную переменную в дальнейшем требуется выбрать в окне настройки операции **Сообщение на E-mail** на вкладке **Получатели оповещения** (Рис. 39).

2. Текст E-mail сообщения:
   - **Имя свойства** – «EmailBody»;
   - **Тип переменной** – «HTML-разметка».

Данную переменную в дальнейшем требуется указать в окне настройки операции **Сообщение на E-mail** на вкладке **Шаблон оповещения** в поле **Текст сообщения** (Рис. 40).

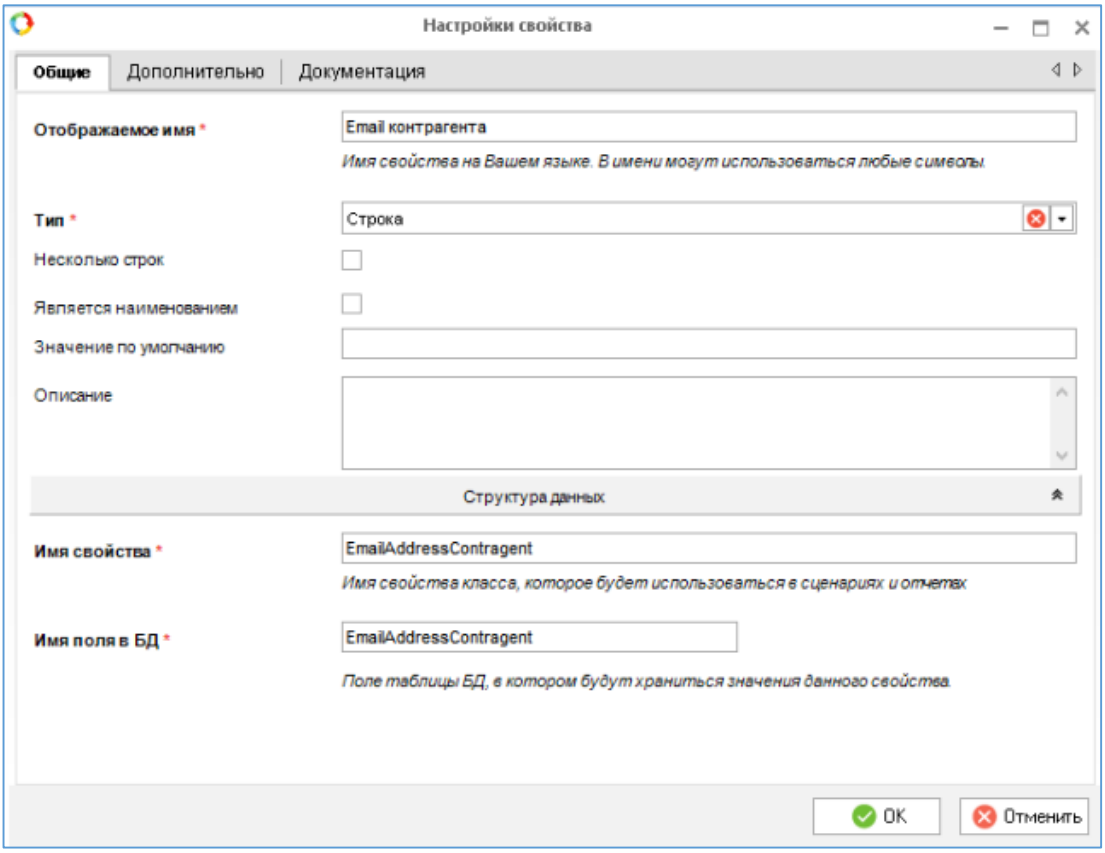


Рис. 38. Окно настройки контекстной переменной

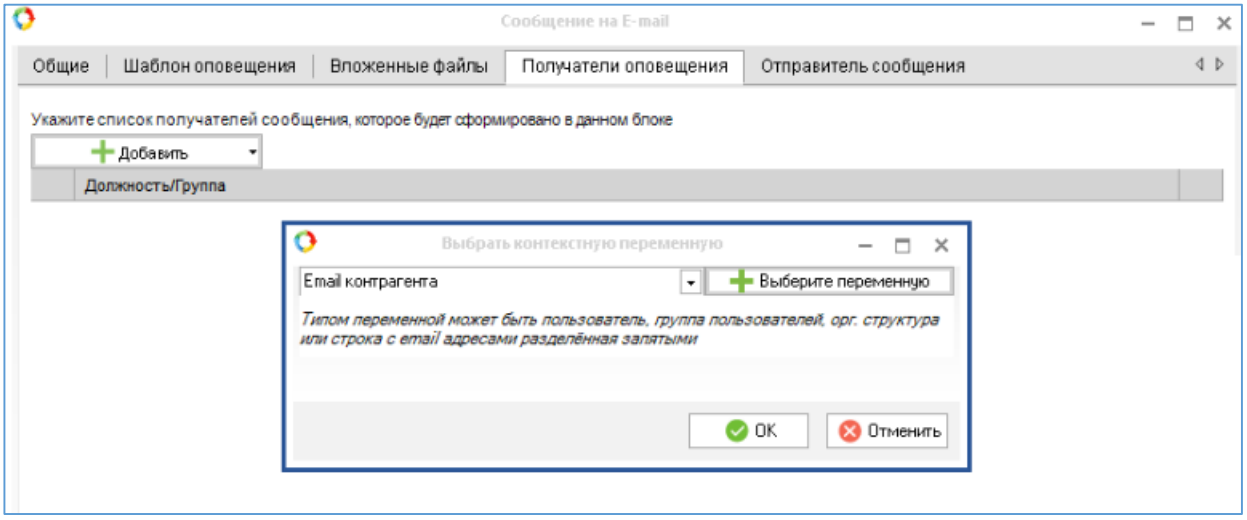


Рис. 39. Окно настройки операции «Сообщение на E-mail». Вкладка «Получатели оповещения»

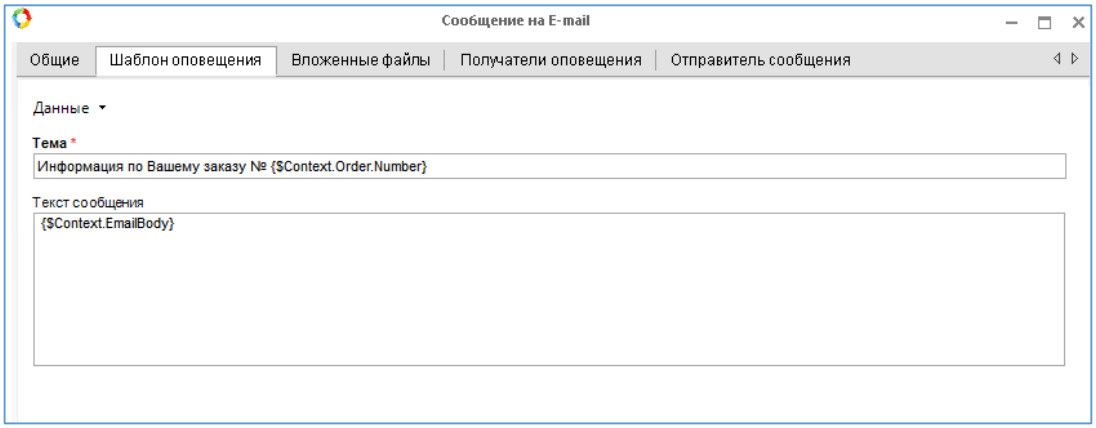


Рис. 40. Окно настройки операции «Сообщение на E-mail». Вкладка «Шаблон оповещения»

Указанные атрибуты должны быть предварительно созданы/откорректированы перед блоком с помощью сценария либо в ручном режиме.

Контекстная переменная **EmailBody** может быть предварительно создана/изменена в следующем сценарии:

```
context.EmailBody = new HtmlString("<h1>Ваш заказ №669 В обработке</h1>");
```

Преимущества:

1. Готовое коробочное решение.
2. Бесплатный компонент.
3. Большое количество различных настроек: тема и текст письма, получатели, возможность вложения документов в письмо.
4. В большинстве случаев не требуется иметь навыки программирования для написания кода на языке C#.

Недостатки:

1. Может быть использован не со всеми версиями системы ELMA.
2. Использует для отправки только системные настройки SMTP-сервера.

5.2. Применение сценария для отправки e-mail

Напишем сценарий на языке C#. Подробнее с информацией по написанию сценариев можно ознакомиться в Базе знаний ELMA.

Откроем бизнес-процесс «Обработка заказа». Перенесем на карту процесса блок **Сценарий**, затем двойным щелчком мыши по нему перейдем к его настройке: введем название и создадим новый сценарий (Рис. 41).

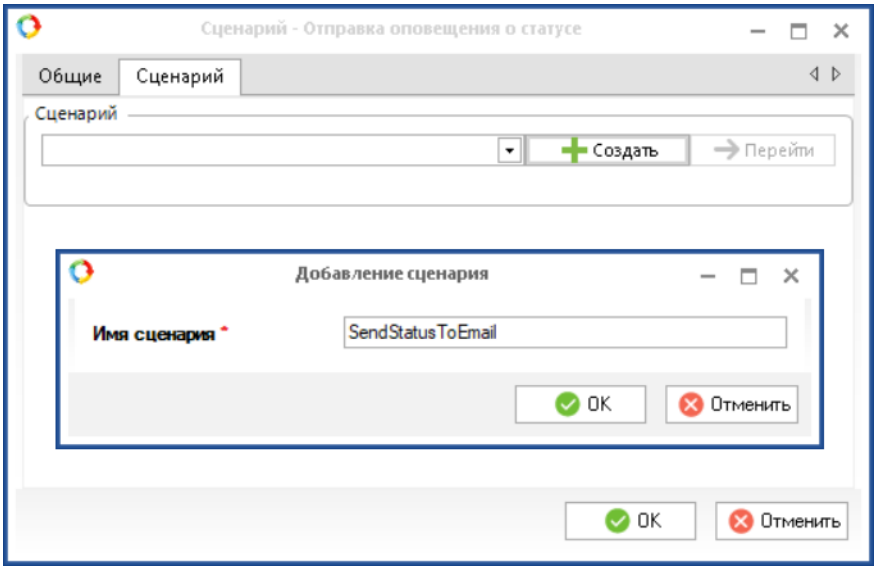


Рис. 41. Настройка блока «Сценарий». Добавление сценария

Для корректной работы сценария должны быть подключены следующие пространства имен:

```
using System;
using System.Net.Mail;
using EleWise.ELMA.API;
using EleWise.ELMA.Services;
using EleWise.ELMA.Messaging.Email;
using EleWise.ELMA.Logging;
using Context = EleWise.ELMA.Model.Entities.ProcessContext.P_OrderProcessing;
```

Текст сценария:

```
public virtual void SendStatusToEmail(Context context)
{
    // Инициализация подключения к SMTP серверу на основе настроек исходящей почты ELMA
    var emailSettingsModule = Locator.GetService<EmailSettingsModule>().Settings;

    var from = emailSettingsModule.SenderEMail;
    var host = emailSettingsModule.SmtpHost;
    var port = emailSettingsModule.SmtpPort;
    var user = emailSettingsModule.SmtpUserName;
    var pass = emailSettingsModule.SmtpPassword;
    var ssltype = emailSettingsModule.SmtpSSLType;

    var smtp_settings = new SmtpSettings(host, port, ssltype, user, pass);
```

```
    if (context.Order.Contractor.Email.Count > 0 && from != null)
    {
        var mail = new MailMessage();

        mail.From = new MailAddress(from);
        foreach (var email in context.Order.Contractor.Email)
        {
            // Установка адреса получателя из свойств контрагента
            mail.To.Add(new MailAddress(email.EmailString));
        }
        // Формирование темы и тела письма
        var currentusername = PublicAPI.Portal.Security.User.GETCurrentUser().FullName;
        mail.Subject = "Информация по Вашему заказу №" + context.Order.Number;
        mail.IsBodyHtml = true;
        mail.Body = "<h1>Ваш заказ №" + context.Order.Number + " " + context.Order.Status.Name +
"</h1>";
        mail.Body += "<h2>Параметры заказа: </h2>";
        var cnt = 0;
        mail.Body +=
"<table><tr><th>№</th><th>Артикул</th><th>Цена</th><th>Количество</th><th>Сумма</th></tr>";
        // Формирование таблицы с подробностями заказа
        foreach (var code in context.Order.Product)
        {
            cnt++;
            mail.Body += "<tr>" +
                "<td>" + cnt + "</td>" +
                "<td>" + code.Product.Name + "</td>" +
                "<td>" + code.Price + "</td>" +
                "<td>" + code.Quantity.ToString() + "</td>" +
                "<td>" + code.Total.ToString() + "</td>" +
                "</tr>";
        }
        mail.Body += "</table>";

        try
        {
            // Отправка сообщения
            PublicAPI.Services.Email.SendMessage(smtp_settings, mail, null, null);
        }
        catch (Exception ex)
        {
            Logger.Log.Error("Ошибка при отправке почты: " + ex);
        }
        finally
        {
            mail.Dispose();
        }
    }
}
```

В случае, если требуется использовать собственные настройки сервера исходящей почты, то блок инициализации подключения к SMTP-серверу будет выглядеть следующим образом:

```
// Инициализация подключения к SMTP серверу с индивидуальными настройками
    var from = "testuser@mail.ru";
    var host = "smtp.mail.ru";
    var port = 465;
    var user = " testuser@mail.ru";
    var pass = "cegthgfhjkm";
    var ssltype = SSLType.SSL;
```

Если же для отправки почты будут использоваться только системные настройки ELMA, то инициализацию **smtp_settings** можно опустить, а для отправки почты использовать команду вида:

```
PublicAPI.Services.Email.SendMessage(mail, null, null);
```

Отправка почты с использованием системных настроек SMTP будет корректно работать только при их правильном заполнении в веб-приложении ELMA в разделе ***Администрирование – Система – Настройки системы – Настройки исходящей почты***. Подробнее см. в **Главе 4.2.3** Краткого руководства по Платформе ELMA BPM.

Результат выполнения сценария изображен на Рис. 42.

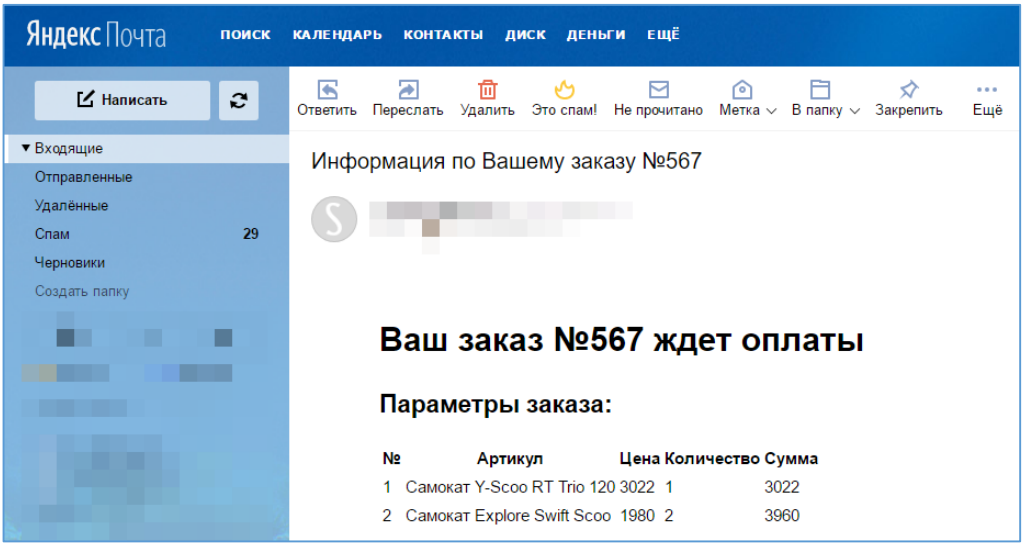


Рис. 42. Входящее письмо со статусом заказа

Недостатки данного подхода:

1. Требуется дублирование кода при необходимости применения сценария в других бизнес-процессах. Данный недостаток можно устранить, если реализовать код в пользовательском расширении (см. раздел ниже).
2. Сложность в поддержке и настройке.
3. Требуются навыки программирования на языке C#.

Подробнее ознакомиться с возможностью отправки почты можно в статье Базы знаний ELMA.

5.3. Использование глобального модуля и пользовательских расширений

Пользовательское расширение – это операция, размещаемая на графической модели процесса и выполняющая действия, определяемые сценарием. Пользовательские расширения позволяют определить и использовать один и тот же сценарий в разных моделях процессов. При этом для использования готовых расширений не требуется наличие навыков написания сценариев.

Хотя пользовательские расширения могут работать самостоятельно, в данном примере реализуем методы непосредственной работы с почтой в глобальном модуле, а вызов этих метод запакуем в пользовательское расширение.

Важно: пользовательские расширения могут работать самостоятельно (как независимые сценарии). В этом случае код внутри пользовательского расширения почти не отличается от кода в сценариях процесса.

В Дизайнере ELMA создадим глобальный модуль **Helper** (Рис. 43), добавим к нему класс **Scripts** и реализуем два метода: **SendMail** и **AddOrderHistory**.

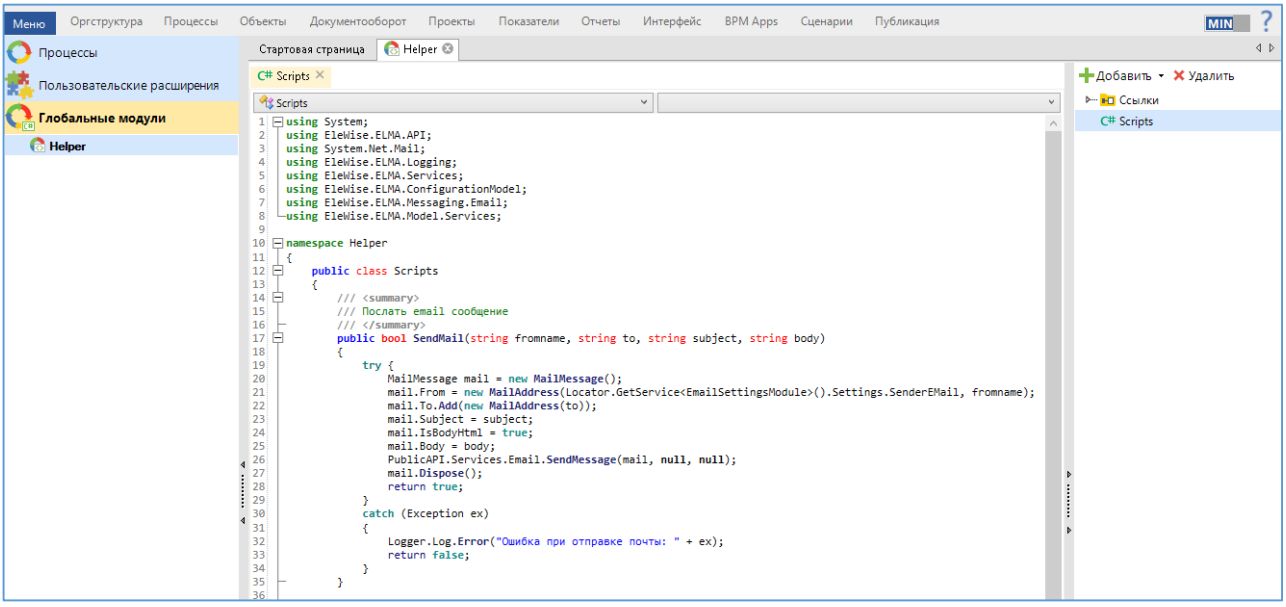


Рис. 43. Создание глобального модуля Helper

Метод **SendMail** отправляет e-mail сообщение с помощью системных настроек SMTP, а в качестве входных параметров принимает строковые атрибуты: имя отправителя, адрес получателя, тему и тело письма. Метод возвращает **True**, если отправка прошла успешно, либо **False**, если не удалось отправить сообщение.

```
public bool SendMail(string fromname, string to, string subject, string body)
{
    MailMessage mail = new MailMessage();

    mail.From = new MailAddress(Locator.GetService<EmailSettingsModule>().Settings.SenderEMail,
fromname);
    mail.To.Add(new MailAddress(to));
    mail.Subject = subject;
    mail.IsBodyHtml = true;
    mail.Body = body;

    try
    {
        PublicAPI.Services.Email.SendMessage(mail, null, null);
        return true;
    }
    catch (Exception ex)
    {
        Logger.Log.Error("Ошибка при отправке почты: " + ex.Message);
        return false;
    }
    finally
    {
        mail.Dispose();
    }
}
```

Для хранения всей истории переписки в рамках заказа в объект **Заказ** добавлен блок **История переписки** (Рис. 44).

История переписки	MailHistory	Блок
Входящее	IsInput	Да / нет
Дата и время	DateTime	Дата / время
Текст письма	HtmlMessage	HTML разметка

Рис. 44. Блок «История переписки» объекта «Заказ»

Метод **AddOrderHistory** осуществляет добавление письма к истории переписки в **Заказе (Order)**, на входе принимает **Заказ (Order)**, тело письма и атрибут типа **boolean**, который указывает на тип письма: Входящее или Исходящее. Метод на выходе также возвращает **True**, если операция выполнена успешно, и **False**, если операция не удалась.

```
public bool AddOrderHistory(Order order, string body, bool isInput)
{
    var order_history = InterfaceActivator.Create<Order_MailHistory>();
    order_history.DateTime = DateTime.Now;
    order_history.HtmlMessage = new HtmlString(body);
    order_history.IsInput = isInput;
    order_history.Save();
    order.MailHistory.Add(order_history);
    order.Save();
    return true;
}
```

Важно: после создания/изменения и публикации сценариев в глобальном модуле требуется перезагрузка сервера.

Создадим пользовательское расширение **Смена статуса заказа**, которое будет использовать методы глобального модуля **Helper**. Для этого следует в Дизайнере ELMA перейти на вкладку **Сценарии**, открыть блок **Пользовательские расширения** (Рис. 46) и нажать на кнопку **Добавить** или выбрать соответствующий пункт в контекстном меню.

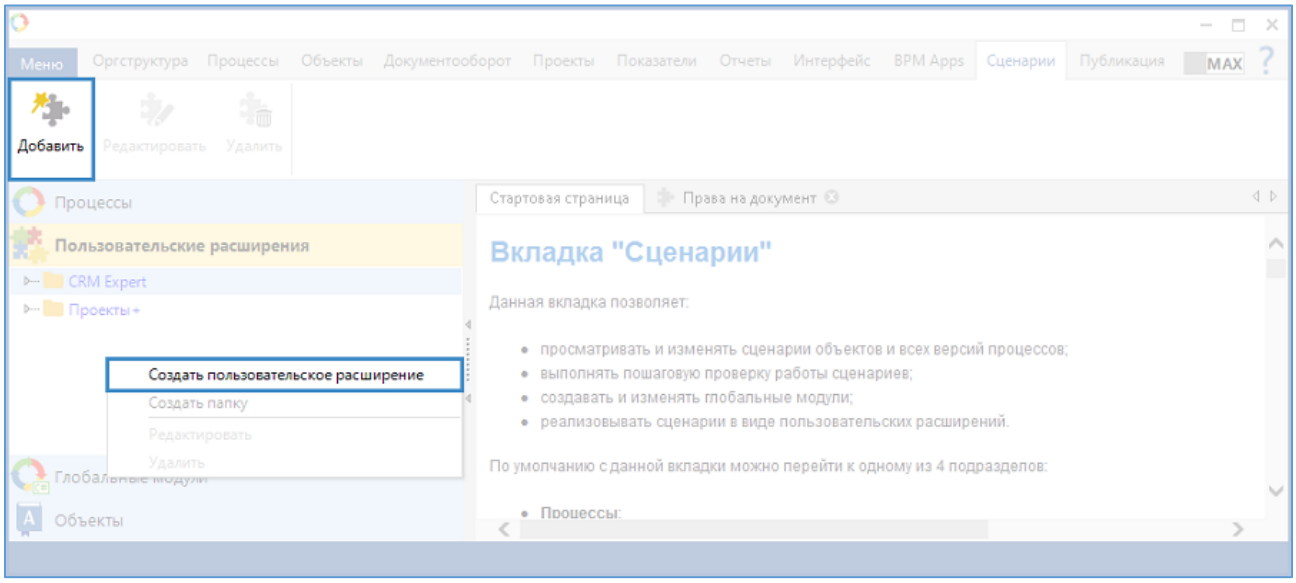


Рис. 45. Создание пользовательского расширения

В открывшемся окне (Рис. 46) необходимо заполнить требуемые поля и нажать на кнопку **Создать**.

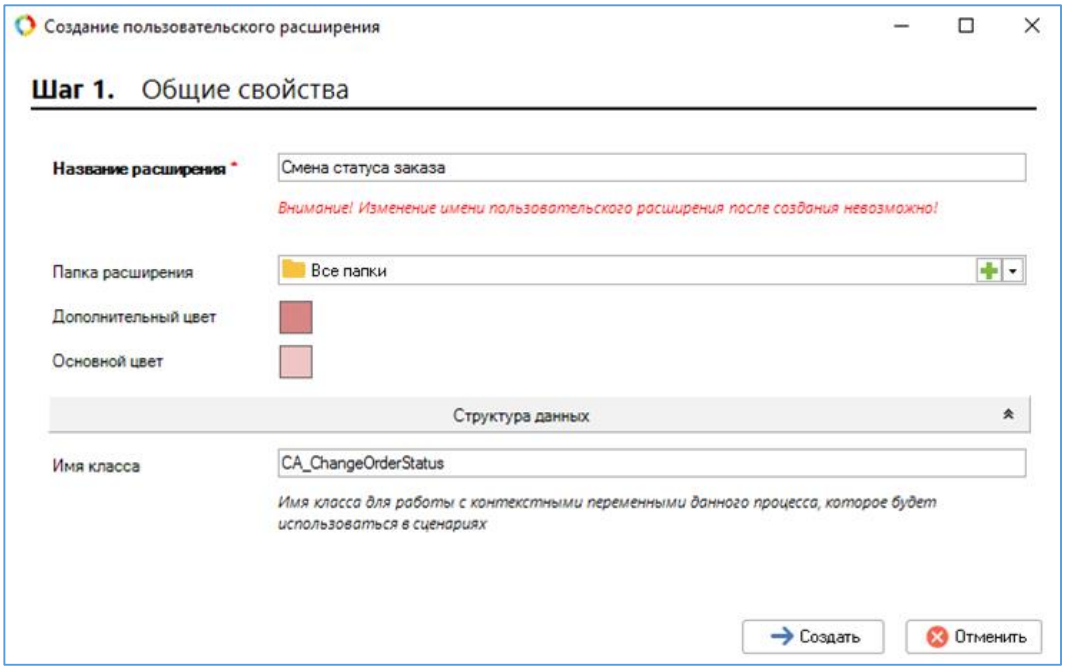


Рис. 46. Диалоговое окно создания пользовательского расширения

Будет открыта карточка созданного пользовательского расширения. Необходимо перейти на вкладку **Параметры** (Рис. 47) и добавить требуемые свойства. Входные и выходные параметры задаются при размещении блока на карте бизнес-процесса и могут быть сопоставлены с контекстными атрибутами, а также заданы вручную.

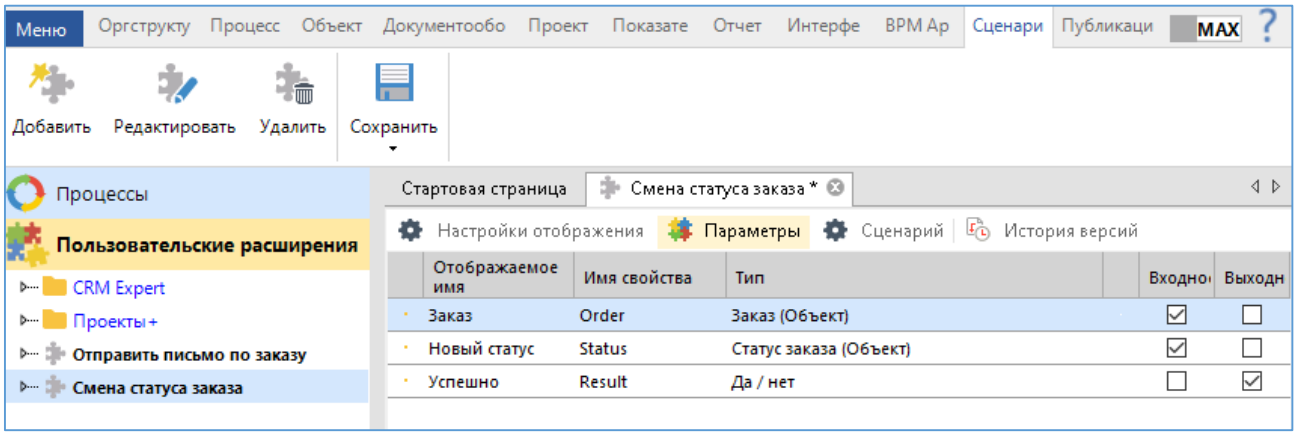


Рис. 47. Параметры пользовательского расширения Смена статуса заказа

Далее перейдем на вкладку **Сценарий** (Рис. 49) и реализуем сценарий на языке C#. Для того, чтобы в сценарии можно было вызывать методы глобального модуля **Helper**, его потребуется подключить.

Для этого необходимо на панели **Настройки** нажать на кнопку **Добавить** и выбрать пункт **Добавить ссылку на сборку** (Рис. 48).

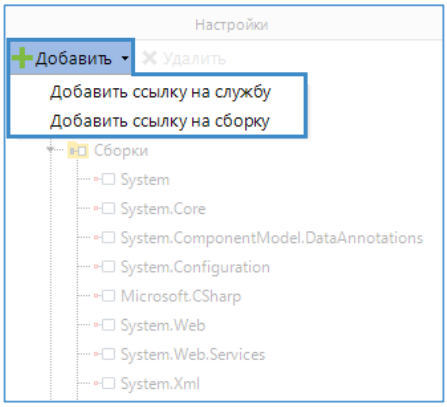


Рис. 48. Панель "Настройки". Кнопка "Добавить"

В открывшемся окне (Рис. 49) на вкладке **Глобальные модули** следует выбрать необходимую сборку и нажать кнопку **ОК**.

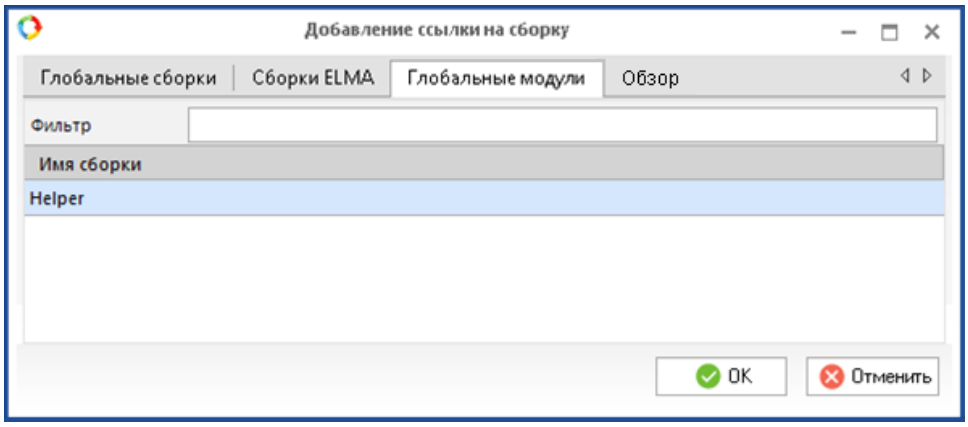


Рис. 49. Подключение глобального модуля

Для корректной работы сценария должны быть подключены следующие пространства имен:

```
using System.Linq;
using EleWise.ELMA.API;
```

В отличие от сценария, используемого в бизнес-процессе, в пользовательском расширении вместо **context.Order** требуется обращаться к атрибуту **Заказ** с помощью **parameters.Order**.

```
public void Execute(Parameters parameters)
{
    parameters.Result = false;
    var currentUser = PublicAPI.Portal.Security.User.GETCurrentUser();
    if (currentUser == null)
    {
        return;
    }

    // Для вызова методов из глобального модуля
    var helper = new Helper.Scripts();
```

```
    // Изменение статуса заказа
    parameters.Order.Status = parameters.Status;
    // Формирование темы и тела письма, его отправка
    var currentusername = currentUser.FullName;
    var subject = "Информация по Вашему заказу №" + parameters.Order.Number;
    var body = "<h1>Ваш заказ №" + parameters.Order.Number + " " + parameters.Order.Status.Name +
"</h1>";
    body += "<h2>Параметры заказа: </h2>";
    var cnt = 0;
    body +=
"<table><tr><th>№</th><th>Артикул</th><th>Цена</th><th>Количество</th><th>Сумма</th></tr>";
    foreach (var code in parameters.Order.Product)
    {
        cnt++;
        body += "<tr><td>" + cnt + "</td>" +
            "<td>" + code.Product.Name + "</td>" +
            "<td>" + code.Price + "</td>" +
            "<td>" + code.Quantity.ToString() + "</td>" +
            "<td>" + code.Total.ToString() + "</td>" +
            "</tr>";
    }
    body += "</table>";

    var contractorEmail = parameters.Order.Contractor.Email.FirstOrDefault();
    if (contractorEmail != null)
    {
        // Отправка письма
        if (helper.SendMail(currentusername, contractorEmail.ToString(), subject, body))
        {
            var historybody = "Ваш заказ №" + parameters.Order.Number + " " +
parameters.Order.Status.Name;
            // Добавить письмо к истории переписки
            helper.AddOrderHistory(parameters.Order, historybody, false);
            parameters.Result = true;
        }
    }
}
```

После создания или внесения изменений требуется опубликовать вновь созданное пользовательское расширение. Публикация пользовательского расширения не требует перезагрузки сервера и его можно сразу использовать при моделировании бизнес-процессов. Подробнее о данном функционале см. в соответствующей статье Базы знаний ELMA.

Откроем бизнес-процесс «Обработка заказа», разместим в нужных местах на карте пользовательское расширение **Смена статуса заказа** и произведем его настройку (Рис. 50).

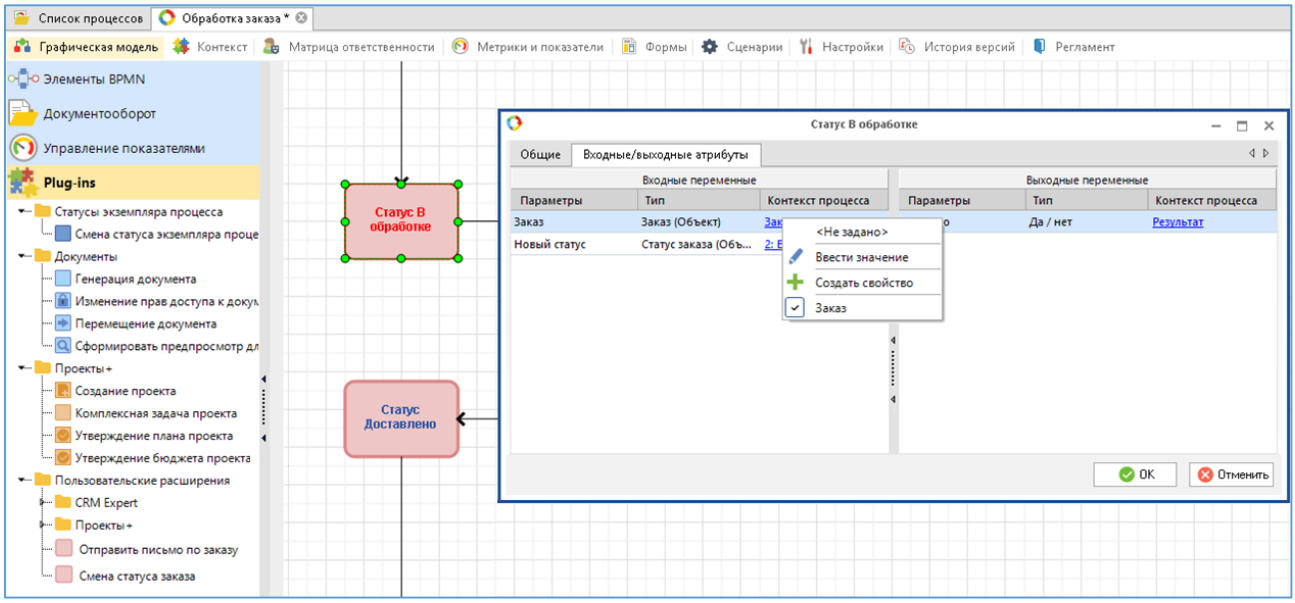


Рис. 50. Использование и настройка пользовательского расширения «Смена статуса заказа»

Требуется заполнить данные (Рис. 51) по статусам заказов, чтобы пользовательское расширение могло изменять их.

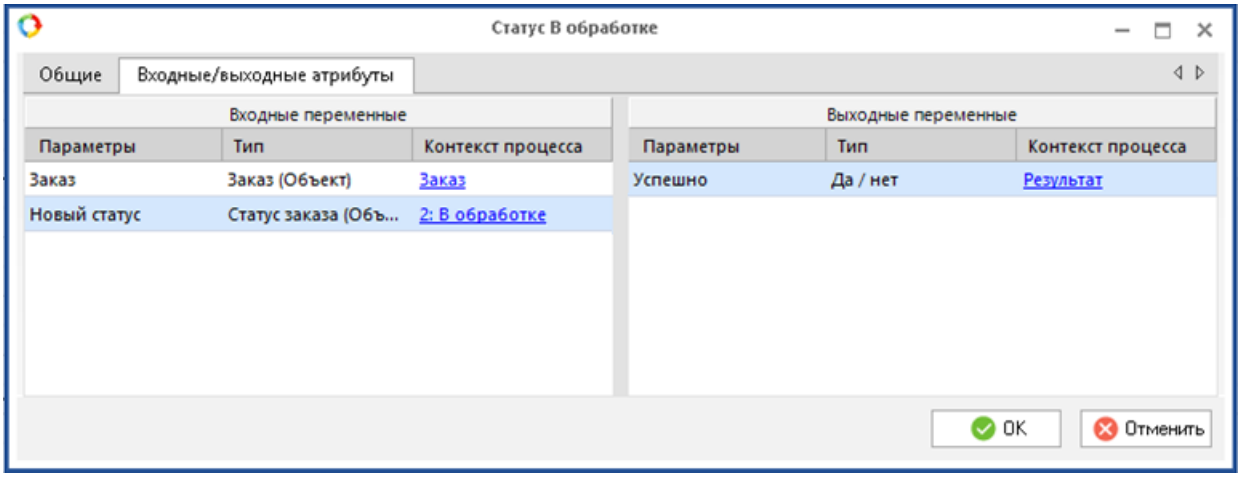


Рис. 51. Диалоговое окно настройки пользовательского расширения

Для этого при вводе параметров (Рис. 50) следует нажать на ссылку **Ввести значение**, после чего в открывшемся диалоговом окне (Рис. 52) ввести идентификатор статуса заказа и нажать на кнопку **ОК**. Корректность введенного идентификатора может быть проверена путем нажатия на кнопку **Проверить**.

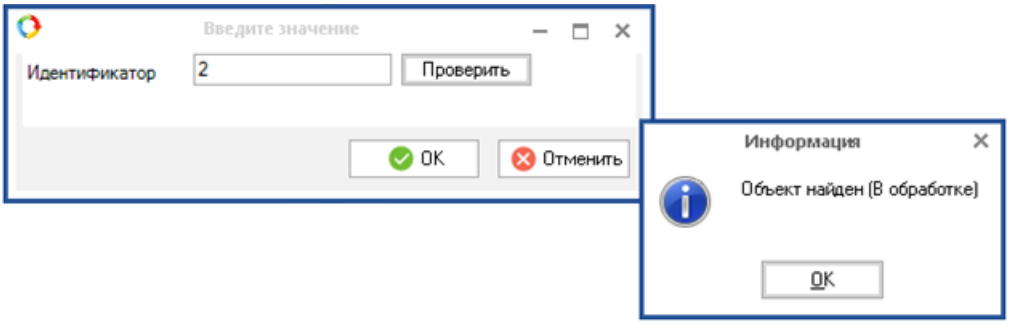


Рис. 52. Ввод и проверка идентификатора статуса заказа

Глава 6. Получение e-mail в ELMA

Одной из востребованных задач интеграции является обработка поступающих писем, их анализ и последующее создание активностей по ним. Применительно к деятельности интернет-магазина, реализуем несколько служебных процессов.

Бизнес-процесс «Проверка почты» (см. **Раздел 6.1**) будет с определенной периодичностью опрашивать почтовый сервер на наличие новых писем. При наличии новых писем будет производиться их проверка на соответствие следующим критериям:

- тема письма содержит номер заказа;
- адрес отправителя совпадает с адресом контрагента по данному заказу.

Если полученное письмо удовлетворяет данным критериям, то запускается внешний бизнес-процесс «Письмо по заказу» (см. **Раздел 6.2**), в рамках которого менеджер получает оповещение через систему и возможность написать ответное письмо, после чего обновляется история заказа.

Важно отметить, что менеджер при выполнении своей задачи (Рис. 53) располагает всей требуемой информацией о заказе, а также историей переписки с клиентом.

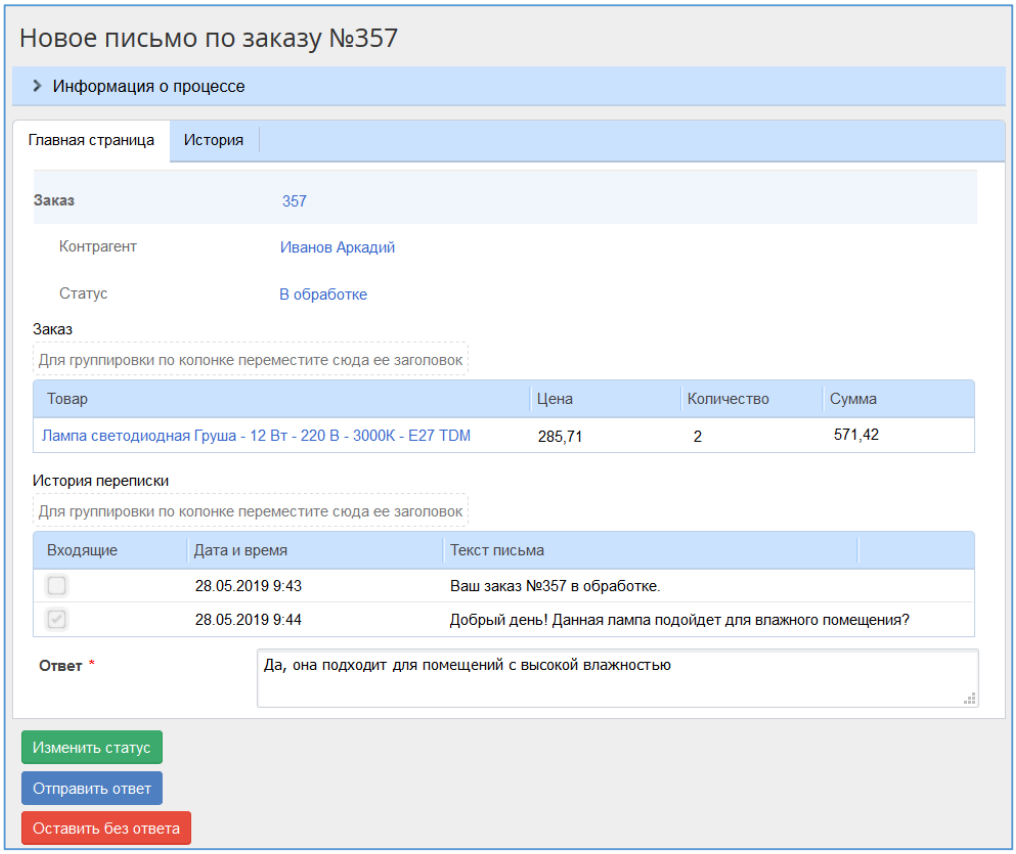


Рис. 53. Форма пользовательской задачи менеджера

6.1. Бизнес-процесс «Проверка почты»

Служебный бизнес-процесс «Проверка почты» реализуем при помощи встроенной в ELMA библиотеки Aspose.Email for .NET.

Карта данного бизнес-процесса представлена на Рис. 54.

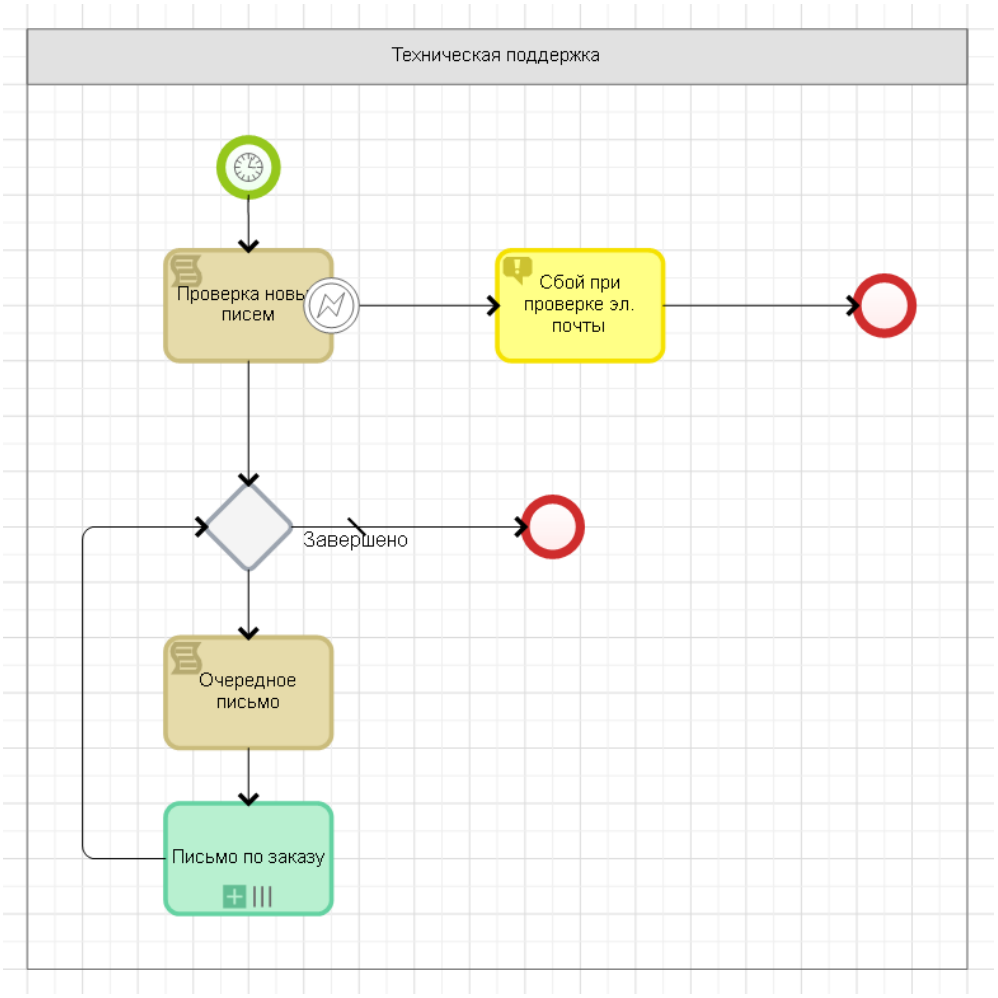


Рис. 54. Бизнес-процесс «Проверка почты»

На Рис. 55 представлен список контекстных переменных, используемых в данном бизнес-процессе.

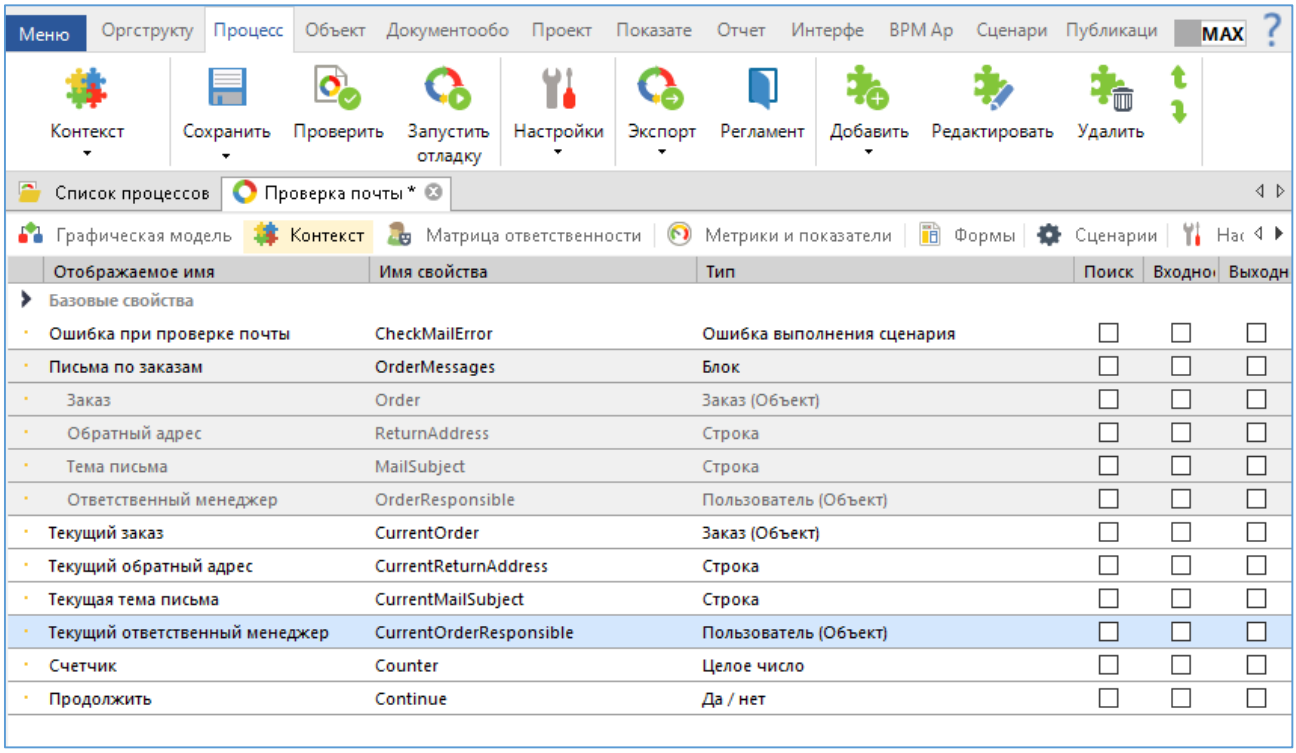


Рис. 55. Контекст процесса «Проверка почты»

Требуется, чтобы данный бизнес-процесс автоматически запускался по таймеру каждые 5 минут. Для этого осуществим настройку стартового события – на вкладке **Общие** установим триггер в положение **Таймер** (Рис. 56).

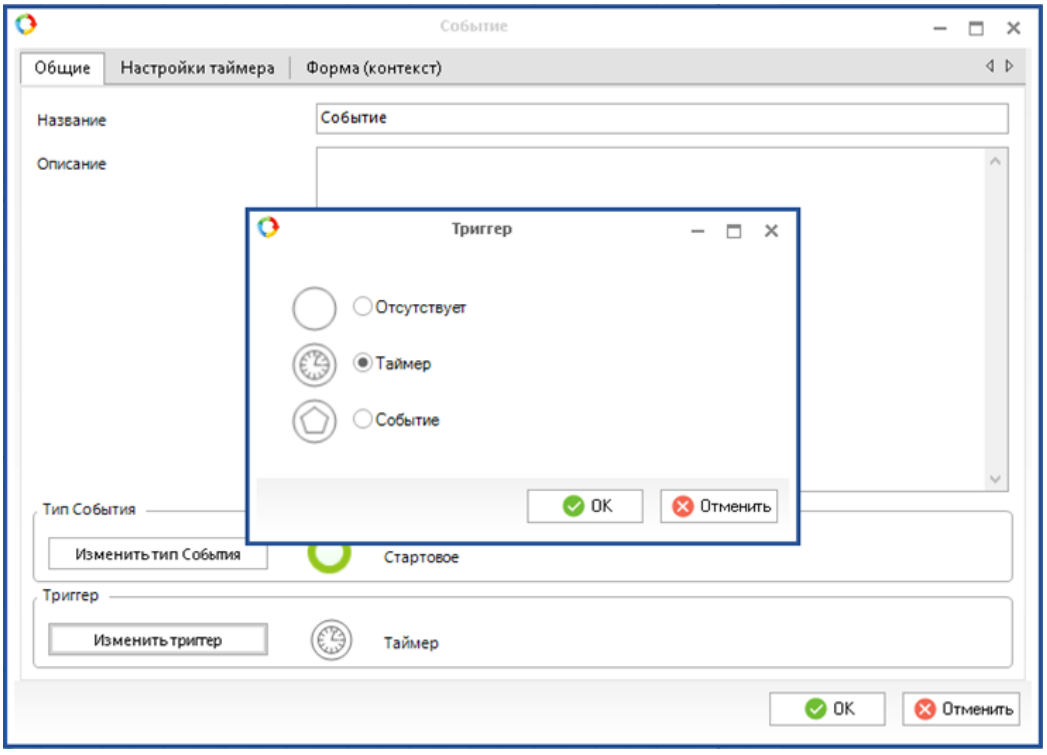


Рис. 56. Настройка триггера стартового события

Далее перейдем на вкладку **Настройки таймера** и установим требуемые значения (Рис. 57).

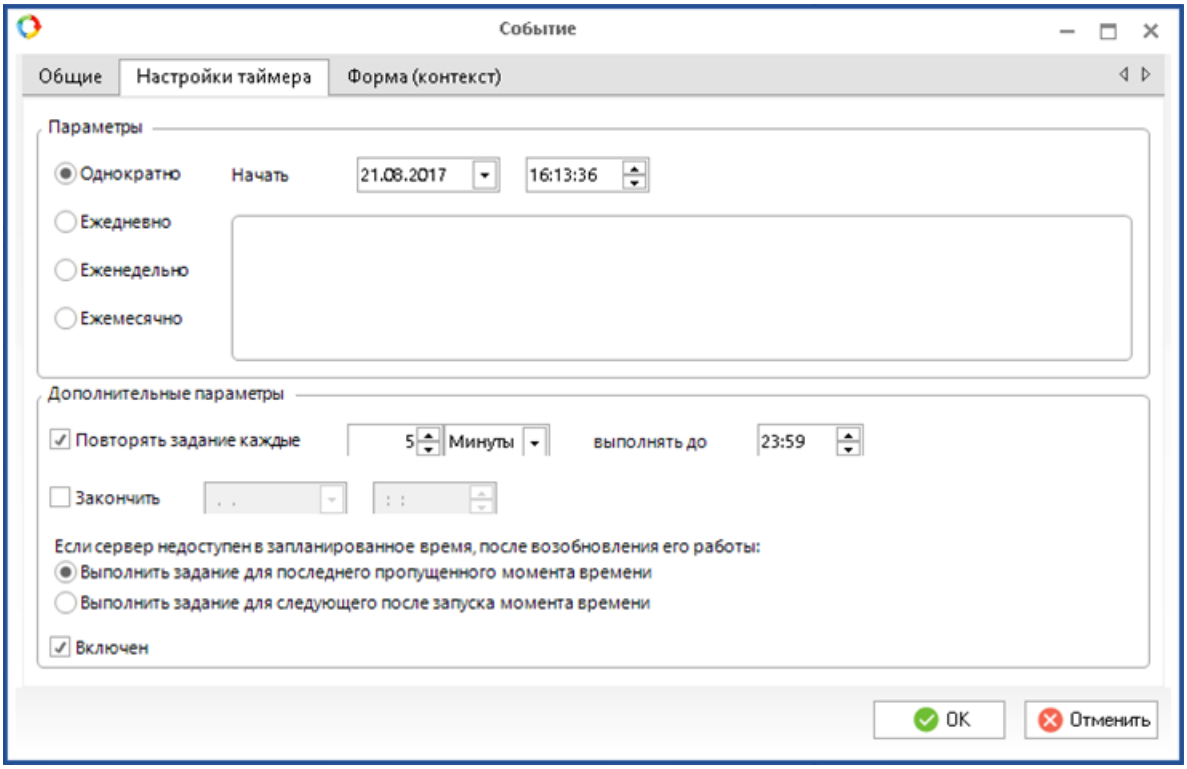


Рис. 57. Настройка стартового события. Вкладка «Настройки таймера»

После стартового события следует блок сценария **Проверка новых писем** (CheckMail) (Рис. 54). В данном блоке устанавливается соединение с почтовым сервером по заранее заданным параметрам. При наличии таковых производится их проверка на соответствие следующим критериям:

- тема письма содержит номер заказа;
- адрес отправителя совпадает с адресом контрагента по данному заказу.

Если получено письмо, удовлетворяющее данным критериям, с помощью вызова метода **AddOrderHistory** из глобального модуля **Helper** обновляется история данного заказа, а атрибуты данного письма заносятся в переменную **Письма по заказам** (с типом данных «Блок»). Для вызова методов глобального модуля **Helper** его следует подключить (Рис. 49).

Для корректной работы сценария требуется использование следующих сборок:

```
using System;
using System.Linq;
using Aspose.Email.Imap;
using Aspose.Email;
using EleWise.ELMA.Model.Managers;
using EleWise.ELMA.Model.Services;
using EleWise.ELMA.Model.Entities.ProcessContext;
```

```
using EleWise.ELMA.Services;
using EleWise.ELMA.CRM.Models;
using EleWise.ELMA.ConfigurationModel;
using EleWise.ELMA.Logging;
```

После объявления класса модуля сценариев следует указать реквизиты подключения к почтовому серверу:

```
    // Параметры почтового сервера
const string host = "imap.mail.ru";
const int port = 993;
const string username = "testuser@mail.ru";
const string password = "cegthgfhjkm";
```

Текст сценария «Проверка новых писем»:

```
public virtual void CheckMail(Context context)
{
    // Подключение методов глобального модуля
    var helper = new Helper.Scripts();
    var messages = Enumerable.Empty<Aspose.Email.Mail.MailMessage>();

    // Попытка подключения
    using(var client = new ImapClient(host, port, username, password)
    {
        SecurityOptions = SecurityOptions.SSLImplicit
    })
    {
        try
        {
            // Подключаемся и выбираем папку "Входящие"
            client.SelectFolder(ImapFolderInfo.InBox);
            messages = client.ListMessages()
                .Where(msg => !msg.IsRead)
                .Select(msg => client.FetchMessage(msg.SequenceNumber));
        }
        catch (Exception ex)
        {
            Logger.Log.Error("Ошибка при получении почты: " + ex.Message);
        }
    }

    var contractorFilter = InterfaceActivator.Create<ContractorFilter>();
    var contractorManager = EntityManager<Contractor>.Instance;
    var orderFilter = InterfaceActivator.Create<OrderFilter>();
    var orderManager = EntityManager<Order>.Instance;

    foreach (var message in messages)
    {
        // Поиск адреса отправителя в базе контрагентов
        contractorFilter.Email = message.From.Address;
        var contractors = contractorManager.Find(contractorFilter, null);

        orderFilter.Query = string.Format("Contractor in ({0})", string.Join(",",
contractors.Select(x => x.Id)));
        var ordersInSubject = orderManager
            .Find(orderFilter, null)
            .Where(o => message.Subject.Contains(o.Number));

        foreach (var order in ordersInSubject)
        {
            Logger.Log.Debug("Найдено письмо от " + message.From.Address + " по заказу " +
order.Number);
```

```
            // Оставим только ответную часть письма
            string messageBodyUTF8 = message.HtmlBody;
            var pos = messageBodyUTF8.IndexOf(username);
            messageBodyUTF8 = messageBodyUTF8.Remove(pos);
            pos = messageBodyUTF8.LastIndexOf("<div>");
            messageBodyUTF8 = messageBodyUTF8.Remove(pos);

            // Добавим письмо к блоку Письма по заказам
            var orderMessage = InterfaceActivator.Create<P_CheckMail_OrderMessages>();
            orderMessage.MailSubject = "Re: " + message.Subject;
            orderMessage.Order = order;
            orderMessage.ReturnAddress = message.From.Address;
            if (order.Contractor.Responsible != null)
            {
                orderMessage.OrderResponsible = order.Contractor.Responsible;
            }
            context.OrderMessages.Add(orderMessage);

            // Добавление записи к истории взаимодействия
            helper.AddOrderHistory(order, messageBodyUTF8, true);
        }
    }

    if (context.OrderMessages.Count > 0)
    {
        context.Counter = 0;
        context.Continue = true;
    }
}
```

В бизнес-процессе следует предусмотреть обработку ошибок сценария **Проверка новых писем**. Для этого из блока сценария реализован исходящий переход в блок **Оповещение** (Рис. 54).

Двойным нажатием по данному переходу следует открыть его настройки и на вкладке **Общие** в блоке **Эскалация** выбрать тип эскалации **Обработка ошибки** (Рис. 58).

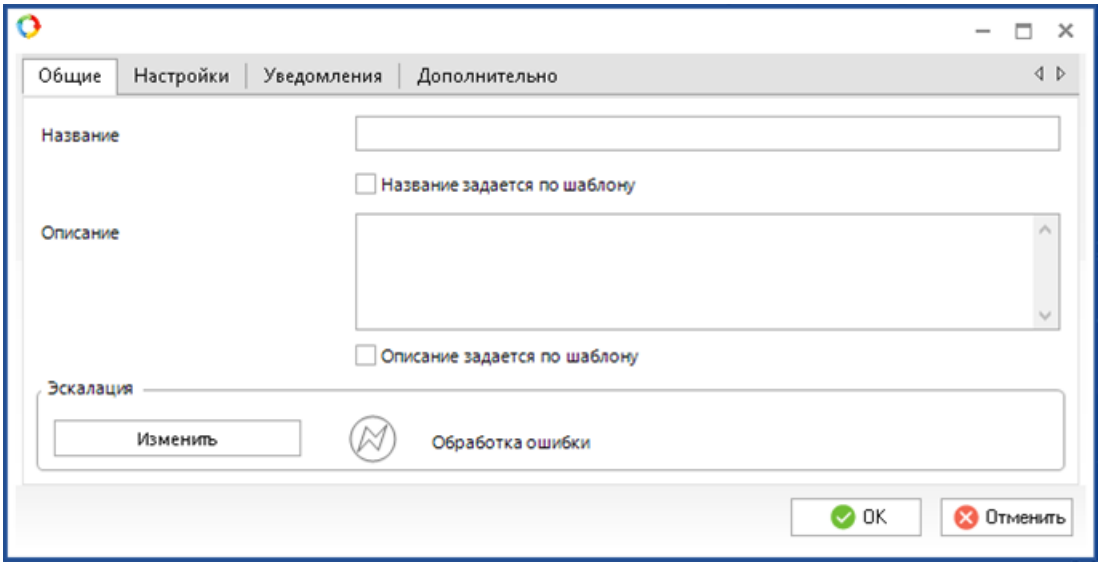


Рис. 58. Диалоговое окно настройки перехода. Вкладка «Общие»

После этого на вкладке **Настройки** (Рис. 59) требуется указать контекстную переменную для записи данной ошибки.

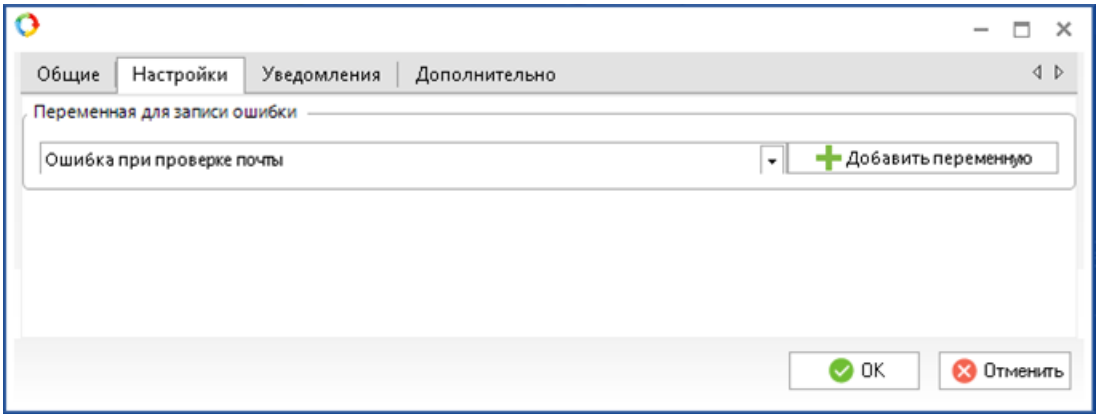


Рис. 59. Диалоговое окно настройки перехода. Вкладка «Настройки»

Настройка блока **Сбой при проверке эл. почты** заключается в добавлении на соответствующих вкладках шаблона оповещения (Рис. 60) и указании получателя (Рис. 61).

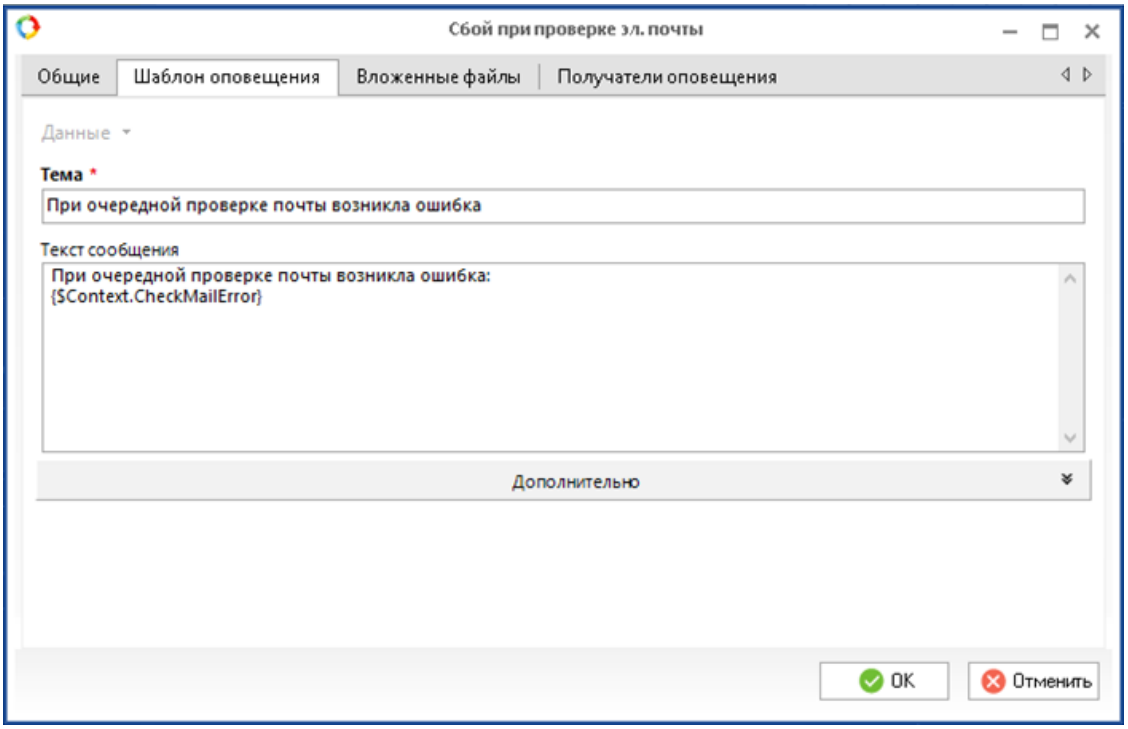


Рис. 60. Диалоговое окно настройки оповещения. Вкладка «Шаблон оповещения»

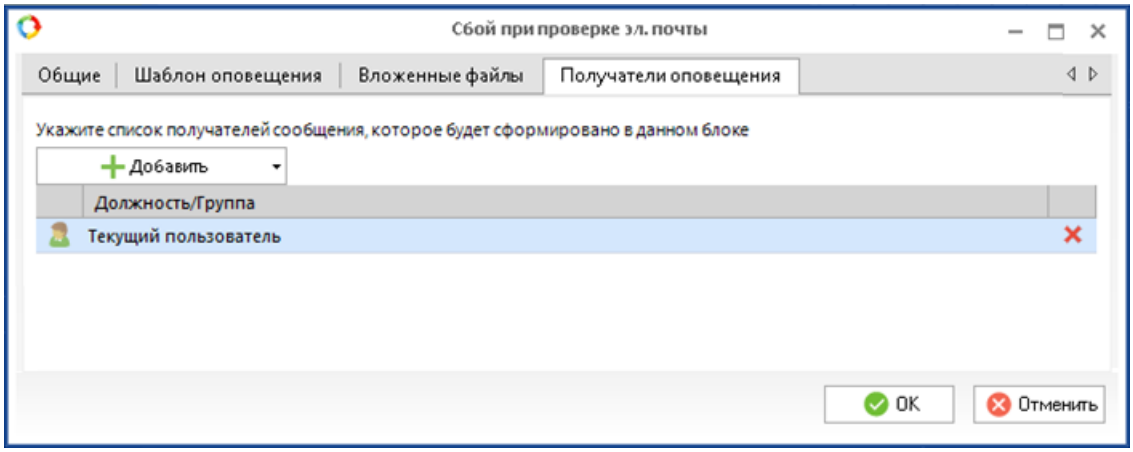


Рис. 61. Диалоговое окно настройки оповещения. Вкладка «Получатели оповещения»

Данное оповещение (как и весь процесс) выполняется в статической зоне ответственности **Техническая поддержка** (Рис. 62).

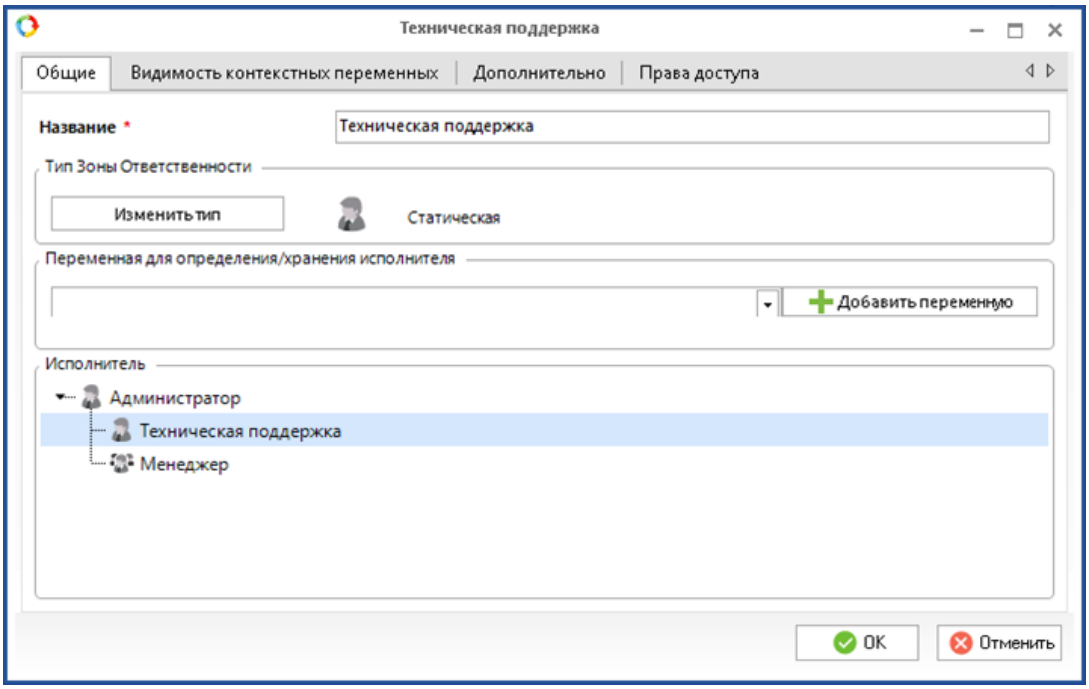


Рис. 62. Диалоговое окно настройки статической зоны ответственности

Следующий этап бизнес-процесса – это настройка Исключающего ИЛИ-шлюза. Нижний переход шлюза (Рис. 54) срабатывает, если контекстная переменная **Продолжить (Continue)** имеет значение **true**, иначе – процесс продолжит движение по правому исходящему переходу (переходу по умолчанию).

Значение данной переменной инициализируется на первом шаге бизнес-процесса «Проверка почты» (Рис. 54). Значение **true** означает, что переменная **Письма по заказу (OrderMessages)** (типа **Блок**) не пустая.

Следующий шаг, который выполняется при наличии писем, это сценарий **Очередное письмо (NextMessage)**.

```
public virtual void NextMessage(Context context)
{
    var message = context.OrderMessages.ToArray()[context.Counter];
    context.CurrentOrder = message.Order;
    context.CurrentMailSubject = message.MailSubject;
    context.CurrentReturnAddress = message.ReturnAddress;
    context.Counter++;

    if (context.Counter == context.OrderMessages.Count)
    {
        context.Continue = false;
    }
}
```

Назначение данного сценария – получить очередное письмо из блока **Письма по заказу** и записать его значения в контекстные переменные (Рис. 63), которые затем передаются в качестве входных переменных для запуска внешнего подпроцесса **Письмо по заказу**.

Текущий заказ	CurrentOrder	Заказ (Объект)	☐	☐	☐
Текущий обратный адрес	CurrentReturnAddress	Строка	☐	☐	☐
Текущая тема письма	CurrentMailSubject	Строка	☐	☐	☐
Текущий ответственный менеджер	CurrentOrderResponsible	Пользователь (Объект)	☐	☐	☐

Рис. 63. Контекстные переменные для запуска внешнего подпроцесса

6.2. Бизнес-процесс «Письмо по заказу»

Пример интерфейса первой задачи менеджера приведен на Рис. 53.

Карта бизнес-процесса «Письмо по заказу» представлена на Рис. 64. Данный бизнес-процесс выполнен без использования сценариев.

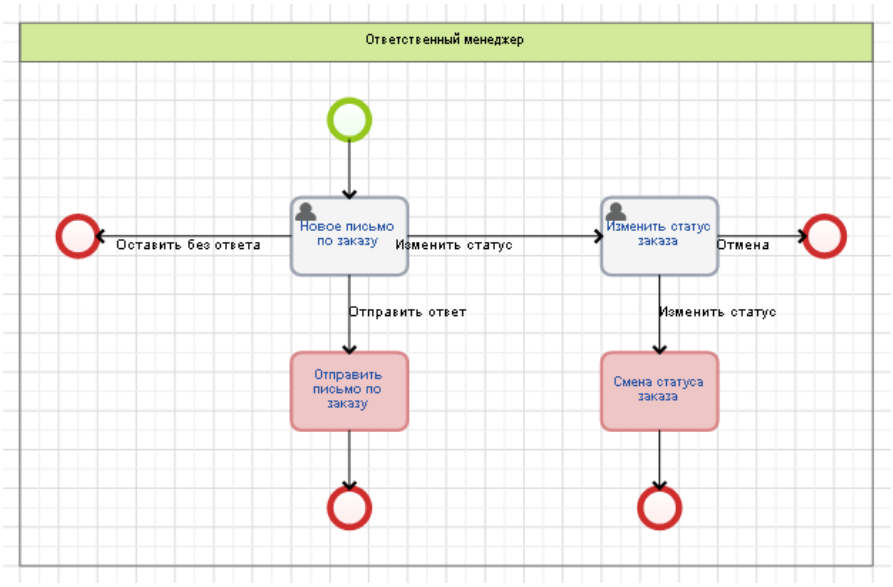


Рис. 64. Бизнес-процесс «Письмо по заказу»

Для корректного запуска данного бизнес-процесса потребуется установить флажок **Входное** на атрибутах (Рис. 65), которые передаются из бизнес-процесса **Проверка почты** (см. выше).

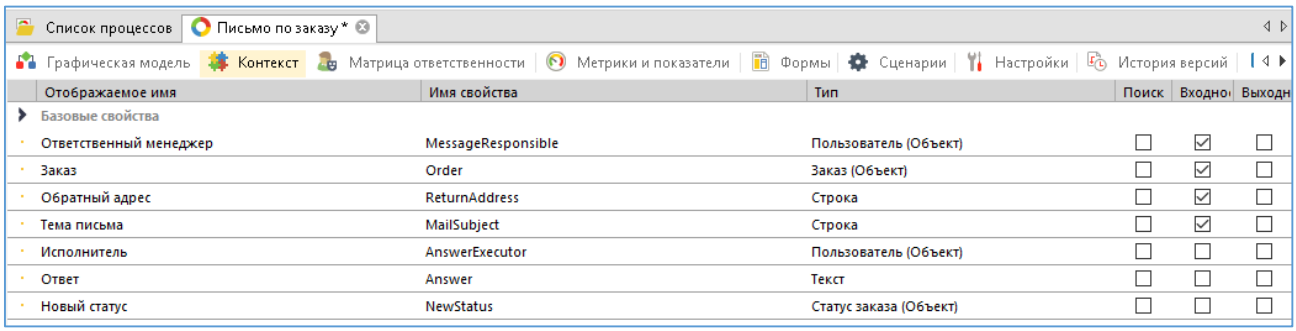


Рис. 65. Контекст бизнес-процесса «Письмо по заказу»

Подробнее о контексте процесса см. в справке по системе ELMA.

Глава 7. Работа с веб-сервисами из системы ELMA

7.1. Использование веб-сервиса на примере получения курсов валют

Для задачи обработки информации в процессе «Оформление заказа» требуется конвертировать стоимость заказа в валюте в рубли. Учитывая тот факт, что изменения валюты на рынке происходят практически каждый день, то задача актуализации оказывается достаточно трудоемкой. Для упрощения задачи можно реализовать интеграцию с сервисом для получения ежедневных данных (курсы валют, учетные цены драг. металлов и др.) Центрального банка Российской Федерации (ЦБ РФ). Описание данного сервиса доступно здесь.

Сервис ЦБ РФ описан по правилам WSDL (Web Services Description Language). Данное описание основано на языке XML и представляет из себя структуру запросов/ответов сервера и форматов данных. Дизайнер ELMA позволяет инициализировать правильно описанный WSDL как новый класс в рамках модуля сценариев. При этом использовать такой сервис можно без необходимости дополнительных инициализаций веб-запросов.

Для актуализации курса валют разработаем служебный бизнес-процесс «Актуализация курса валют», который будет обновлять данные в справочнике **Курс валют**. Бизнес-процесс будет запускаться по таймеру один раз в день от имени администратора системы.

На схеме бизнес-процесса (Рис. 66) будет всего один элемент – сценарий получения/обновления данных курса валют.

Рис. 66. Бизнес-процесс «Актуализация курса валют»

Далее необходимо добавить сценарий (Рис. 67) в блок **Актуализация курса валют**, в котором будет реализована интеграция с сервисом ЦБ РФ и логика актуализации справочника **Курс валют**.

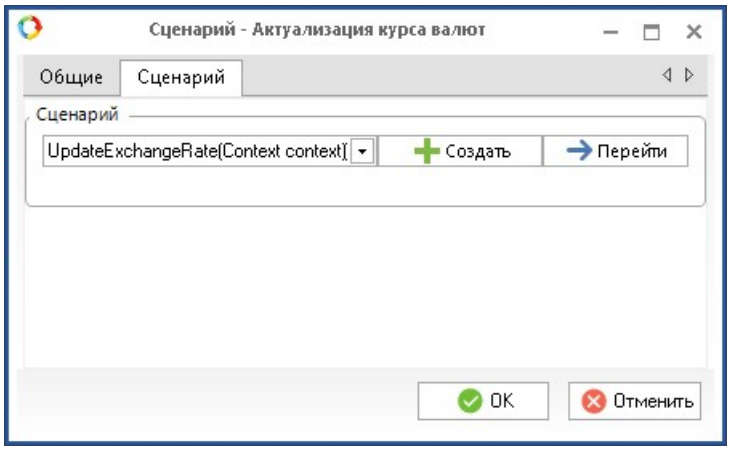


Рис. 67. Добавление сценария в бизнес-процесс

Первым делом для реализации интеграции с сервисом ЦБ ФР необходимо перейти в созданный сценарий и в правой части окна подключить ссылку на сервис через выбор пункта контекстного меню **Добавить ссылку на службу** (Рис. 48).

После этого откроется диалоговое окно, в котором необходимо указать адрес до веб-сервиса, нажать на кнопку **Перейти** и задать необходимое пространство имен для импортируемого класса (Рис. 68).

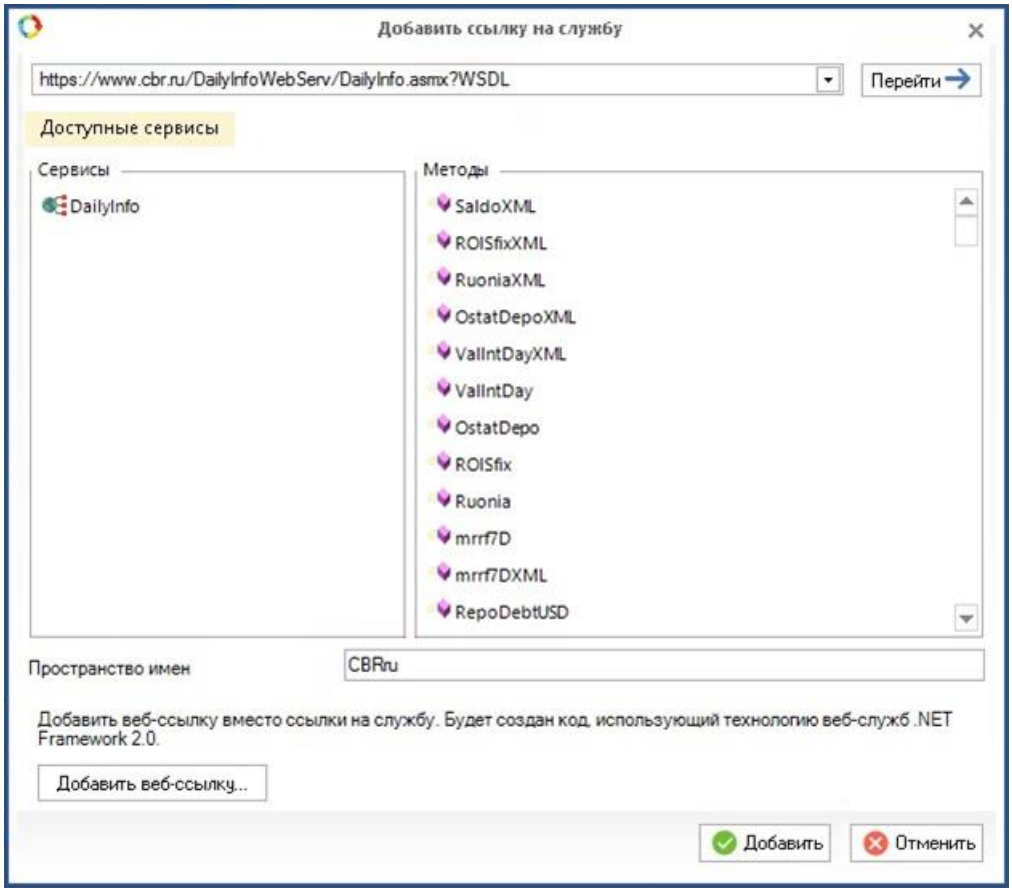


Рис. 68. Настройки импорта WS-ссылки

После нажатия на кнопку **Перейти** можно также увидеть, какие методы реализованы на стороне сервиса. Пространство имен не должно совпадать с уже существующими в системе.

Для работы необходимо использовать следующие пространства имен:

```
using System.ServiceModel;
using System.Data;
```

Также конкретно для этого сервиса необходимо подключить сборку **System.Data**, так как в сервисе указана такая зависимость.

Опишем логику работы сценария актуализации справочника **Курс валюты**. Для начала необходимо создать экземпляр подключения к сервису ЦБ РФ и получить актуальные данные. Все методы сервиса для получения актуальных данных приведены на официальном сайте ЦБ РФ.

Сервис ЦБ РФ использует протокол SOAP, который позволяет работать со строго типизированными данными. В данном случае требуется получить актуальные курсы валют на текущую дату, для этого будем использовать метод **GETCursOnDate**. Метод **GETCursOnDate** является строго типизированным и возвращает результат типа

DataSet (System.Data). Тип ответа можно посмотреть в описании метода или после подключения веб-ссылки во всплывающей подсказке при наведении курсора на метод **GetCursOnDate**.

```
        // Создадим экземпляр подключения к сервису ЦБ РФ
        var newWFP = new CBRru.DailyInfo();
        //Используем метод сервиса, который возвращает информацию в виде DataSet –
        // это структура, содержащая набор таблиц данных
        DataSet cursSet = newWFP.GetCursOnDate(DateTime.Now);
```

Таким образом мы получили данные по курсу валют к рублю на текущую дату. Теперь переменная **cursSet** хранит в себе таблицы с актуальными данными, которые необходимо записать в справочник **Курс валюты**.

Для этого запустим два цикла: один по полученным таблицам, другой (внутри первого цикла) – по строкам. Таким образом мы сможем получить доступ ко всем значениям таблицы. В теле второго цикла пропишем логику обновления справочника. Чтобы избежать дублирования записей, необходимо в справочнике **Курс валюты** найти запись, удовлетворяющую следующему условию: **Валюта курса**, у которой **Код ISO алфавитный** равен коду из строки таблицы и **Код ISO алфавитный Валюты к оценке** равен **'RUB'**.

Если запись найдена, то у нее необходимо перезаписать такие данные, как: **Дата котирования, Номинал, Курс**. В случае, если такая запись не найдена, то необходимо создать новую.

Для работы со следующим сценарием необходимо подключить пространства имен:

```
using EleWise.ELMA.Model.Services;
using EleWise.ELMA.CRM.Models;
```

Текст сценария:

```
// Создадим экземпляр подключения к сервису ЦБ РФ
    var newWFP = new CBRru.DailyInfo();

    //обычно в документации к сервису указано, в каких полях какие данные хранятся,
    //однако для примера разберем полный обход каждой таблицы из полученной структуры
    //загружаем в переменную currencyEstimated по Guid валюту Российский рубль
    var currencyEstimated = EntityManager<Currency>.Instance.Load(new Guid("C24F7F9B-0C36-43FA-
A044-6B12E2CC1CCC"));
    // словарь с курсами валют в виде ['id валюты', 'курс'] для поиска курсов обмена
    var currencyRates = EntityManager<ExchangeCurrencyRate>.Instance
        .Find(string.Format("CurrencyEstimated = {0}", currencyEstimated.Id))
        .ToDictionary(rate => rate.CurrencyValuation.Id);
    // словарь с валютами в виде ['код', 'курс'] для поиска курсов обмена
    var currencies =
EntityManager<Currency>.Instance.FindByIdArray(currencyRates.Keys.ToArray());
    var ratesByCurrencyCode = currencies.ToLookup(c => c.CodeISOAlphabetic, c =>
currencyRates[c.Id]);

    //создадим фильтр для поиска записи
    var filter = InterfaceActivator.Create<ExchangeCurrencyRateFilter>();
```

```
    //Используем метод сервиса, который возвращает информацию в виде DataSet – это структура,
содержащая набор таблиц данных
    using (DataSet cursSet = newWFP.GETCursOnDate(DateTime.Now))
    {
        foreach (var row in cursSet.Tables.SelectMany(t => t.Rows))
        {
            // находим курс по коду валюты, либо создаем новый
            var isoCode = row["VchCode"].ToString();
            var rate = ratesByCurrencyCode[isoCode].FirstOrDefault()
                ?? CreateExchangeCurrencyRate(row, currencyEstimated,
                    currencies.FirstOrDefault(c => c.CodeISOAlphabetic == isoCode));

            rate.ValuationDate = DateTime.Now;
            rate.Nominal = Convert.ToInt64(tRow["Vnom"]);
            rate.ExchangeRate = Convert.ToDouble(tRow["Vcurs"]);

            rate.Save();
        }
    }
```

Для удобства восприятия кода вынесем процедуру создания записи в справочнике **Курс валюты** в отдельный метод **CreateExchangeCurrencyRate**.

```
//метод для создания записи в справочнике Курс валюты
private ExchangeCurrencyRate CreateExchangeCurrencyRate(DataRow tRow, Currency currencyEstimated,
Currency currencyValuation)
{
    var exchangeCurrencyRate = EntityManager<ExchangeCurrencyRate>.Create();

    exchangeCurrencyRate.CurrencyEstimated = currencyEstimated;
    exchangeCurrencyRate.CurrencyValuation = currencyValuation;

    return exchangeCurrencyRate;
}
```

Текст сценария интеграции с ЦБ РФ:

```
/// <summary>
/// Актуализация справочника Курс валюты
/// </summary>
/// <param name="context">Контекст процесса</param>
public virtual void UpdateExchangeRate(Context context)
{
    // Создадим экземпляр подключения к сервису ЦБ РФ
    var newWFP = new CBRru.DailyInfo();

    //обычно в документации к сервису указано, в каких полях какие данные хранятся,
    //однако для примера разберем полный обход каждой таблицы из полученной структуры
    //загружаем в переменную currencyEstimated по Guid валюту Российский рубль
    var currencyEstimated = EntityManager<Currency>.Instance.Load(new Guid("C24F7F9B-0C36-43FA-
A044-6B12E2CC1CCC"));
    // словарь с курсами валют в виде ['id валюты', 'курс'] для поиска курсов обмена
    var currencyRates = EntityManager<ExchangeCurrencyRate>.Instance
        .Find(string.Format("CurrencyEstimated = {0}", currencyEstimated.Id))
        .ToDictionary(rate => rate.CurrencyValuation.Id);
    // словарь с валютами в виде ['код', 'курс'] для поиска курсов обмена
    var currencies =
EntityManager<Currency>.Instance.FindByIdArray(currencyRates.Keys.ToArray());
    var ratesByCurrencyCode = currencies.ToLookup(c => c.CodeISOAlphabetic, c =>
currencyRates[c.Id]);

    //Используем метод сервиса, который возвращает информацию в виде DataSet – это структура,
содержащая набор таблиц данных
    using (DataSet cursSet = newWFP.GETCursOnDate(DateTime.Now))
    {
        foreach (var row in cursSet.Tables.SelectMany(t => t.Rows))
        {
            // находим курс по коду валюты, либо создаем новый
            var isoCode = row["VchCode"].ToString();
            var rate = ratesByCurrencyCode[isoCode].FirstOrDefault()
                ?? CreateExchangeCurrencyRate(currencyEstimated,
                    currencies.FirstOrDefault(c => c.CodeISOAlphabetic == isoCode));

            rate.ValuationDate = DateTime.Now;
            rate.Nominal = Convert.ToInt64(tRow["Vnom"]);
            rate.ExchangeRate = Convert.ToDouble(tRow["Vcurs"]);

            rate.Save();
        }
    }
}


/// <summary>
/// Создание новой записи справочника Курс валюты на основе полученных данных из сервиса ЦБ РФ
/// </summary>
/// <param name="tRow"></param>
/// <param name="currencyEstimated"></param>
/// <returns></returns>
private ExchangeCurrencyRate CreateExchangeCurrencyRate(Currency currencyEstimated, Currency
currencyValuation)
{
    var exchangeCurrencyRate = EntityManager<ExchangeCurrencyRate>.Create();

    exchangeCurrencyRate.CurrencyEstimated = currencyEstimated;
    exchangeCurrencyRate.CurrencyValuation = currencyValuation;

    return exchangeCurrencyRate;
}
```

Дополнительно справочник **Валюта** можно автоматически актуализовать валютами из сервиса ЦБ РФ – эта логика есть в методе **GETCurrency**. Для того чтобы воспользоваться им отдельно (например, перед началом работ), можно в редактор сценариев вставить приведенный ниже сценарий и запустить эмуляцию. В веб-браузере откроется страница эмуляции сценариев, на которой необходимо заполнить поле **Сценарий**, в параметре **Откат после выполнения** установить значение **Нет** и запустить сценарий. После этого справочник **Валюта** заполнится всеми валютами из сервиса ЦБ РФ. После заполнения справочника сценарий можно удалить.

Текст сценария для автозаполнения справочника **Валюта:**

```
public virtual void UpdateCurrency(string VchCode, DataRow tRow)
{
    // Создадим экземпляр подключения к сервису ЦБ РФ
    var newWFP = new CBRru.DailyInfo();
    //Используем метод сервиса, который возвращает информацию в виде DataSet – это структура,
содержащая набор таблиц данных
    using (DataSet cursSet = newWFP.GETCursOnDate(DateTime.Now))
    {
        // словарь с данными о валютах в виде ['код', 'ряд данных']
        var currencies = cursSet.Tables
            .SelectMany(t => t.Rows)
            .ToDictionary(r => r["VchCode"].ToString());

        var existingCurrencyCodes = EntityManager<Currency>.Instance
            .FindAll()
            .Select(c => c.CodeISOAlphabetic)
            .ToHashSet();

        var missingCurrencyCodes = currencies.Keys.Where(code =>
!existingCurrencyCodes.Contains(code));

        foreach (var currencyCode in missingCurrencyCodes)
        {
            var newCurrency = InterfaceActivator.Create<Currency>();

            newCurrency.Name = currencies[currencyCode]["Vname"].ToString();
            newCurrency.CodeISOAlphabetic = currencyCode;

            newCurrency.Save();
        }
    }
}
```

7.2. Десериализация и запись в справочник

Не все сервисы используют SOAP-протоколы и возвращают данные в виде объектов конкретных классов. Чаще всего результатом может являться текст в формате XML или JSON. В этом случае для удобства работы рекомендуется десериализовать результат в свою промежуточную структуру классов и работать уже с ней. Рассмотрим пример десериализации ответа XML, полученного из сервиса ЦБ РФ, и актуализации справочника **Курс валюты**. Для начала аналогичным образом, как описано выше, необходимо в бизнес-процессе создать сценарий, подключить ссылку на сервис, сборку **System.Data** и пространства имен:

```
using System.Xml;
using EleWise.ELMA.Serialization;
```

Создадим экземпляр подключения и вызовем метод **GETCursOnDateXML**. Данный метод возвращает ответ типа XmlNode. Тип ответа можно посмотреть в описании метода или после подключения веб-ссылки в всплывающей подсказке при наведении курсора на метод **GETCursOnDateXML**.

```
    // Создадим экземпляр подключения к сервису ЦБ РФ
    var newWFP = new CBRru.DailyInfo();
    // Метод GETCursOnDateXML позволяет вернуть курс в виде XML
    XmlNode cursSetXml = newWFP.GETCursOnDateXML(DateTime.Now);
```

Далее необходимо сформировать модель объекта десериализации. Это можно сделать вручную или автоматически с помощью онлайн-приложений, конвертирующих ответ XML в классы. Для формирования класса необходимо получить ответ сервиса или взять готовый пример, который может присутствовать в описании метода. Ответ сервиса можно вывести в консоль, добавив строку **Console.WriteLine(cursSetXml.OuterXml)**, и выполнить сценарий в режиме эмуляции. Полученный в консоли ответ необходимо вставить в приложение-конвертер и сгенерировать классы. Полученные классы необходимо вставить в сценарий и подключить пространство имен **using System.Xml.Serialization**. В данном случае модель объекта десериализации выглядит следующим образом:

```
     [XmlRoot(ElementName = "ValuteData")]
    public class CursOnDate
    {
        [XmlElement(ElementName = "ValuteCursOnDate")]
        public List<CurrencyCurs> ValuteCursOnDate { get; set; }

        [XmlAttribute(AttributeName = "OnDate")]
        public string OnDate { get; set; }

        [XmlAttribute(AttributeName = "xmlns")]
        public string Xmlns { get; set; }
    }

    [XmlRoot(ElementName = "ValuteCursOnDate")]
```

```
    public class CurrencyCurs
    {
        [XmlElement(ElementName = "Vname")]
        public string Vname { get; set; }

        [XmlElement(ElementName = "Vnom")]
        public string Vnom { get; set; }

        [XmlElement(ElementName = "Vcurs")]
        public string Vcurs { get; set; }

        [XmlElement(ElementName = "Vcode")]
        public string Vcode { get; set; }

        [XmlElement(ElementName = "VchCode")]
        public string VchCode { get; set; }
    }
```

Следующим этапом будет десериализация. Для этого подключим пространство имен **using EleWise.ELMA.Serialization**. Для десериализации ответа будем использовать типизированный сериализатор XML **XmlSerializer**. Данный класс позволяет сериализовать объект в XML, и наоборот – десериализовать XML в объект. В данном случае будем использовать метод десериализации. В типизированном сериализаторе XML укажем модель объекта, в который будем десериализовать полученные данные XmlSerializer**<ValuteData>**, а в его методе десериализации – полученный ответ от сервиса, который необходимо десериализовать **Deserialize(cursSetXml.OuterXml)**. В результате получится следующая строка:

```
var obj = XmlSerializer<ValuteData>.Deserialize(cursSetXml.OuterXml);
```

Таким образом мы десериализовали полученные на текущую дату данные по курсу валют к рублю в описанную нами модель. Теперь переменная **obj** хранит в себе список валют с актуальными курсами, которые необходимо записать в справочник **Курс валюты**.

Логика сценария ничем не отличается от примера, описанного в предыдущей главе. Для работы со следующим сценарием необходимо подключить пространства имен:

```
using EleWise.ELMA.Model.Services;
using EleWise.ELMA.CRM.Models;
```

Ниже приведен текст сценария десериализации ответа типа XML и актуализации справочника **Курс валюты**. Описание класса для десериализации идет после основного класса сценариев процесса:

```
public partial class P_UpdateExchangeRateXML_Scripts :
EleWise.ELMA.Workflow.Scripts.ProcessScriptBase<Context>
{
    /// <summary>
```

```
    /// vam
    /// </summary>
    /// <param name="context">Контекст процесса</param>
    public virtual void UpdateExchangeRateXML(Context context)
    {
        // Создадим экземпляр подключения к сервису ЦБ РФ
        var newWFP = new CBRru.DailyInfo();
        //метод GETCursOnDateXML позволяет вернуть курс в виде XML
        XmlNode cursSetXml = newWFP.GETCursOnDateXML(DateTime.Now);
        Console.WriteLine(cursSetXml.OuterXml);
        // Десериализация
        var obj = XmlSerializer<ValuteData>.Deserialize(cursSetXml.OuterXml);
        // Загружаем в переменную currencyEstimated по Guid валюту Российский рубль
        var currencyEstimated = EntityManager<Currency>.Instance.Load(new Guid("C24F7F9B-0C36-
43FA-A044-6B12E2CC1CCC"));
        // словарь с курсами валют в виде ['id валюты', 'курс'] для поиска курсов обмена
        var currencyRates = EntityManager<ExchangeCurrencyRate>.Instance
            .Find(string.Format("CurrencyEstimated = {0}", currencyEstimated.Id))
            .ToDictionary(rate => rate.CurrencyValuation.Id);
        // словарь с валютами в виде ['код', 'курс'] для поиска курсов обмена
        var currencies =
EntityManager<Currency>.Instance.FindByIdArray(currencyRates.Keys.ToArray());
        var ratesByCurrencyCode = currencies.ToLookup(c => c.CodeISOAlphabetic, c =>
currencyRates[c.Id]);

        foreach (var element in obj.ValuteCursOnDate)
        {
            // находим курс по коду валюты либо создаем новый
            var isoCode = element.VchCode;
            var rate = ratesByCurrencyCode[isoCode].FirstOrDefault()
                ?? CreateExchangeCurrencyRate(row, currencyEstimated,
                    currencies.FirstOrDefault(c => c.CodeISOAlphabetic == isoCode));

            rate.ValuationDate =  DateTime.ParseExact(obj.OnDate, "yyyyMMdd", null);
            rate.Nominal = Convert.ToInt64(element.Vnom);
            rate.ExchangeRate = Convert.ToDouble(element.Vcurs.Replace(".", ","));

            rate.Save();
        }
    }
    /// <summary>
    /// Создание новой записи справочника Курс валюты на основе полученных данных из сервиса ЦБ
РФ
    /// </summary>
    private ExchangeCurrencyRate CreateExchangeCurrencyRate(Currency currencyEstimated, Currency
currencyValuation)
    {
        var exchangeCurrencyRate = EntityManager<ExchangeCurrencyRate>.Create();

        exchangeCurrencyRate.CurrencyEstimated = currencyEstimated;
        exchangeCurrencyRate.CurrencyValuation = currencyValuation;

        return exchangeCurrencyRate;
    }

    [XmlRoot(ElementName = "ValuteData")]
    public class ValuteData
    {
        [XmlElement(ElementName = "ValuteCursOnDate")]
        public List<ValuteCursOnDate> ValuteCursOnDate
        {
            get;
            set;
        }
```

```
        [XmlAttribute(AttributeName = "OnDate")]
        public string OnDate
        {
            get;
            set;
        }

        [XmlAttribute(AttributeName = "xmlns")]
        public string Xmlns
        {
            get;
            set;
        }
    }

    [XmlRoot(ElementName = "ValuteCursOnDate")]
    public class ValuteCursOnDate
    {
        [XmlElement(ElementName = "Vname")]
        public string Vname
        {
            get;
            set;
        }

        [XmlElement(ElementName = "Vnom")]
        public string Vnom
        {
            get;
            set;
        }

        [XmlElement(ElementName = "Vcurs")]
        public string Vcurs
        {
            get;
            set;
        }

        [XmlElement(ElementName = "Vcode")]
        public string Vcode
        {
            get;
            set;
        }

        [XmlElement(ElementName = "VchCode")]
        public string VchCode
        {
            get;
            set;
        }
    }
}
```

Глава 8. Создание собственного лога

8.1. Использование логов ELMA в сценариях

Анализ происходящих в системе процессов является естественной потребностью системного администратора или разработчика. Функция логирования позволяет вести анализ этих процессов в системе, отслеживать процессы, в которых возникли проблемы, определять причины возникновения и устранять их. Иногда логирование используется как запись результата запросов или операций, которые во внешней системе являются платными. Например, это могут быть платные сервисы интеграции с **Контур.Фокус** или **DaData**.

Одним из самых простых и распространённых способов логирования является логирование в текстовый файл. При использовании данного способа отдельное событие представляет собой одну строку в текстовом файле. В системе ELMA по умолчанию уже настроено логирование, которое позволяет анализировать стандартные процессы системы. Подробнее о общесистемном логе можно прочитать в **Кратком руководстве администратора ELMA** (в главе 9.3).

Для работы со стандартным логированием необходимо в редакторе сценариев подключить следующее пространство имен:

```
using EleWise.ELMA.Logging;
```

Обратиться к стандартному логу и записать в него какую-либо информацию можно следующим образом:

```
Logger.Log.Error("Ошибка выполнения сценария");
```

Теперь каждый раз система, дойдя до этой строки кода, будет записывать текст, указанный в кавычках, в файл **error-log-<Текущая дата>**, расположенный в директории **<Папка с системой ELMA>\Web\logs\Error**. Текст ошибки может быть как статическим, так и динамическим (например, содержать имя объекта или полученное сообщение).

Система ELMA позволяет создавать собственные логи, которые хранятся в отдельной папке. Эта возможность позволит не только быстрее находить их, но и исключит факт перемешивания их с другими логами, существующими в системе. Как правило, собственные логи создаются для какого-то отдельного функционала системы. Например, если система ELMA интегрируется с двумя внешними системами, то для каждой внешней системы лучше создать собственный лог.

8.2. Реализация текстового лога

Для создания собственного лога необходимо в файл **log4net.config**, расположенный в директории **<Папка с системой ELMA>\Web\Config**, вставить следующий код:

```
<appender name="WebRequestsLogger" type="log4net.Appender.RollingFileAppender">
    <encoding value="utf-8" />
    <file value="logs/WebRequestsLogger/WebRequestsLogger-"/>
    <appendToFile value="true"/>
    <maxSizeRollBackups value="100"/>
    <maximumFileSize value="100Mb"/>
    <rollingStyle value="Composite"/>
    <staticLogFileName value="false"/>
    <datePattern value="yyyyMMdd"/>
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock"/>
    <filter type="log4net.Filter.LoggerMatchFilter">
      <loggerToMatch value="WebRequestsLogger" />
    </filter>
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level %logger{1} - %message%newline"/>
    </layout>
  </appender>
<logger name="WebRequestsLogger" additivity="false">
    <level value="DEBUG"/>
    <appender-ref ref="WebRequestsLogger" />
  </logger>
```

В первой строке кода XML-файла параметр **appender name** – это имя нового логгера, по которому в дальнейшем будет использоваться логгер. После сохранения XML-файла автоматически сформируется новая папка **<Папка с системой ELMA>\Web\logs\WebRequestsLogger**, в которую будут сохраняться все лог-файлы. Параметр **file value** указывает на путь, по которому будет создана папка для хранения лог-файлов нового логгера. Данный путь является типовым, и его не следует менять. Остальные параметры, описанные в XML-файле, являются тонкой настройкой, которая редко отличается от типовой.

Далее для использования нового лога в скриптах необходимо его вызвать. В случае, если необходимо использовать новый лог в одном месте (например, в бизнес-процессе), то достаточно загрузить его в статическую контекстную переменную и обращаться к ней в ходе бизнес-процесса.

```
// загрузка логгера WebRequestsLogger
private static ILogger _newLogger = Logger.GETLogger("WebRequestsLogger ");
```

В случае, если необходимо использовать новый лог в разных местах, то целесообразнее будет объявить класс и вынести его в модуль или глобальный модуль. Таким образом удастся избежать написания одного и того же кода вызова лога несколько раз. Для работы с логированием необходимо подключить пространство имен:

```
using EleWise.ELMA.Logging;
```

Текст сценария:

```
namespace EleWise.ELMA.WebRequestsLogging
{
    public class WebRequestsLogging
    {
        // загрузка логгера KFLogger
        private static ILogger _webRequestsLogger = Logger.GetLogger("WebRequestsLogger ");

        public static ILogger Log {
            get { return _webRequestsLogger; }
        }
    }
}
```

Теперь для использования собственного лога в сценарии необходимо подключить модуль, в котором описан лог, и указать пространства имен:

```
using EleWise.ELMA.Logging;
//пространство имен согласно наименованию namespace модуля в котором описан собственный лог
using Elewise.ELMA.WebRequestsLogging;
```

Обращение к собственному логу выглядит следующим образом:

```
WebRequestsLogging.Log.Error("Ошибка выполнения сценария");
```

Глава 9. Динамическая настройка активного подключения

9.1. Справочник с реквизитами подключения

Бывают такие случаи, когда интеграция с внешней системой требует настроек дополнительных параметров, которые время от времени могут меняться. Например, это может быть ключ, логин, пароль, адрес подключаемого сервиса, а также какие-либо другие параметры. Вероятность того, что потребуется изменять параметры интеграции с внешней системой, заставляет задуматься о простоте этих изменений. Если указывать данные параметры в исполняемом коде, то каждый раз при изменении необходимо будет исправлять или переписывать этот код, что повлечет за собой неудобство изменения параметров, а также увеличение трудозатрат. Оптимальным вариантом хранения значений параметров для интеграции является вынесение их в веб-приложение системы. В системе ELMA есть два способа для решения данной задачи.

К более простому способу относится хранение информации в справочнике. Такой способ удобнее использовать в случае, если изначально неизвестно, со сколькими системами будет выполняться интеграция. Таким образом все настройки для интеграций с различными внешними системами будут храниться в одном справочнике.

Рассмотрим пример интеграции с сервисом **Контур.Фокус**. **Контур.Фокус** – это сервис быстрой проверки контрагентов, который позволяет проверять контрагентов, уменьшать риски, анализировать конкурентную среду. Для подключения к данному сервису необходим ключ доступа, который можно получить у менеджера **Контур.Фокус**. В зависимости от приобретенного тарифа, ключ к **Контур.Фокус** может меняться, поэтому целесообразно вынести его хранение в веб-интерфейс системы ELMA.

Создадим справочник **Настройки интеграций** с именем класса и таблицей в базе данных **IntegrationSettings**, добавим в него часто встречающиеся при интеграции свойства (Рис. 69): **Наименование системы** (установим флажок **Является наименованием**), **URL**, **Логин**, **Пароль**, **Код**, **Подключение активно**.

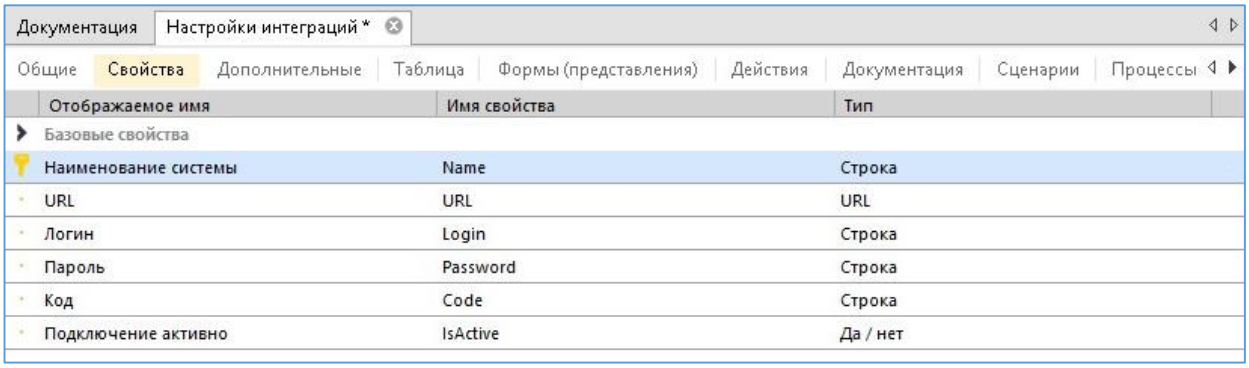


Рис. 69. Свойства справочника

Для свойств **Код** и **Подключение активно** при их настройке на вкладке **Настройки свойства** необходимо установить флажок **Участвует в поиске (фильтре)**.

После настройки справочника его необходимо опубликовать и перезапустить сервер. После перезапуска сервера в веб-интерфейсе в разделе **Справочники** отобразится новый справочник **Настройки интеграций**.

В справочнике необходимо создать (Рис. 70) новую запись, указав ее название, URL, ключ доступа в поле **Пароль**, задать код для поиска в сценарии, установить активность данной записи и нажать на кнопку **Сохранить**.

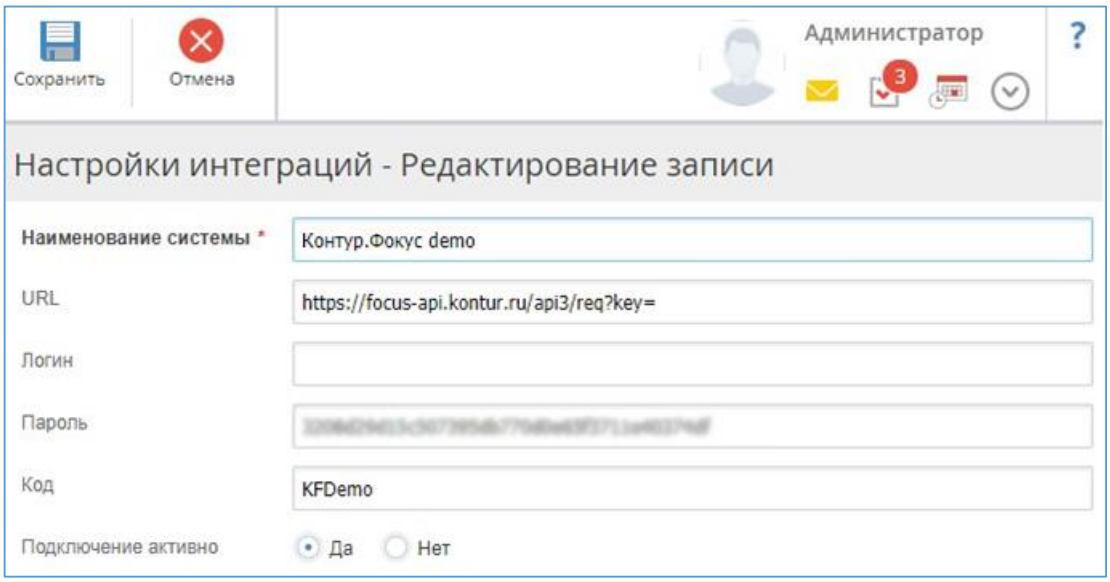


Рис. 70. Создание записи справочника «Настройки интеграций»

Теперь при подключении к сервису **Контур.Фокус** необходимо получать настройки из данного справочника. Получить нужные параметры справочника можно простым фильтром. Важным нюансом является то, что нам необходимо получить строго 1 запись по коду и обязательно с переключателем **IsActive**. В качестве

реквизитов подключения мы можем использовать только одну запись, поэтому нет необходимости загружать все. Значение **IsActive** переменной **Подключение активно** позволит быстро переключать используемые реквизиты (например, между тестовой и production-средой), а также полностью остановить интеграцию с определенной системой. Это может быть полезно, если необходимо на время остановить интеграцию: можно сбросить переключатели **IsActive** – в этом случае сценарии встанут на ошибку в очереди исполнения без необходимости прерывать процессы, а после активации переключателей их можно будет выполнить автоматически или вручную.

```
   // создание фильтра
    var filter = InterfaceActivator.Create<IntegrationSettingsFilter>();
    //условие для поиска по фильтру
    filter.Query = "Code = 'KFDemo' AND IsActive = true";
    //поиск записи в справочнике, удовлетворяющей условию фильтра
    var settings = EntityManager<IntegrationSettings>.Instance.Find(filter,
new FetchOptions(){MaxResults = 1}).FirstOrDefault();
```

Теперь из переменной **settings** можно получить все значения параметров, относящиеся к настройкам интеграции **Контур.Фокус**.

9.2. Осуществление настроек в веб-приложении в разделе Администрирование

Ко второму, более сложному способу хранения значений параметров для интеграции с внешней системой, относится добавление собственного раздела настроек в веб-приложении ELMA в разделе **Администрирование – Настройки системы**. Такой способ подходит в случае, если интеграция будет выполняться только с одной системой.

Рассмотрим пример, аналогичный вышеописанному с сервисом **Контур.Фокус** и стандартными параметрами для подключения: **URL**, **Пароль**, **Активность подключения**.

Для добавления простого блока настроек необходимо реализовать 3 класса, наследованных от **GlobalSettingsBase**, **GlobalSettingsModuleBase** и **GlobalSettingsModuleControllerBase**. Реализацию можно выполнять как в отдельном, так и глобальном модуле.

Реализуем первый класс, унаследованный от **GlobalSettingsBase –** базовый класс для глобальных настроек, хранит настройки в полях объекта**.** Для этого создадим отдельный класс и назовем его **KFSettingsBase**. В данном классе опишем поля объекта для хранения настроек. Для реализации класса необходимо подключить сборку **EleWise.ELMA.SDK.Web** и следующие пространства имен:

```
using EleWise.ELMA.Runtime.Settings;
using EleWise.ELMA.Model.Attributes;
using EleWise.ELMA;
```

Текст сценария:

```
public class KFSettingsBase : GlobalSettingsBase
{
    /// <summary>
    /// Инициализация параметров настройки, задаем начальные значения
    /// </summary>
    public KFSettingsBase()
    {
        AccessKey = "";
        URL = "";
        IsActive = false;
    }

    /// <summary>
    /// Параметры настроек для Контур.Фокус
    /// </summary>
    // Настройки для параметра Ключ доступа
    // Отображаемое имя параметра для ввода ключа
    [DisplayName(typeof(__Resources_KFSettingsBase), "AccessKey")]
    //Обязательность заполнения поля
    [Required(true)]
    // Атрибут для хранения значения параметра
    public string AccessKey { get; set; }
```

```
    // Настройки для параметра Адрес сервиса
    // Отображаемое имя параметра для адреса сервиса
    [DisplayName(typeof(__Resources_KFSettingsBase), "URL")]
    //Обязательность заполнения поля
    [Required(true)] //Обязательность заполнения поля
    // Атрибут для хранения значения параметра
    public string URL { get; set; }

    // Настройки для параметра Активность подключения
    [DisplayName(typeof(__Resources_KFSettingsBase), "IsActive")]
    // Атрибут для хранения значения параметра
    public bool IsActive { get; set; }

    /// <summary>
    /// Описание ресурсов
    /// </summary>
    internal class @__Resources_KFSettingsBase
    {
        // Значение отображаемого имени параметра Ввода ключа
        public static string AccessKey { get { return SR.T("Ключ доступа"); } }
        // Значение отображаемого имени параметра Адрес сервиса
        public static string URL { get { return SR.T("Адрес сервиса"); } }
        // Значение отображаемого имени параметра Активность подключения
        public static string IsActive { get { return SR.T("Активность подключения"); } }
    }
}
```

Для реализации второго класса **GlobalSettingsModuleBase<TSettings>**, который определяет название модуля, его раздел в системе, а также методы для загрузки/сохранения настроек, создадим новый класс **KFSettingsModuleBase**. Обозначим данный класс как компонент.

Для этого необходимо подключить пространство имен **using EleWise.ELMA.ComponentModel** и пометить класс атрибутом **[Component]**.

```
/// <summary>
/// Компонент, генерирующий новый раздел
/// </summary>
[Component]
public class KFSettingsModuleBase : GlobalSettingsModuleBase<KFSettingsBase>
{
    /// <summary>
    /// Генерация нового Guid для нового раздела(модуля) в системе
    /// </summary>
    public override Guid ModuleGuid
    {
        get { return new Guid("{658A7D31-873B-4aa1-B183-54EE55DE0EAD}"); }
    }

    /// <summary>
    /// Название нового раздела(модуля)
    /// </summary>
    public override string ModuleName
    {
        get { return SR.T("Настройки подключения Контур.Фокус"); }
    }
}
```

Для реализации третьего класса, который необходим для определения положения настроек модуля, а также для отрисовки форм просмотра/редактирования, создадим новый класс **KFSettingsModuleControllerBase**. Обозначим данный класс как компонент, для этого необходимо пометить его атрибутом **[Component]**.

Для работы с этим классом необходимо подключить пространство имен **EleWise.ELMA.Web.Mvc.Models.Settings**.

```
/// <summary>
/// Компонент для определения положения настроек модуля, а также для отрисовки форм
просмотра/редактирования
/// </summary>
[Component]
public class KFSettingsModuleControllerBase : GlobalSettingsModuleControllerBase<KFSettingsBase,
KFSettingsModuleBase>
{
    public KFSettingsModuleControllerBase(KFSettingsModuleBase module)
        : base(module)
    {
    }
}
```

После публикации глобального модуля потребуется перезапуск системы. В результате в веб-приложении ELMA в разделе **Администрирование – Настройки системы** отобразится новый раздел **Настройки подключения Контур.Фокус** с начальными параметрами подключения (Рис. 71).

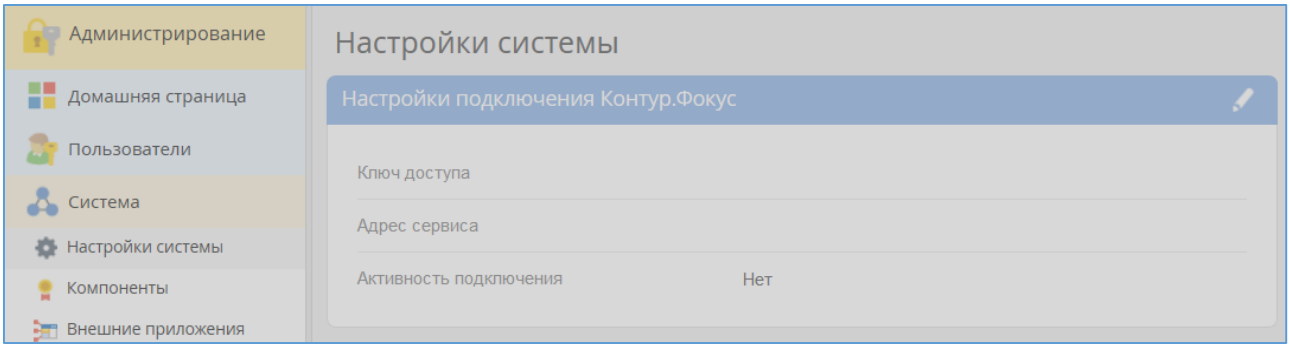


Рис. 71. Веб-приложение ELMA. Раздел «Администрирование – Настройки системы»

Для получения настроек для использования в сценарии необходимо подключить модуль для интеграции. Для этого необходимо на панели **Настройки** (Рис. 48) в контекстном меню блока **Модули** выбрать пункт **Добавить ссылку на сборку**, указать нужный модуль и подключить имя пространства (в нашем случае – **using KFSettings**).

Обращение к параметрам данного модуля будет осуществляться через **Locator**. Для работы с **Locator** необходимо подключить сборку имен **using EleWise.ELMA.Services**.

Для получения значения параметров необходимо использовать следующий код:

```
var KFSettings = Locator.GetServiceNotNull<KFSettingsModuleBase>().Settings;
//Пример получения значения ключа доступа
var key = KFSettings.AccessKey;
//Пример получения значения адреса сервиса
var url = KFSettings.URL;
```

9.3. Заголовки запросов – использование EndPointURL из настроек

В зависимости от того, с каким типом сервиса выполняется интеграция, будет реализован способ подключения к этому сервису. Рассмотрим два типа сервиса: **REST-сервис** и **WSDL-сервис**.

9.3.1. REST-сервис

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения. В общем случае REST является очень простым интерфейсом управления информацией без использования каких-либо дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором (таким, как URL). Каждый URL в свою очередь имеет строго заданный формат.

Для примера рассмотрим получение данных из сервиса **Контур.Фокус** по методу **Req** (получение реквизитов). Представьте ситуацию, что нам необходимо получить реквизиты контрагента в только в одном бизнес-процессе. ИНН и ОГРН контрагента вводятся вручную на форме пользовательской задачи в контекстные переменные **context.INN** и **context.OGRN**, адрес сервиса хранится в настройках системы в разделе **Администрирование**.

Адрес сервиса для получения данных по методу **Контур.Фокус** имеет как статическую часть, так и динамическую. Статическую часть запроса зададим в настройках (Рис. 72), а динамическую будем формировать при вызове метода.

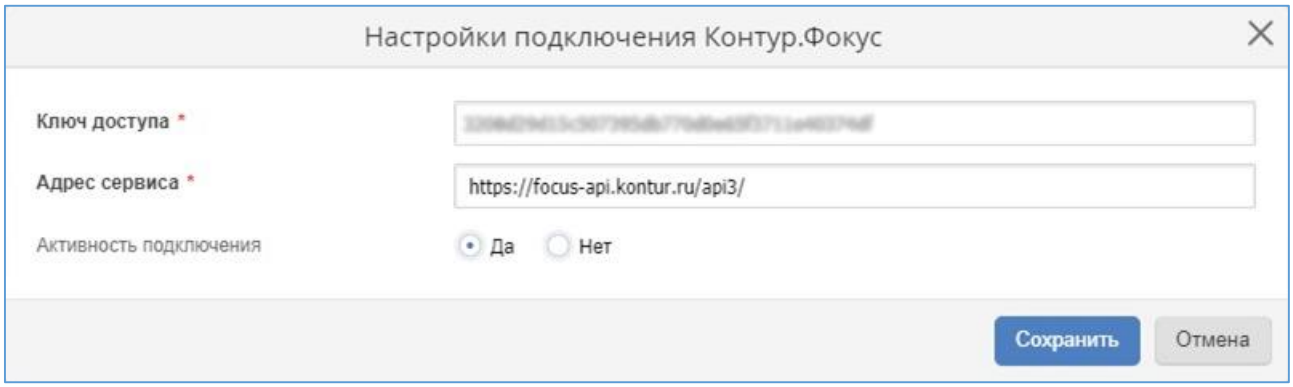


Рис. 72. Диалоговое окно «Настройки подключения Контур.Фокус»

Полный пример запроса можно найти в описании метода. В **Контур.Фокус** полный запрос имеет вид: **https://focus-api.kontur.ru/api3/req?key=Ключ доступа&ogrn=&inn=6663003127**,

где: **req** – название метода, по которому осуществляется запрос, **key**=<…> – ключ доступа, **ogrn**=**&inn**=6663003127 – параметр, по которому делаем запрос, ИНН или ОГРН юридического лица.

Реализуем подключение к сервису **Контур.Фокус** и вызовем метод **Req**. Для дальнейшей работы необходимо подключить пространства имен:

```
using EleWise.ELMA.Services;
using System.Net;
using System.IO;
```

Текст сценария:

```
/// <summary>
    /// GET-запрос к Контур-Фокус, Метод req
    /// </summary>
    /// <param name="method">Метод</param>
    /// <param name="parameters">Параметры запроса</param>
    /// <returns></returns>
    public static void Request(string method, Dictionary<string, string> parameters)
    {
        // Получение настроек
        var KFSettings = Locator.GetServiceNotNull<KFSettingsModuleBase>().Settings;

        //Получение значения адреса сервиса
        var url = KFSettings.URL;
        //Получение значения ключа доступа
        var key = KFSettings.AccessKey;
        //Укажем вызываемый метод вручную, тк мы заранее знаем какой метод нужен
        var methodKF = "req";
        //Формирование адреса для вызова метода
        var requestUrl = string.Format("{0}{1}?key={2}&{3}&{4}",
                              apiUrl,
                              method,
                              key,
                              context.OGRN,
                              context.INN);
        //Генерация запроса
        //
        var request = WebRequest.Create(requestUrl) as HttpWebRequest;
        request.Method = "GET";
        request.Timeout = 1000;

        //Получение ответа
        using (var response = (HttpWebResponse)request.GetResponse();)
        {
            using (var streamReader = new StreamReader(response.GetResponseStream(),
Encoding.UTF8))
            {
                // обработка полученных данных
            }
        }
    }
```

Теперь переменная **response** содержит в себе данные, полученные из **Контур.Фокус**.

Следующим шагом будет десериализация данных. **Контур.Фокус** возвращает ответ в формате JSON. Десериализация данных выполняется аналогично **Разделу**

7.2 данного краткого руководства. Для автоматического формирования класса десериализации объекта необходимо использовать сервисы, конвертирующие данные из JSON в C#.

9.3.2. WSDL-сервис

WSDL (Web Services Description Language) – язык описания веб-сервисов и доступа к ним, основанный на языке XML. Рассмотрим реализацию на ранее смоделированном бизнес-процессе актуализации курса валют. Вынесем ссылку для подключения к сервису в веб-интерфейс системы в раздел **Администрирование**. Подробное описание данных настроек приведено в предыдущем разделе. При этом необходимо подключить требуемую ссылку на сервис.

```
     /// <summary>
    /// Подключение к сервису ЦБ РФ
    /// </summary>
    /// <param name="context"></param>
    public void loggingOnToService(Context context)
    {
        //Получение URL из раздела Администрирование
        var url = Locator.GetServiceNotNull<CBRruSettings>().Settings.URL;
        //Создание конечной точки для подключения
        using (var client = new CBRru.DailyInfoSoapClient(new BasicHttpBinding(), new
EndpointAddress(url)))
        {
            var curSet = client.GetCursOnDate(DateTime.Now);
        }
    }
```

Подключение реализовано. Теперь переменная **curSet** содержит данные, полученные из сервиса ЦБ РФ.

9.4. Вызов веб-сервисов из системы ELMA в процессах

Реализовать интеграцию с внешним сервисом можно в бизнес-процессе, плагинах или модулях. Здесь важно понимать – в каком случае и где это удобнее реализовывать.

Реализация в бизнес-процессах удобна в том случае, когда интеграция будет использоваться только в рамках данного бизнес-процесса и в последующем она точно нигде не понадобится. К примерам таких бизнес-процессов можно отнести служебный бизнес-процесс актуализации курса валют. Данный бизнес-процесс не зависит ни от входящих данных, ни от надобности пользователю. Можно сказать, что он работает в фоновом режиме.

Что касается плагинов, то их удобнее использовать, когда применение интеграции предполагается в нескольких бизнес-процессах, а также, если для интеграции необходимы динамические входные данные. Интегрироваться с **Контур.Фокус**, например, удобнее через плагины. Один плагин равносилен одному методу.

Использование модулей при интеграции удобно в том случае, если интеграция будет использоваться неоднократно, и в модулях будет храниться подключение к сервису, сериализация/десериализация данных, объектные модели сериализации/десериализации данных. Также в модулях можно описать и собственные логгеры для интеграции с сервисом.

Получается, что суть модуля состоит в том, чтобы реализовать в себе всю логику работы с внешней системой целиком: получить данные от сервиса, привести их в нужный вид и вернуть пользователю готовый десериализованный ответ. Или же наоборот – получить данные из системы, подготовить ответ в нужном формате и отправить во внешний сервис (если говорить об отправке данных во внешний сервис, с которым предполагается интеграция).

9.5. Обработка ошибок в процессе

Во время интеграции с внешним сервисом могут возникать различные ошибки, обработку которых также необходимо предусмотреть. В системе есть два способа обработать ошибки: раздел **Очередь исполнения** и возможность обработки ошибки, заложенная в логике бизнес-процесса.

Рассмотрим первый вариант – раздел **Очередь исполнения** (Рис. 73).

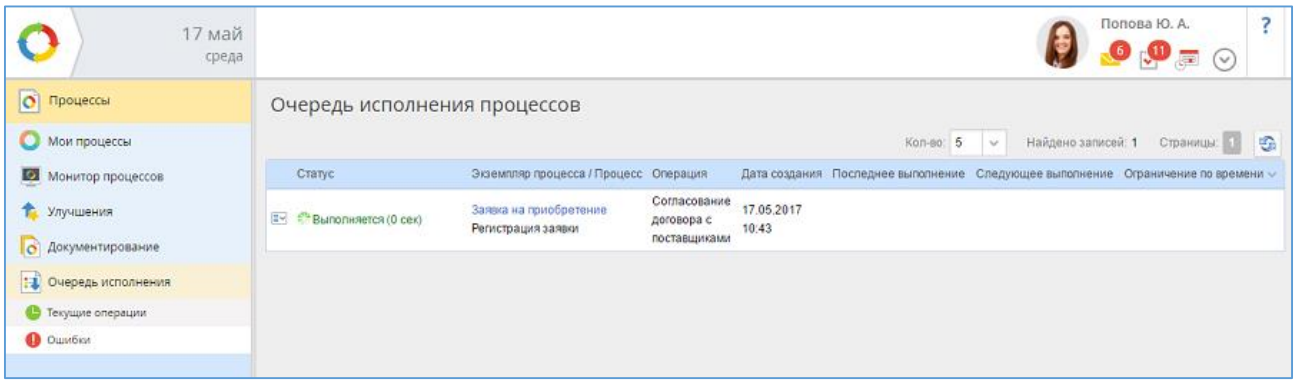


Рис. 73. Раздел «Очередь исполнения»

Данный раздел системы позволяет отслеживать процессы, которые в текущий момент времени обрабатываются в службе исполнения процессов, а также ошибки их исполнения. В случае, если при выполнении процесса происходит сбой при выполнении какой-либо операции, то информация об этом попадает в данный раздел с соответствующим сообщением об ошибке. Система осуществляет 9 попыток выполнения операции, после чего попытки выполнения операции прекращаются. На странице **Очередь исполнения процессов** сохраняется запись о неуспешном завершении выполнения и информация об ошибке. Более подробная информация приведена в справке по Платформе ELMA BPM.

Рассмотрим второй вариант – когда логика обработки ошибки заложена в бизнес-процессе. Возьмем ранее созданный бизнес-процесс «Актуализация курса валют» (Рис. 74). Добавим на карту бизнес-процесса пользовательскую задачу, на которую будет происходить эскалация в случае возникновения ошибки при актуализации курса валют. На переход из сценария необходимо установить эскалацию **Обработка ошибки**.

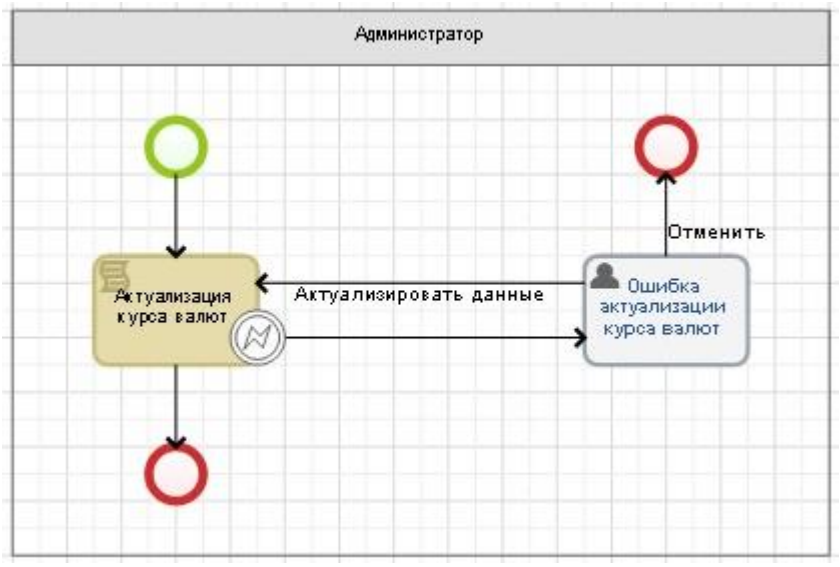


Рис. 74. Бизнес-процесс «Актуализация курса валют»

Запись ошибки будет вестись в контекстную переменную **Ошибка** (тип **Ошибка выполнения сценария**). Данную контекстную переменную необходимо разместить на форме пользовательской задачи **Ошибка актуализации курса валют**. В случае возникновения ошибки в сценарии переход по процессу будет осуществляться в данную задачу. Из задачи доступны два перехода: **Отменить** – завершить операцию актуализации курса валют и **Актуализировать данные –** повторно выполнить сценарий актуализации курса валют. Бизнес-процесс может быть завершен в двух случаях: если операция актуализации отменена пользователем и, если актуализация была выполнена успешно.

Глава 10. Организация интеграции при отсутствии веб-сервисов

10.1. Интеграция через файл

В системе ELMA есть несколько встроенных инструментов, которые реализуют данный тип интеграции:

1. Импорт контрагентов и возможностей из Excel шаблона
2. Генерация документа на основе шаблона
3. Экспорт записи в Excel

Сгенерировать документ с помощью шаблона можно как с помощью блока **Генерация документа** из раздела **Plug-ins** на карте бизнес-процесса, так и с помощью сценария на языке C#.

Практически любой список объектов системы, представленный в виде таблицы (процессы, документы, задачи, справочник и т.д.) может быть экспортирован в Excel файл с требуемыми полями с использованием необходимого фильтра данных объектов.

На Рис. 75 представлен пример такого экспорта записей справочника **Общий прайс**, а на Рис. 76 результат.

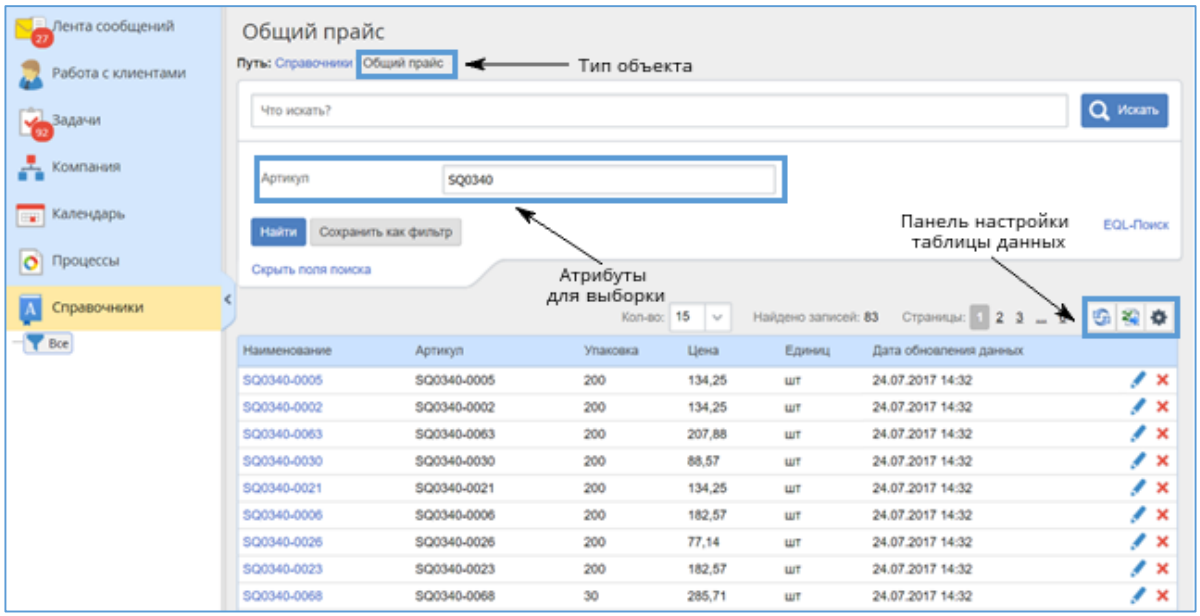


Рис. 75. Ручной экспорт записей справочника «Общий прайс»

	A	B	C	D	E	F	G
1	Наименование	Артикул	Упаковка	Цена	Единиц	Дата обновления данных	
2	SQ0340-0005	SQ0340-0005	200	134,25	шт	24.07.2017 14:32	
3	SQ0340-0002	SQ0340-0002	200	134,25	шт	24.07.2017 14:32	
4	SQ0340-0063	SQ0340-0063	200	207,88	шт	24.07.2017 14:32	
5	SQ0340-0030	SQ0340-0030	200	88,57	шт	24.07.2017 14:32	
6	SQ0340-0021	SQ0340-0021	200	134,25	шт	24.07.2017 14:32	
7	SQ0340-0006	SQ0340-0006	200	182,57	шт	24.07.2017 14:32	
8	SQ0340-0026	SQ0340-0026	200	77,14	шт	24.07.2017 14:32	
9	SQ0340-0023	SQ0340-0023	200	182,57	шт	24.07.2017 14:32	
10	SQ0340-0068	SQ0340-0068	30	285,71	шт	24.07.2017 14:32	
11	SQ0340-0028	SQ0340-0028	200	182,57	шт	24.07.2017 14:32	
12	SQ0340-0061	SQ0340-0061	200	207,88	шт	24.07.2017 14:32	
13	SQ0340-0069	SQ0340-0069	30	285,71	шт	24.07.2017 14:32	
14	SQ0340-0062	SQ0340-0062	200	207,88	шт	24.07.2017 14:32	

Рис. 76. Результат экспорта записей справочника «Общий прайс»

На практике часто требуются дополнительные возможности для интеграции с другими системами через файлы, а также обеспечение непрерывной синхронизации данных. В таких случаях потребуется создать служебный бизнес-процесс, который в заданные интервалы времени или по заданным событиям будет осуществлять репликацию данных по заранее определенной логике и содержанию. В таких бизнес-процессах часто прибегают к использованию в сценариях уже имеющиеся в системе библиотеки для работы с документами Microsoft Office: **Aspose.Words** и **Aspose.Cells**.

Интеграцию через файл можно разделить на два направления:

1. Создание/обновление данных в системе ELMA из файла. Примеры данной интеграции доступны в статьях Базы знаний ELMA: **Загрузка пользователей из файла формата Excel** и **Перенос данных из файла Excel в блок**, а также один из вариантов описан в **Разделе 10.1.1** данного документа.
2. Генерация файла с данными из ELMA. Примеры такой интеграции доступны в статьях Базы знаний ELMA: **Запись данных в Excel файл**, **Генерация файла по шаблону**.

10.1.1. Обновление справочника ELMA из Excel файла

Задача – обеспечить ведение справочника товаров, поставляемых различными поставщиками, с регулярным автоматическим обновлением позиций.

Для решения поставленной задачи был создан справочник **Общий прайс** (Рис. 77), который содержит каталог товаров (Рис. 78), укомплектованный товарами различных поставщиков.

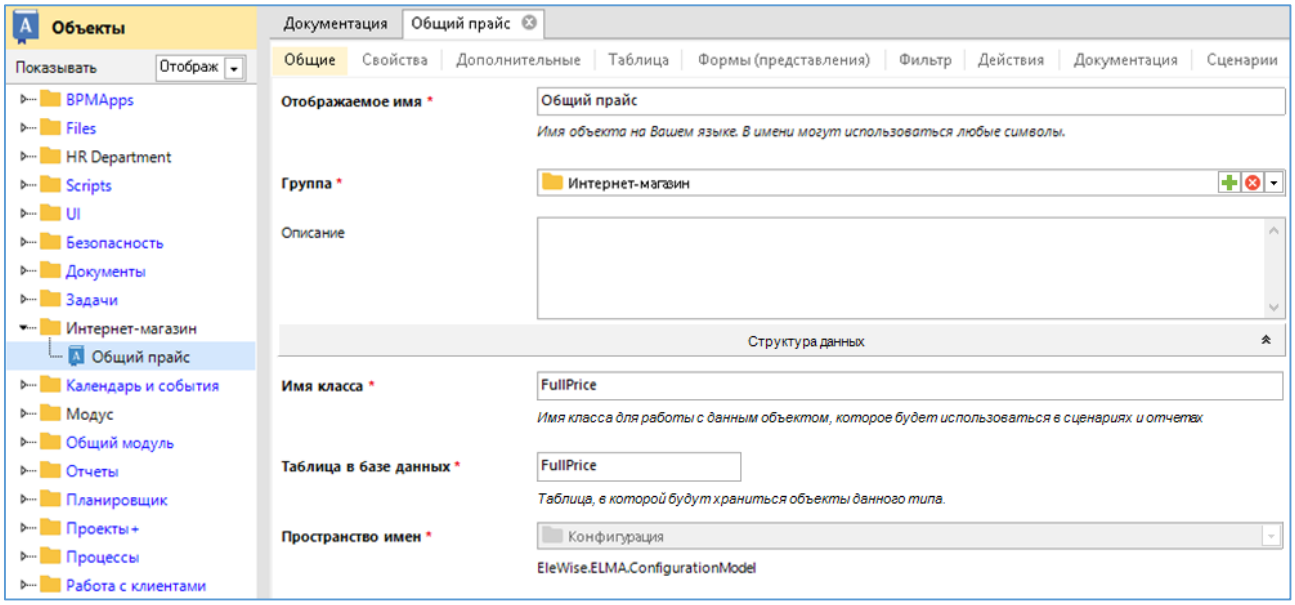


Рис. 77. Справочник «Общий прайс». Вкладка «Общие»

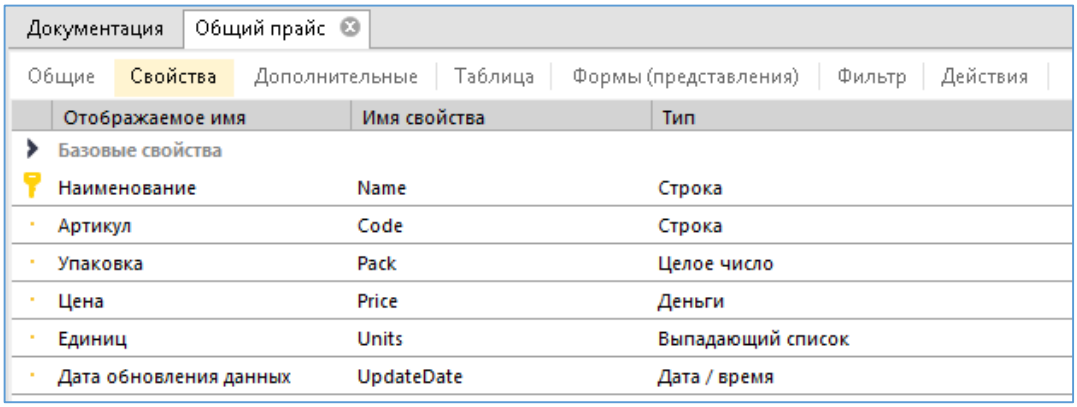


Рис. 78. Справочник «Общий прайс». Вкладка «Свойства»

Для обеспечения быстрого поиска по **Названию** и **Артикулу** товара средствами системы ELMA следует при настройке соответствующих свойств на вкладке **Дополнительно** установить флажок **Участвует в быстром поиске** (Рис. 79). Также для обеспечения возможности фильтрации в сценариях по полю **Артикул** следует на этой же вкладке для соответствующего свойства установить флажок **Участвует в поиске (фильтре)**.

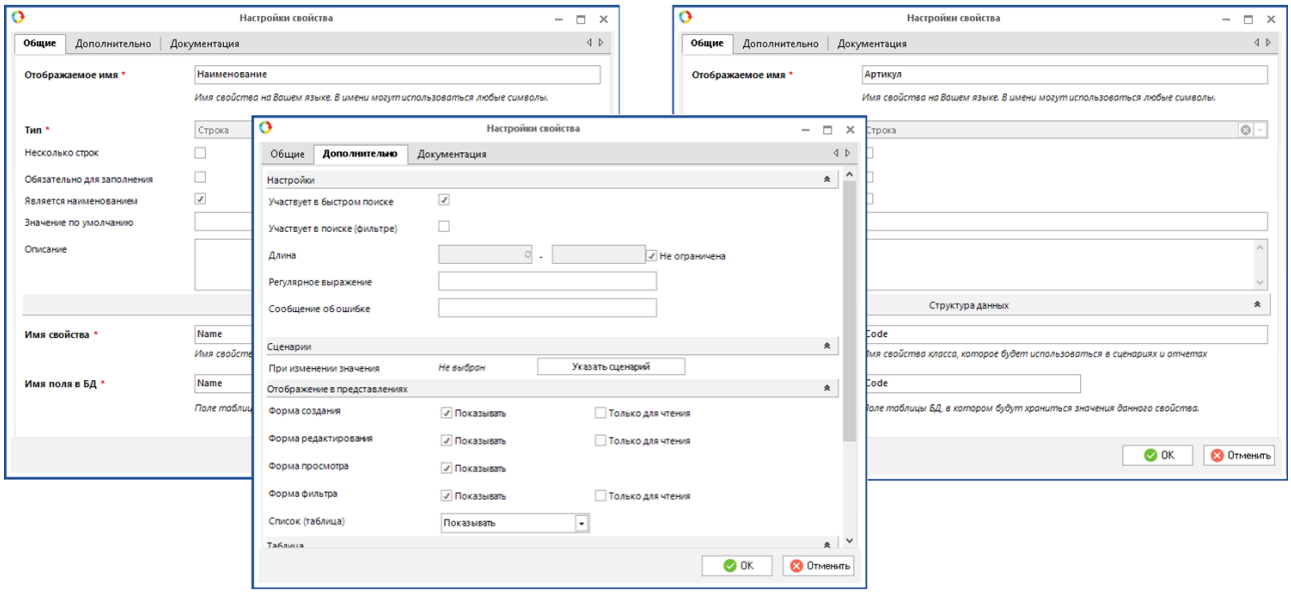


Рис. 79. Настройка свойств «Название» и «Артикул»

После создания справочника следует опубликовать его и перезапустить сервер ELMA.

Для хранения прайсов каждого поставщика будет использоваться документ типа **Файл**. После создания документа для его дальнейшего использования в сценариях на языке C# следует сохранить уникальный идентификатор документа. Id документа отображается в адресной строке веб-браузера при переходе в его карточку (Рис. 80).

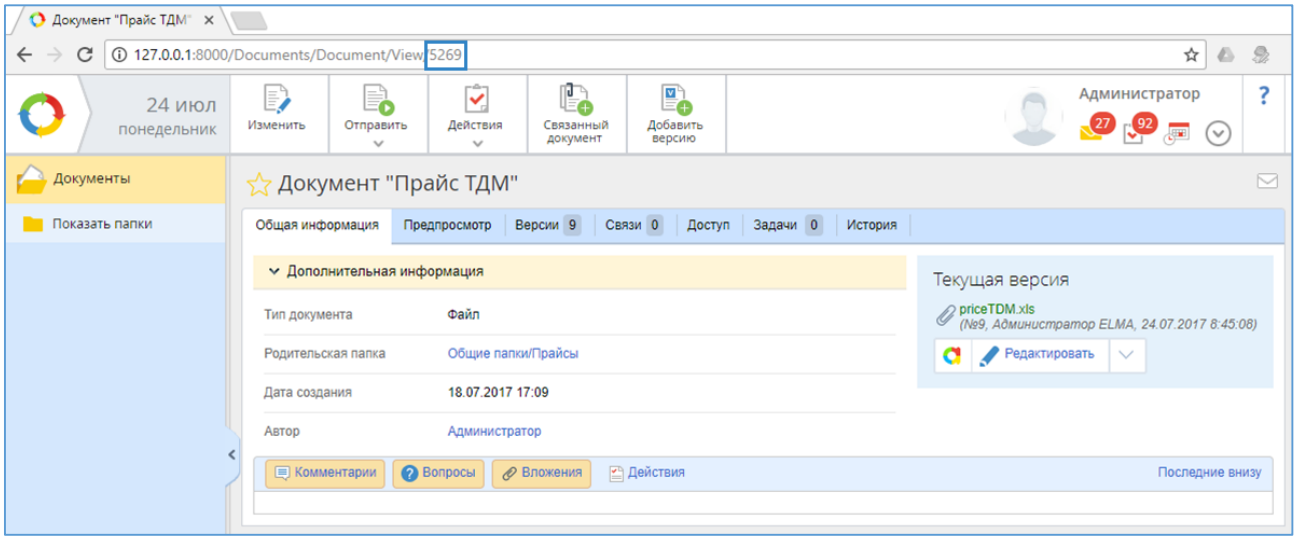


Рис. 80. Карточка документа с выделенным Id документа

Для регулярного обновления справочника товаров был создал служебный бизнес-процесс «Актуализация прайса» (Рис. 81, Рис. 82).

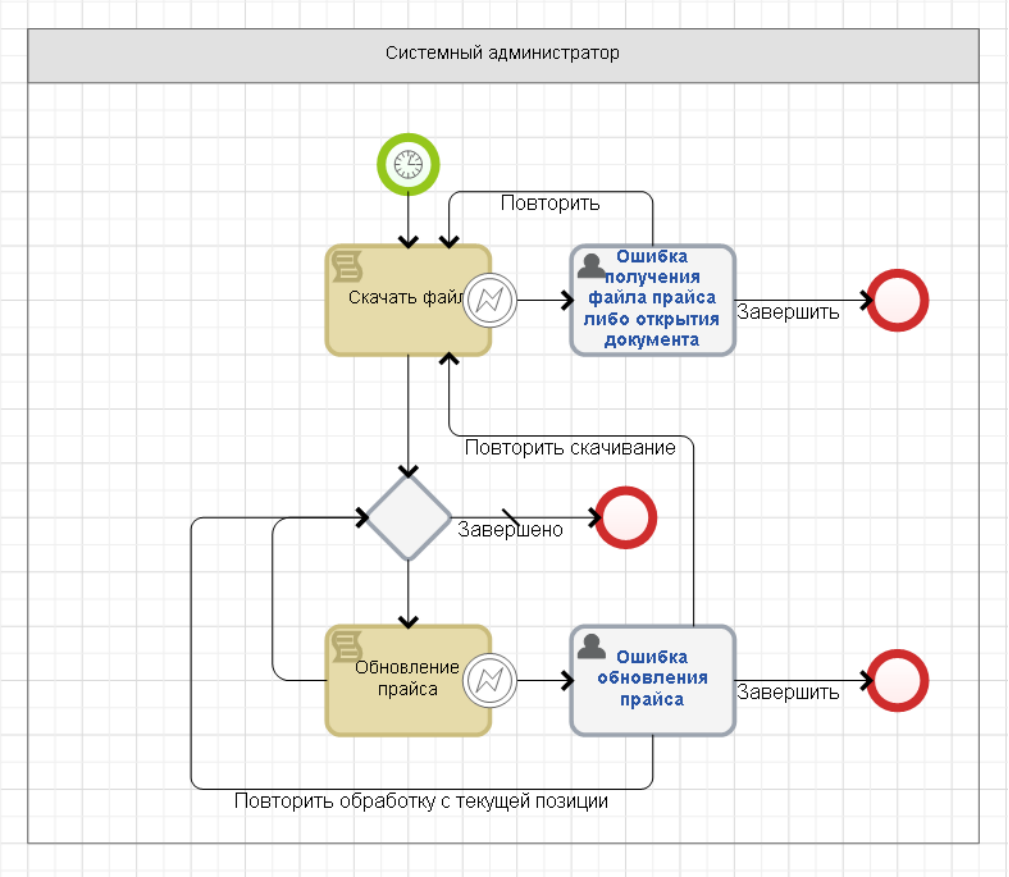


Рис. 81. Карта бизнес-процесса «Актуализация прайса»

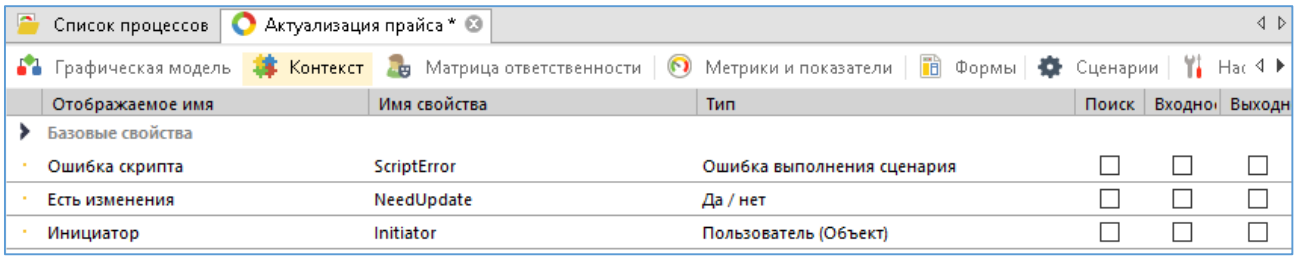


Рис. 82. Контекст бизнес-процесса «Актуализация прайса»

Установим на стартовом событии запуск по таймеру и настроим его на ежедневный ночной запуск в рабочие дни (Рис. 83).

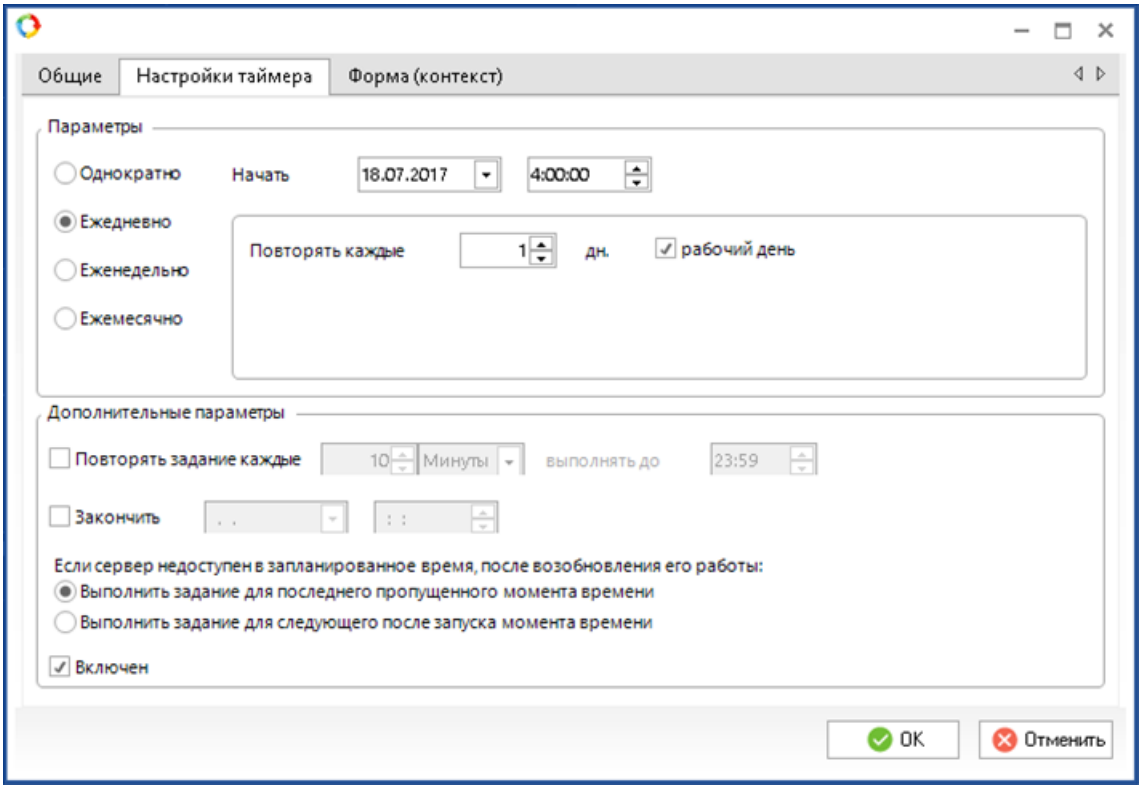


Рис. 83. Настройки стартового события

Процесс разделим на два этапа:

1. Получение нового файла и определение необходимости обновления справочника.
2. Обновление справочника.

Разработаем сценарий **DownloadFile** для первого пункта, в котором будет происходить скачивание файла, а затем его сравнение с предыдущим скачанным прайсом. Если была получена новая версия прайса, то добавляем версию к документу, делаем ее текущей, а затем запускаем процедуру обновления позиций справочника ELMA с помощью сценария **UpdatePrice**.

Ввиду того, что обычно прайсы содержат тысячи позиций, бизнес-процесс предусматривает обработку данных небольшими порциями, что уменьшает нагрузку на сервер и базу данных, а также исключает ошибку по таймауту при длительной работе сценария в блоке.

Алгоритм обновления позиции прайса:

1. Если справочник содержит запись с выбранным артикулом, то такая позиция обновляется параметрами из Excel файла.
2. Если артикула нет в справочнике, то создается новая запись по параметрам из Excel файла.

Стоит отметить, что данный сценарий разработан под строго определенный формат данных в Excel таблице. Для организации работы с несколькими поставщиками, которые имеют различные форматы прайсов, потребуется:

1. Добавить новый документ типа **Файл** для каждого поставщика.
2. Сделать копию данного процесса.
3. В процессе изменить идентификатор документа.
4. Доработать механизм поиска позиции начала данных для нового формата, а также сопоставления столбцов таблицы Excel и полей справочника.
5. Указать новое время запуска.

Ниже приведен листинг сценариев бизнес-процесса.

Пространства имен:

```
using System;
using System.IO;
using System.Net;
using System.Linq;
using EleWise.ELMA.API;
using EleWise.ELMA.Model.Common;
using EleWise.ELMA.Model.Managers;
using EleWise.ELMA.Files;
using EleWise.ELMA.Services;
using EleWise.ELMA.Runtime.Managers;
using EleWise.ELMA.Logging;
using EleWise.ELMA.ConfigurationModel;
using Aspose.Cells;

using Context = EleWise.ELMA.Model.Entities.ProcessContext.P_SyncPrice;
```

Текст сценария:

```
namespace EleWise.ELMA.Model.Scripts
{
    // Модуль сценариев процесса "Актуализация прайса"
    public partial class P_SyncPrice_Scripts :
EleWise.ELMA.Workflow.Scripts.ProcessScriptBase<Context>
    {
        const string url = "http://www.tdme.ru/download/";
        // наименование файла
        const string pricefile = "priceTDM.xls";

        // Скачать прайс и определить требуется ли обновление
        public virtual void DownloadFile(Context context)
        {
            // в данном примере обновляется версия для одного и того же документа с прайс-листом,
            // то есть его требуется задать за рамками процесса – в настройках.
            // Подробнее о создании настроек описано в Главе 9.2.
            var document =
Locator.GetService<PriceIntegrationSettingsModule>().Settings.PriceDocumentId;

            // В качестве простого примера можно подставить id конкретного документа,
            // например, PublicAPI.Docflow.Document.LoadOrNull (123L),
            // однако этого не рекомендуется делать в промышленных решениях

            var remoteFileHash = string.Empty;
            var localFileHash = string.Empty;
```

```
            MemoryStream downloadedFileStream;

            using (var cleint = new WebClient())
            {
                byte[] downloadedFile = webclnt.DownloadData(url + pricefile);
                downloadedFileStream = new MemoryStream(downloadedFile);
                remoteFileHash = CalculateHash(downloadedFileStream);
            }

            // текущая версия прайса
            using (var localFileStream =
File.OpenRead(document.CurrentVersion.File.ContentFilePath))
            {
                localFileHash = CalculateHash(localFileStream);
            }

            context.NeedUpdate = remoteFileHash != localFileHash;

            if (context.NeedUpdate)
            {
                Logger.Log.Error("Файлы имеют различия, инициируется сценарий обновления
прайса");

                context.NeedUpdate = true;
                context.CurrentRow = 0;

                using (downloadedFileStream)
                {
                    // добавление новой версии в документ
                    PublicAPI.Docflow.DocumentVersion.AddDocumentVersion(
                        document,
                        CreateBinaryFile(downloadedFileStream, pricefile),
                        Documents.Models.DocumentVersionStatus.Current
                    );
                }
            }
            else
            {
                Logger.Log.Error("Файлы идентичны, обновление прайса не требуется");
            }
        }

        static string CalculateHash(Stream stream)
        {
            using (var md5 = MD5.Create())
            {
                var hash = md5.ComputeHash(stream);
                return Encoding.Default.GetString(hash);
            }
        }

        // Преобразование потока в бинарный файл
        private BinaryFile CreateBinaryFile(Stream stream, string fileName)
        {
            var temp = BinaryFile.CreateContentFilePath(fileName);
            using (var fs = new FileStream(temp, FileMode.CreateNew, FileAccess.Write))
            {
                stream.Seek(0, SeekOrigin.Begin);
                stream.CopyTo(fs);
            }
            var mimeMappingService = Locator.GetServiceNotNull<IMimeMappingService>();
            var contractTemplate = new BinaryFile
            {
                ContentType = mimeMappingService.GetTypeByExtension(Path.GetExtension(fileName)),
```

```
                Name = Path.GetFileName(fileName),
                ContentFilePath = temp,
                CreateDate = DateTime.Now,
            };
            DataAccessManager.FileManager.SaveFile(contractTemplate);
            return contractTemplate;
        }

        // Обновление прайса
        public virtual void UpdatePrice(Context context)
        {
            // файл текущей версии прайса
            var doc = PublicAPI.Docflow.Document.LoadOrNull(priceid);
            if (doc == null)
            {
                return;
            }
            var newprice = doc.CurrentVersion.File;
            int row = (int) context.CurrentRow;
            // количество строк, обрабатываемых в рамках одного блока/транзакции
            int cnt = 50;

            var book = new Workbook(newprice.ContentFilePath);
            var sheet = book.Worksheets[0];

            Func<int, string> getCell = index => sheet.Cells[row, index];
            Func<int, string> getString = index => getCell(index).StringValue;
            Func<int, bool> isCellEmpty = index => string.IsNullOrWhiteSpace(getString(index));

            // Определить строку заголовка данных таблицы Excel
            if (row == 0)
            {
                while (sheet.Cells[row, 1].StringValue != "Номенклатура")
                {
                    row++;
                }
                row = row + 2;
            }

            var fullPriceManager = EntityManager<FullPrice>.Instance;

            var fullPriceCount = fullPriceManager.Count();
            var maxLoadingCount = 100;
            var fullPrices = fullPriceCount < maxLoadingCount
                ? fullPriceManager.FindAll().ToLookup(x => x.Code)
                : null;

            // Общий цикл по всей таблице Excel
            while (!isCellEmpty(1) && row < context.CurrentRow + cnt)
            {
                row++;
                var code = getString(2);
                if (!string.IsNullOrWhiteSpace(code))
                {
                    // Если данные были загружены заранее, то поиск будет вестись среди значений
`fullPrices`
                    var pos = fullPriceCount < maxLoadingCount
                        ? fullPriceManager.Find(filter, FetchOptions.First).FirstOrDefault()
                        : fullPrices[code].FirstOrDefault();

                    pos = pos ?? fullPriceManager.Create();

                    pos.Name = getString(1);
                    pos.Code = code;
```

```
                    if (!isCellEmpty(3))
                    {
                        pos.Pack = (long) getCell(3).IntValue;
                    }
                    if (!isCellEmpty(4))
                    {
                        pos.Price = getCell(4).FloatValue;
                    }
                    if (!isCellEmpty(5))
                    {
                        var value = getString(5);
                        if (pos.Units == null)
                        {
                            pos.Units = new DropDownItem(value);
                        }
                        else
                        {
                            pos.Units.Value = value;
                        }
                    }

                    pos.UpdateDate = DateTime.Now;
                    pos.Save();
                }
            }

             book.Dispose();

            if (row == context.CurrentRow + cnt)
            {
                Logger.Log.Error("Обработаны позиции с " + context.CurrentRow + " по " + row);
            }
            else
            {
                Logger.Log.Error("Процесс завершен. Всего обработано " + row + " строк");
                context.NeedUpdate = false;
            }

            context.CurrentRow = row;
        }
    }
}
```

10.1.2. Генерация Excel файла из справочника ELMA

Для ручного формирования прайса может быть использован встроенный механизм экспорта данных из справочников. Права на экспорт элементов каждого конкретного справочника назначается администратором системы в разделе ***Администрирование – Пользователи – Настройки доступа – Справочники***.

При наличии прав доступа на странице соответствующего справочника на панели настроек таблицы списка элементов будет отображена иконка  . При нажатии на данную иконку осуществляется запуск процедуры экспорта записей справочника в Excel файл.

Однако для автоматической генерации, а также для реализации дополнительной логики при формировании Excel файла потребуется создать служебный бизнес-процесс, который в заданное время будет формировать Excel файл из позиций справочника ELMA. Данный механизм подробно описан в соответствующей статье Базы знаний ELMA.

10.2. Интеграция через базу данных

В ходе процесса обработки заказа происходит неоднократная смена статуса заказа. Для того чтобы пользователь мог видеть актуальный статус своих заказов на портале, реализуем ее актуализацию.

В рассматриваемом решении используется интеграция с CMS **OpenCart** и базой данных **MySQL**. Для удобства применения выполним решение в виде пользовательского расширения. Создадим расширение **Обновление статуса на портале** (Рис. 84), в котором входными параметрами будут **Заказ (Order)** и **Новый статус (Status)**, а выходным параметром будет **Результат (Result)** (Рис. 85), принимающий значения **True/False** (**False** будет означать неудачное выполнение расширения).

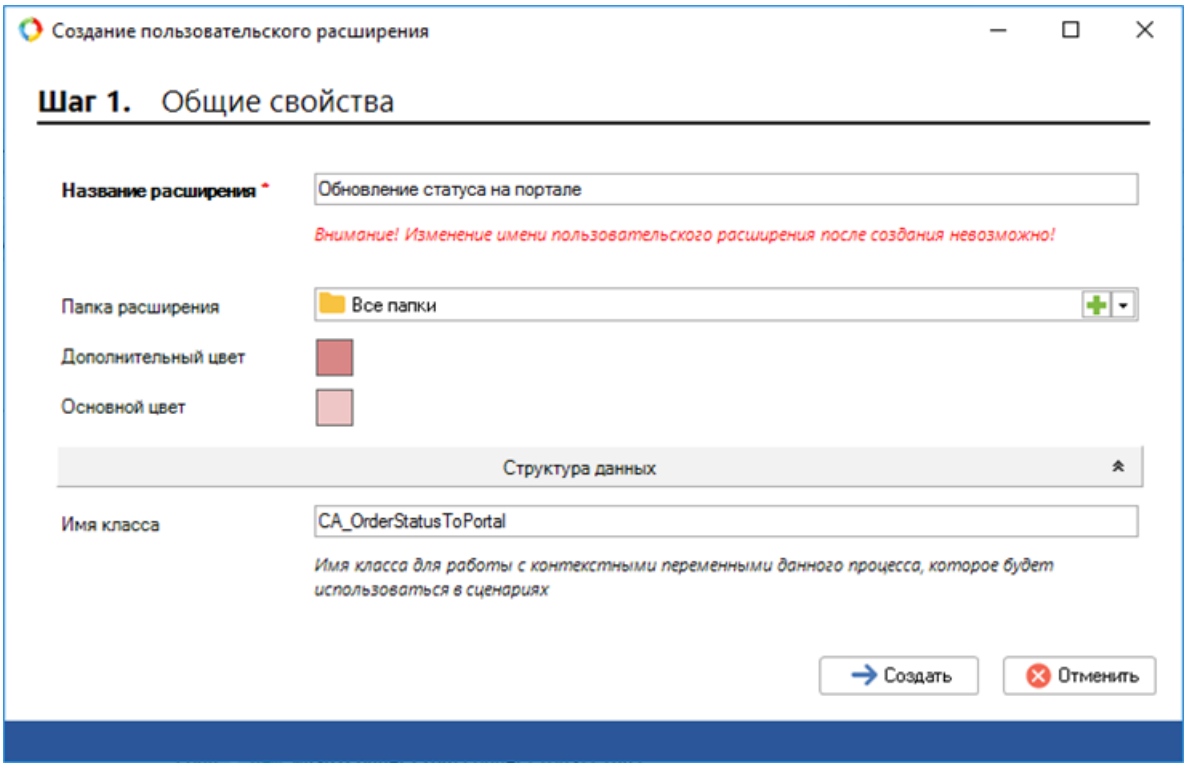


Рис. 84. Создание пользовательского расширения «Обновление статуса на портале»

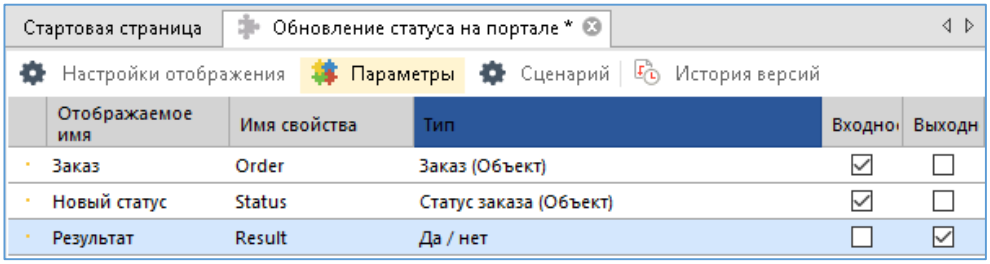


Рис. 85. Параметры пользовательского расширения «Обновление статуса на портале»

В системе ELMA существует возможность подключения внешних библиотек C# и их использования при написании сценариев. Для этого следует скопировать dll с библиотекой в папку **<Папка с системой ELMA>\Web\bin** (потребуется перезапуск сервера), а также в папку **<Папка с системой ELMA>\Designer** (потребуется перезапуск Дизайнера ELMA).

Далее необходимо на вкладке **Сценарий** (Рис. 86) пользовательского расширения на панели **Настройки** (Рис. 48) нажать на кнопку **Добавить – Добавить ссылку на сборку**, перейти на вкладку **Обзор** и выбрать требуемую dll.

После этого можно выполнить **using** и воспользоваться библиотекой. Для решения задачи воспользуемся библиотекой **MySql.Data**, которая позволяет выполнять различные действия с СУБД MySQL в сценариях C#. Текущая стабильная версия библиотеки 6.9.9.

После скачивания пакета и размещения файла **MySql.Data.dll** в соответствующей папке перейдем на вкладку **Сценарий** пользовательского расширения **Обновление статуса на портале** и подключим библиотеку (Рис. 86).

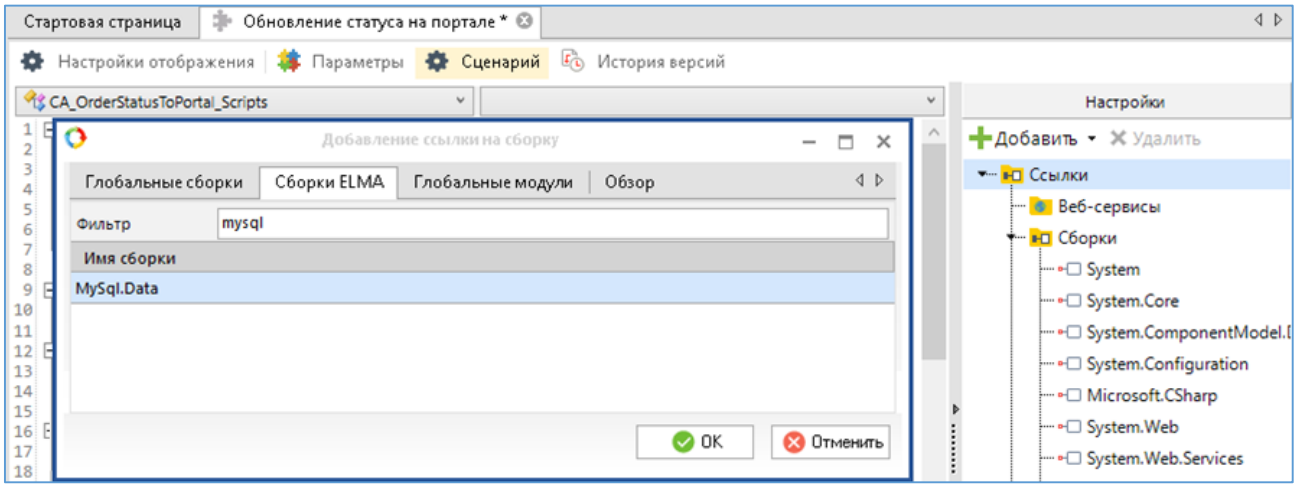


Рис. 86. Обновление статуса на портале. Добавление ссылки на сборку MySQL.Data

После добавления ссылки на сборку можно подключить пространство имен **MySql.Data.MySqlClient**.

```
using System;
using MySql.Data.MySqlClient;
using Parameters = EleWise.ELMA.Workflow.Models.Parameters.CA_OrderStatusToPortal;
```

Для обеспечения синхронизации справочника **Статус заказа**, а также для корректного изменения статуса на портале, на вкладке **Сценарии** данного пользовательского расширения необходимо добавить следующий метод:

```
public static long GetStatusID(MySqlConnection conn, MySqlTransaction tr, string statusname)
{
    MySqlCommand cmd = new MySqlCommand();
    MySqlDataReader reader;
    long StatusID = 0;
    cmd.Connection = conn;
    cmd.Transaction = tr;
    cmd.CommandText = String.Format("SELECT order_status_id " +
        "FROM opencart.oc_order_status " +
        "WHERE name like '{0}'", statusname);

    try
    {
        reader = cmd.ExecuteReader();
        if (reader.HasRows)
        {
            while (reader.Read())
            {
                Int64.TryParse(reader["order_status_id"].ToString(), out StatusID);
            }
        }
    }
    finally
    {
        reader.Close();
    }

    if (StatusID > 0)
    {
        return StatusID;
    }
    else
    {
        cmd.CommandText = "INSERT INTO opencart.oc_order_status(name, language_id) " +
            "VALUES(@name, @language_id)";
        cmd.Prepare();
        cmd.Parameters.AddWithValue("@name", statusname);
        cmd.Parameters.AddWithValue("@language_id", 1);
        cmd.ExecuteNonQuery();
        return GetStatusID(conn, tr, statusname);
    }
}
```

Метод в качестве входных параметров получает текущее соединение **MySQL**, текущую транзакцию и строковое наименование статуса. Производится **SELECT** запрос в базу данных. Если идентификатор найден, то он передается на выход метода. Если идентификатор не был найден, значит портал не содержит данного статуса, и он создается командой **INSERT**, а затем извлекается идентификатор и подается на выход метода.

Для установки соединения с MySQL сервером потребуется указать строку подключения к базе данных:

```
const string connectionString = "server=localhost;" +
                                "user=root;" +
                                "database=opencart;" +
                                "SslMode=none;" +
                                "charset=utf8";
```

Данные реквизиты также можно хранить одним из описанных в **Глава 9** способов.

Основной метод имеет следующий сценарий:

```
public void Execute(Parameters parameters)
{
    MySqlConnection conn = null;
    string StatusName = parameters.Status.Name;
    long orderid = parameters.Order.OpenCartOrder_id;
    try
    {
        conn = new MySqlConnection(connectionString);
        // Создаем соединение по строке подключения
        conn.Open();
        // Начало транзакции
        using (var tr = conn.BeginTransaction())
        {
            // Инициализация запроса
            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = conn;
            cmd.Transaction = tr;
            // Получение идентификатора
            long StatusId = GetStatusID(conn, tr, StatusName);
            cmd.CommandText = string.Format("UPDATE opencart.oc_order " +
                                            "SET order_status_id = {0} " +
                                            "WHERE order_id={1} ", StatusId, orderid);
            // Выполнение запроса
            cmd.ExecuteNonQuery();
            // Подтверждение транзакции
            tr.Commit();
        }
    }
    catch (MySqlException ex)
    {
        // Отказ транзакции
        tr.Rollback();
        parameters.Result = false;

    }
    finally
    {
        if (conn != null)
        {
            // Закрыть соединение
            conn.Close();
        }
    }
}
```

В основном методе устанавливается соединение с СУБД MySQL по заданной строке подключения, затем открывается новая транзакция, выполняется метод получения идентификатора статуса, после чего обновляется **Статус заказа** и транзакция завершается. Если в процессе соединения с базой данных либо при выполнении запросов произошла ошибка, то транзакция будет отменена, а в выходной параметр **Результат** запишется значение **False**, которое может быть в дальнейшем обработано.

После публикации данного пользовательского расширения его можно разместить на карте бизнес-процесса (Рис. 87)

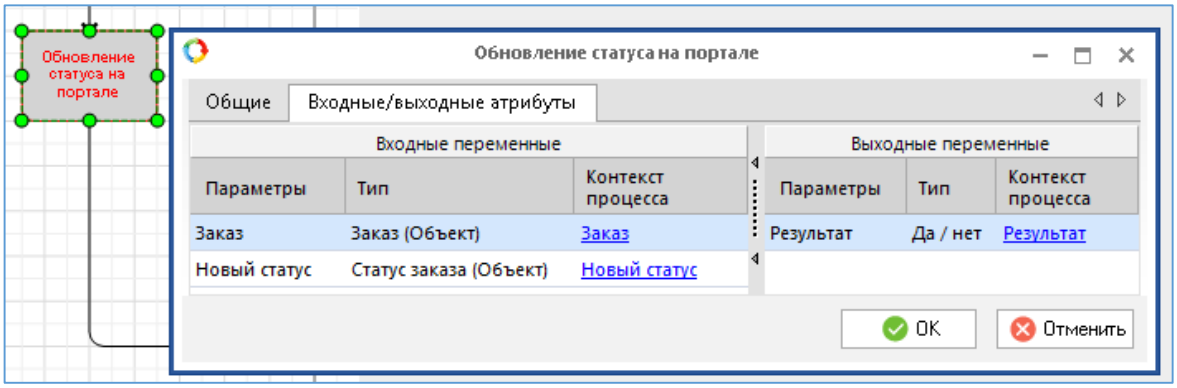


Рис. 87. Размещение блока «Обновление статуса заказа» на карте бизнес-процесса

Глава 11. Полезные ресурсы

Помимо текущего руководства, посвященного основным возможностям, которые предоставляет **Платформа ELMA BPM** для интеграции с другими внешними системами, доступно отдельное Краткое руководство по интеграции системы ELMA с системой «1С:Предприятие».

Также существуют аналогичные издания, в которых описываются основные возможности приложений системы ELMA:

- Краткое руководство по Платформе ELMA BPM
- Краткое руководство по внутреннему порталу ELMA
- Краткое руководство по приложению ELMA ECM+
- Краткое руководство по приложению ELMA CRM+
- Краткое руководство по приложению ELMA Проекты+
- Краткое руководство по приложению ELMA KPI
- Краткое руководство администратора ELMA
- Краткое руководство по настройке работы системы ELMA на высоких нагрузках

Данные руководства знакомят читателя с ключевыми особенностями системы, подробное и исчерпывающее описание функционала системы ELMA содержится в справке, которая входит в поставку системы, а также всегда доступна в сети Интернет: http://www.elma-bpm.ru/kb/help.

Справочные материалы по каждому приложению разбиты на три категории: для пользователя, для внедрения и для администратора, что позволяет быстро найти нужную информацию.

Общее описание приложений и условия их приобретения доступны на **сайте ELMA**: http://www.elma-bpm.ru. Также на данном сайте всегда можно обратиться в компанию ELMA с помощью кнопки **Задать вопрос**, расположенной на главной странице сайта.

Ключевые возможности приложений и основные способы их использования продемонстрированы в **on-line демоверсии** http://www.elma-bpm.ru/download. Если же вы хотите подробнее изучить какое-либо из приложений, по этой же ссылке доступно скачивание демоверсии с такими же настройками, как и в on-line версии.

Система ELMA постоянно развивается, и на базе Платформы и приложений разрабатываются компоненты, предназначенные для решения различных более узких и конкретных задач. Со списком и условиями приобретения таких готовых решений Вы можете ознакомиться в **ELMA Store**: https://store.elma-bpm.ru.

При разработке собственных решений полезными окажутся материалы **Базы знаний ELMA**: http://www.elma-bpm.ru/kb.

Если же при работе в системе возникли вопросы технического характера, можно обратиться на сайт технической поддержки ELMA: http://support.elma-bpm.ru.

Для получений консультаций по системе ELMA или по сотрудничеству с компанией ELMA позвоните нам:

- Ижевск: +7 (3412) 93-66-93
- Москва: +7 (499) 921-02-87
- Казань: +7 (843) 567-17-69
- Киев: +7 (909) 050-10-06
- Алматы: +7 (727) 313-15-04