Введение

Данный документ содержит описание процесса разработки прикладных приложений на основе системы «ИнМета».

Основы создания приложений «ИнМета»

Предварительные требования

Для создания обучающего приложения «ИнМета» нам понадобятся:

- Операционная система «Windows 7 Ultimate» или одна из серверных редакций Windows, перечисленных в руководстве системного администратора «ИнМета».
- Платформа «Microsoft .Net Framework» версии 2.0 с установленными пакетами обновлений.
- Веб-сервер «Internet Information Services» с включенными компонентами «ASP» и «ASP.NET».
- Сервер баз данных «Microsoft SQL Server», возможно в редакции «Express».
- Инструментальная система «ИнМета» версии 1.9.106 и выше.

И несколько слов о том, какие знания нам потребуются для прохождения обучения:

- «Microsoft SQL Server» – достаточно базовых навыков, позволяющих установить СУБД на компьютер, и создать пустую базу данных.
- «SQL» – потребуется хорошее знание языка запросов «SELECT», включая понимание того, что такое «JOIN».
- «XML» – этот язык разметки необходимо знать досконально, так как на нем основан язык описания метаданных ИнМеты, а так же различные конфигурационные файлы.
- «HTML» – потребуется уровень выше среднего, так весь интерфейс конечного пользователя ИнМеты построен на принципах Web/Intranet.
- «VBScript» – потребуется уверенное владение этим языком программирования, так как именно он является основным языком программирования прикладной логики приложений ИнМета.

Подготовка приложения

Далее в документе будем считать, что система «ИнМета» установлена в папку «C:\Program Files\Integro\InMeta». Ничего страшного, если вы установили систему в другую папку, просто учитывайте это при размещении файлов.

В нашем примере мы назовем наше приложение «Training».

Создание каталогов приложения

Для нового приложения необходимо создать *базовую папку приложения*, которая размещается в папке установки «ИнМета» и называется также как и приложение: «C:\Program Files\Integro\InMeta\Training».

В базовой папке приложения необходимо создать следующие вложенные папки:

- «Meta» - для размещения *метаданных* приложения.
- «Web» - для размещения веб-содержимого нашего приложения.
- «Web\gen_docs» - для размещения шаблонов выходных документов.

- «Db» - для размещения файлов данных (SQL Server). Можно и в другом месте, это не критично.
- «Attachments» - для размещения прикрепленных файлов.

Папки «Meta», «Web» и «gen_docs» должны быть размещены именно там, где указано. Это важно.

Прочие папки можно разместить в любом удобном месте, но в нашем примере мы разместим их именно так, как описывалось выше.

Создание скрипта перезапуска приложения

При внесении некоторых изменений в метаданные приложения необходимо перезагружать некоторые подсистемы, работающие на основе метаданных.

Для автоматизации этой процедуры в базовой папке приложения создадим файл «restart.cmd»:

```
net stop w3svc /y
net stop inmetaserver
net start inmetaserver
net start w3svc
```

Создание базового набора метаданных

Метаданные приложения содержат полное описание прикладной логики и различные административные настройки. Метаданные представлены в виде набора файлов в формате «XML». В базовый набор метаданных входят следующие обязательные файлы:

- «_db_info.xml» – ключевые параметры приложения.
- «_ui.xml» – перечень рабочих мест пользователей.
- «_users.xml» – список пользователей, авторизованных для работы с приложением.
- «_config.xml» – рабочие настройки среды выполнения.
- «_roles.xml» – список пополняемых пользовательских функций подсистемы прав доступа.
- «_policies.xml» – список предопределенных пользовательских функций подсистемы прав доступа.
- «_denies.xml» – перечень настраиваемых запретов подсистемы прав доступа.
- «_developer.xml» – метаданные приложения, автоматически формируемые различными средствами разработки.
- «_gen_docs.xml» - метаданные приложения, автоматически формируемые подсистемой проектирования выходных документов.
- «meta-app.xml» - точка входа метаданных. Содержит полный перечень и последовательность загрузки файлов, содержащих метаданные приложения.

В нашем примере создадим перечисленные файлы со следующим содержимым:

_db_info.xml

Определим идентификатор приложения и параметры подключения к СУБД (база данных будет создана в последующих разделах):

```
<?xml version="1.0" encoding="windows-1251"?>
<dbinfo>
    <id>Training</id>
    <caption>Система "Обучение"</caption>
    <driver name="ADO.MSSQL">
        <param name="server">.</param>
        <param name="database">Training</param>
        <param name="login-name">sa</param>
        <param name="login-password">1</param>
        <param name="command-timeout">600</param>
        <param name="connection-timeout">600</param>
    </driver>
</dbinfo>
```

_policies.xml

Зарегистрируем встроенные функции системы «ИнМета»:

```
<?xml version="1.0" encoding="windows-1251"?>
<meta xml-builder-skin="inmeta">
    <policy name="SecurityAccess" caption="Настройка прав доступа"/>
    <policy name="ViewUpdateLog" caption="Просмотр журнала изменений"/>
    <policy name="ExportData" caption="Экспорт данных"/>
</meta>
```

_roles.xml

Создадим пустой список пользовательских функций:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
</inmeta>
```

_ui.xml

Создадим одно рабочее место с названием «Adm», заголовком «Администратор» и стартовой страницей «Default.asp». Разрешим пользователям, относящихся к этому рабочему месту, использование встроенных функций:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
    <context name="Adm" entry-point="Default.asp" caption="Администратор">
        <SecurityAccess type="policy"/>
        <ViewUpdateLog type="policy"/>
        <ExportData type="policy"/>
    </context>
</inmeta>
```

_users.xml

Создадим одного пользователя и свяжем его с рабочим местом администратора:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
    <user account="учетная запись" caption="имя пользователя" context="Adm" />
</inmeta>
```

_denies.xml

Создадим пустой список запретов подсистемы прав доступа:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
</inmeta>
```

_developers.xml

Подготовим пустой файл для подсистем разработки:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
</inmeta>
```

_gen_docs.xml

Подготовим пустой файл для подсистемы построения выходных документов:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
</inmeta>
```

_config.xml

Заполним параметры среды выполнения приложения:

```
<?xml version="1.0" encoding="windows-1251" ?>
<meta xml-builder-skin="inmeta">
    <config development-mode="true">
        <app-path>training</app-path>
        <gen-docs developer-package="_gen_docs.xml"/>
    </config>
</meta>
```

meta-app.xml

Перечислим созданные выше файлы с метаданными:

```
<?xml version="1.0" encoding="windows-1251"?>
<meta>
  <meta-data src="_db_info.xml" />
  <meta-data src="_policies.xml" />
  <meta-data src="_roles.xml" />
  <meta-data src="_ui.xml" />
  <meta-data src="_users.xml" />
  <meta-data src="_denies.xml" />
  <meta-data src="_config.xml" />
  <meta-data src="_developer.xml" />
  <meta-data src="_gen_docs.xml" />
</meta>
```

Создание базового веб-содержимого

В папке «Web» создадим файл «Default.asp» следующего содержания:

```
<html>

<head>
    <title>Система "Обучение"</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
    <meta http-equiv="Content-Language" content="ru">
</head>

<body>
    <h1>Пример приложения "Обучение"</h1>
</body>

</html>
```

Создание базы данных

База данных создается при помощи утилиты «SQL Management Studio» из комплекта поставки СУБД «Microsoft SQL Server».

Разместим файлы базы данных в папке приложения «Db» и дадим ей имя, совпадающее с идентификатором приложения, «Training».

Теперь необходимо выполнить первоначальную подготовку базы данных:

- Перезагрузим приложение при помощи файла «restart.cml».
- Запустим утилиту «DBManager.exe».
- В главном окне утилиты выберем наше приложение «Training» и нажмем «Далее».
- Появится сообщение «База данных не изменилась». Нажмем «ОК».
- Поставим галочку «Обрабатывать системные таблицы» и нажмем кнопку «Обновить». Утилита создаст несколько служебных таблиц, необходимых для работы нашего приложения.
- Можно закрыть утилиту обновления БД. База данных подготовлена для дальнейшей работы.

Автоконфигурирование приложения

В базовый комплект «ИнМета» включена утилита «InMeta.ServerConfigurator», предназначенная для автоматической настройки различных параметров сервера, на котором функционируют приложения «ИнМета».

Данная утилита выполняет как общую настройку сервера для всех приложений (регистрация ASP.NET, общая настройка IIS и пр.), так и специфические настройки для каждого приложения (создание веб-приложения в IIS, настройка прав доступа на папки приложения и пр.).

Просто запускаем утилиту «InMeta.ServerConfigurator.exe», размещенную в папке «ServerRuntime\InMeta.net», ждем завершения ее работы и закрываем окно .

Проверка работоспособности

Запускаем проводник и в строке адреса вводим «http://localhost/training». Если все прошло нормально, мы увидим страницу с заголовком «Система Обучение».

Создание простого реестра

Итак, все готово для создания чего-нибудь полезного. Начнем с создания простого реестра, содержащего информацию о физических лицах.

Описание класса

В первой редакции информация о физическом лице будет состоять из имени, фамилии и даты рождения.

В ИнМете реестру соответствует понятие *класс*, а для обозначения характеристик объектов используется термин *свойство*. Для описания классов и свойств используется язык описания метаданных ИнМеты, основанный на XML.

В нашем примере в папке с метаданными мы создадим файл «training.xml»:

```
<?xml version="1.0" encoding="windows-1251"?>
<inmeta xml-builder-skin="inmeta">
</inmeta>
```

Не забудем зарегистрировать этот файл в «meta-app.xml»:

```
<?xml version="1.0" encoding="windows-1251"?>
<meta>
    <meta-data src="training.xml" />
    <meta-data src="_db_info.xml" />
    <meta-data src="_policies.xml" />
    <meta-data src="_roles.xml" />
    <meta-data src="_ui.xml" />
    <meta-data src="_users.xml" />
    <meta-data src="_denies.xml" />
    <meta-data src="_config.xml" />
    <meta-data src="_developer.xml" />
    <meta-data src="_gen_docs.xml" />
</meta>
```

Добавим описание класса физических лиц с тремя свойствами в файл «training.xml»:

```
<?xml version="1.0" encoding="windows-1251" ?>
<inmeta xml-builder-skin="inmeta">
    <class name="Person" caption="Физическое лицо">
        <property name="FirstName" data-type="string" caption="Имя" data-length="100"/>
        <property name="LastName" data-type="string" caption="Фамилия" data-length="100"/>
        <property name="BirthDate" data-type="datetime" caption="Дата рождения"/>
    </class>
</inmeta>
```

Для каждого элемента метаданных мы определили имя и заголовок. Имя будет использоваться для программного доступа. Заголовок для отображения интерфейса конечного пользователя.

Для каждого свойства мы указали тип данных (две строки и дата). Для строк мы дополнительно задали ограничение на максимальный размер строки.

Синхронизация базы данных

Теперь необходимо синхронизировать нашу базу данных с новыми метаданными приложения.

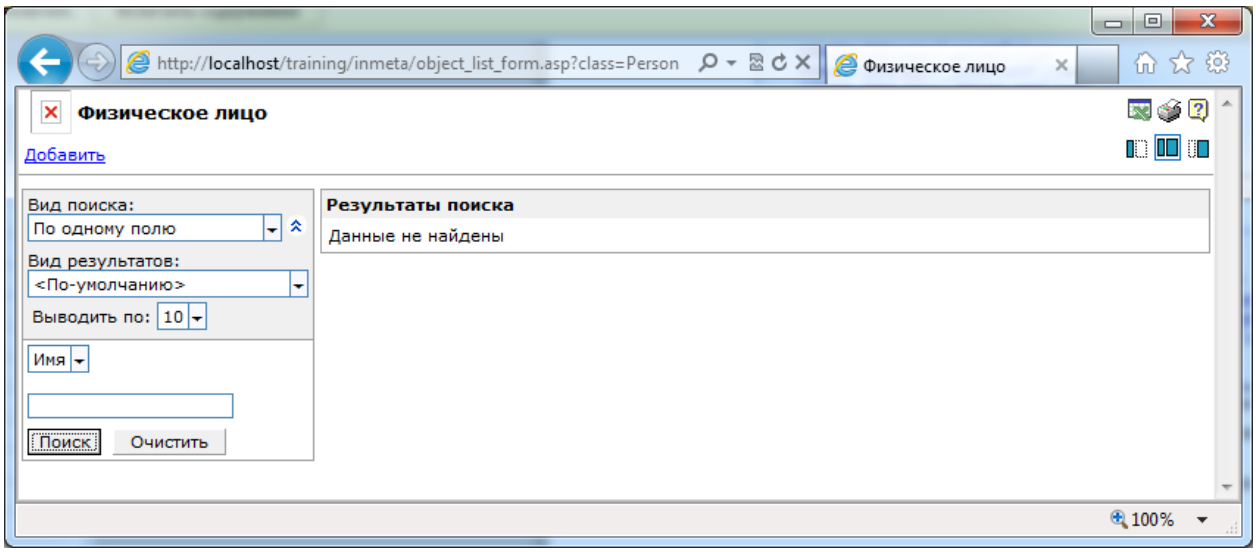
Запускаем утилиту «DBManager.exe», выбираем «Training», нажимаем «Далее».

В списке видим, что нужно создать новую таблицу «Person». Нажимаем «Обновить» и дожидаемся завершения. Все готово, в базе данных появилась новая таблица для размещения записей о физических лицах.

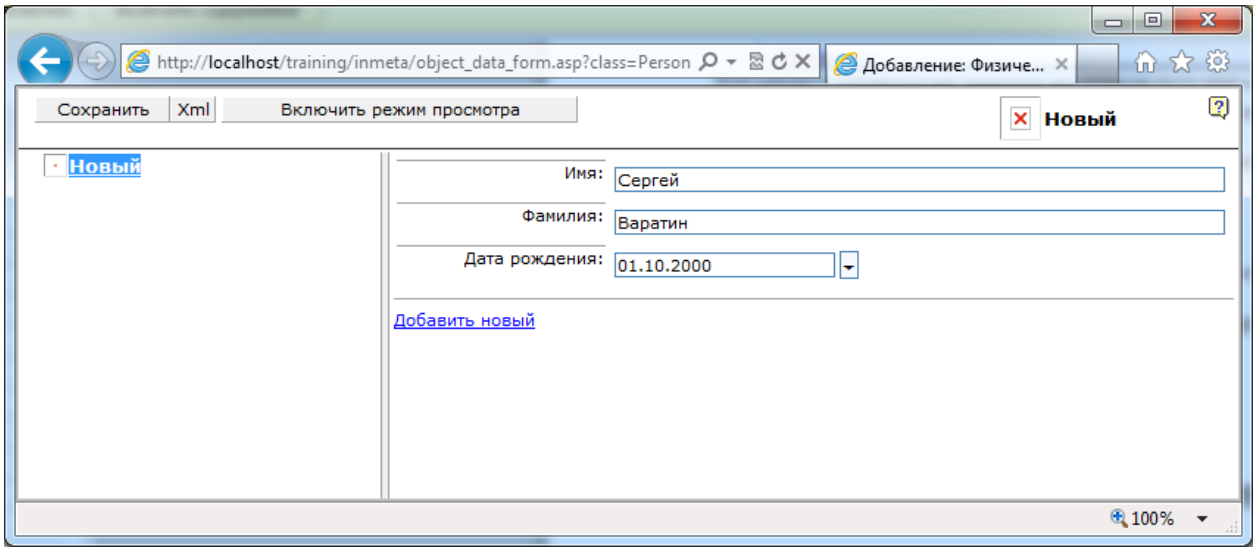
Проверка работоспособности нового реестра

Все готово для работы с новым реестром. В проводнике откроем универсальную поисковую форму «object_list_form.asp» с нашим реестром физических лиц.

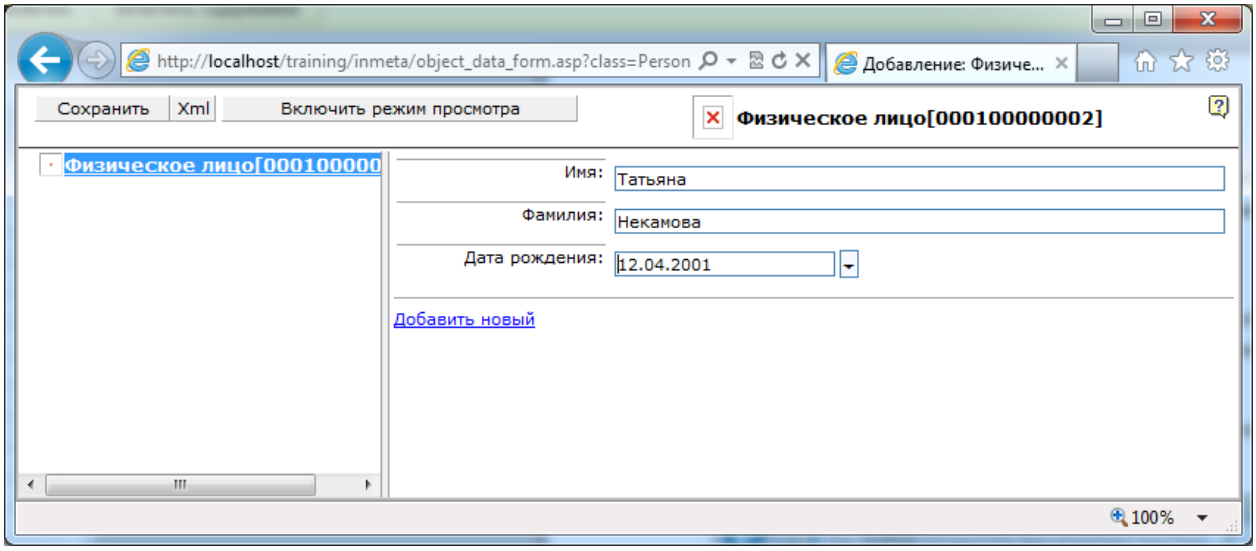
Поисковая форма находится по адресу «http://localhost/training/inmeta/object_list_form.asp?class=Person». Параметр «class» определяет реестр, с которым должна работать форма:



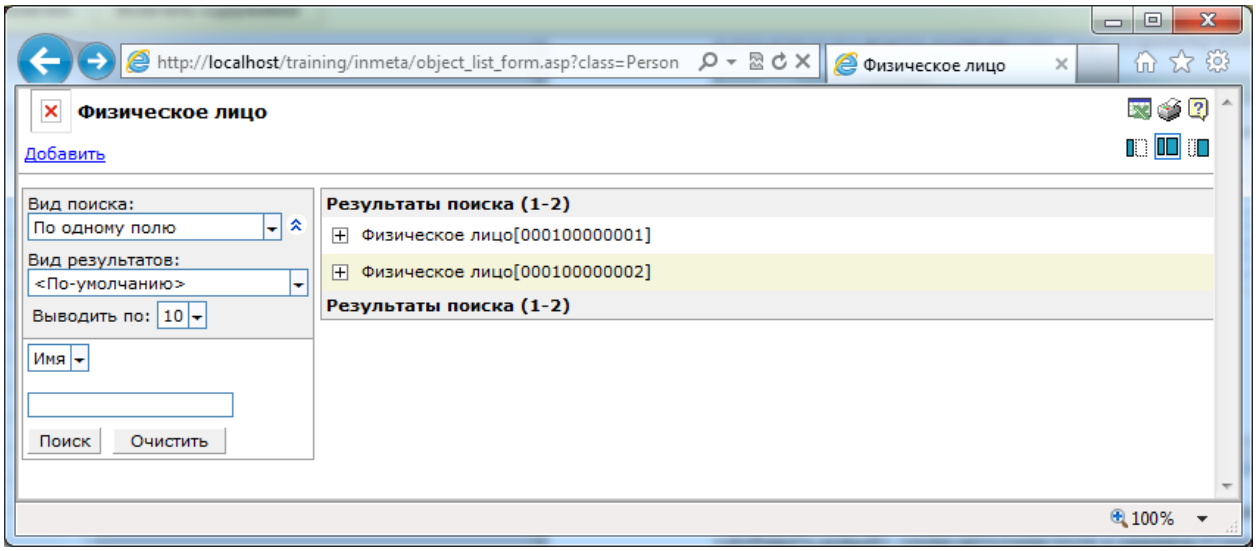
Заведем пару записей в нашем реестре. Для этого нажмем «Добавить» и попадем в универсальную форму ввода данных по реестровому объекту:



Заполним поля и нажмем «Сохранить». Добавим еще одну запись при помощи «Добавить новый», снова заполним поля и нажмем «Сохранить»:



Теперь можно вернуться в поисковую форму кнопочкой проводника «Назад» (большая круглая синяя кнопка в левом верхнем углу окна). В поисковой форме в списке результатов видим две записи:



Все неплохо, кроме одного: вместо имен и фамилия мы видим слово «Физическое лицо» с непонятными цифрами. О том, как это поправить, будем разбираться в следующем разделе.

А пока добавим ссылку на реестр физических лиц на главную страницу приложения «Default.asp», размещенную в папке «web»:

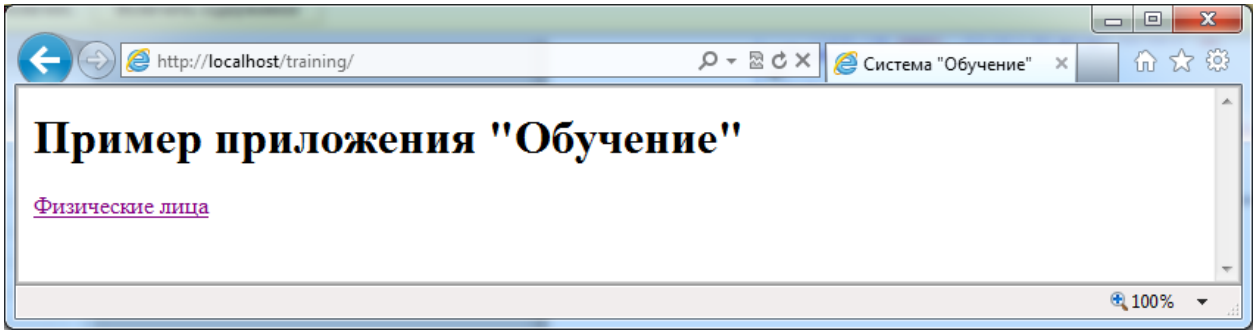
```
<html>

<head>
    <title>Система "Обучение"</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
    <meta http-equiv="Content-Language" content="ru">
</head>

<body>
    <h1>Пример приложения "Обучение"</h1>
    <a href="inmeta/object_list_form.asp?class=Person">Физические лица</a>
</body>

</html>
```

Теперь она выглядит вот так:



И по ссылке мы сразу попадаем в поисковую форму реестра физических лиц.

Создание представления объектов

Пришло время поправить отображение списка объектов на поисковой форме. В предыдущем разделе мы получили список без имен и фамилий.

Для отображения информации о реестровых объектах в формах интерфейса пользователя ИнМеты используются *представления объектов*.

Представление объекта – это функция, определенная в метаданных класса, предназначенная для преобразования значений свойств объекта в текст HTML. Функция представления объекта описывается в виде текста с *управляющими вставками*.

> *Текст с управляющими вставками* – текст, содержащий вставки на языке программирования. Управляющие вставки заключаются в последовательности <% *программа* %> или <%= *выражение* %>. Если вы программировали в среде ASP или JSP, то данный подход не вызовет трудности в понимании. Заметим, что тексты с управляющими вставками используются в ИнМете не только в представлениях объектов, но и в других элементах метаданных.
>
> Для программных вставок определено программное окружение, характерное для контекста, в котором используется подход. С одной стороны программное окружение предоставляет набор переменных и объектов, на основе которых формируется целевой текст. С другой стороны программное окружение предоставляет объект для программного добавления текста в выходной поток (целевой текст).
>
> Интерпретация выходного текста также зависит от контекста использования. К примеру, текст, сформированный функцией представления объекта, обычно является текстом HTML, а текст шаблона поиска является текстом запроса SQL.

В тексте представления объекта программное окружение предоставляет набор переменных *имяСвойстваProperty*, содержащих значения свойств объекта, для которого формируется текст представления объекта.

Существует два представления объектов с предопределенными именами: *default* и *description*, используемые по умолчанию. Если метаданные приложения не содержат определения этих представлений, они будут автоматически сгенерированы системой. Результат работы сгенерированных представлений мы наблюдали, когда в поисковой форме выводился заголовок класса и числовой идентификатор объекта.

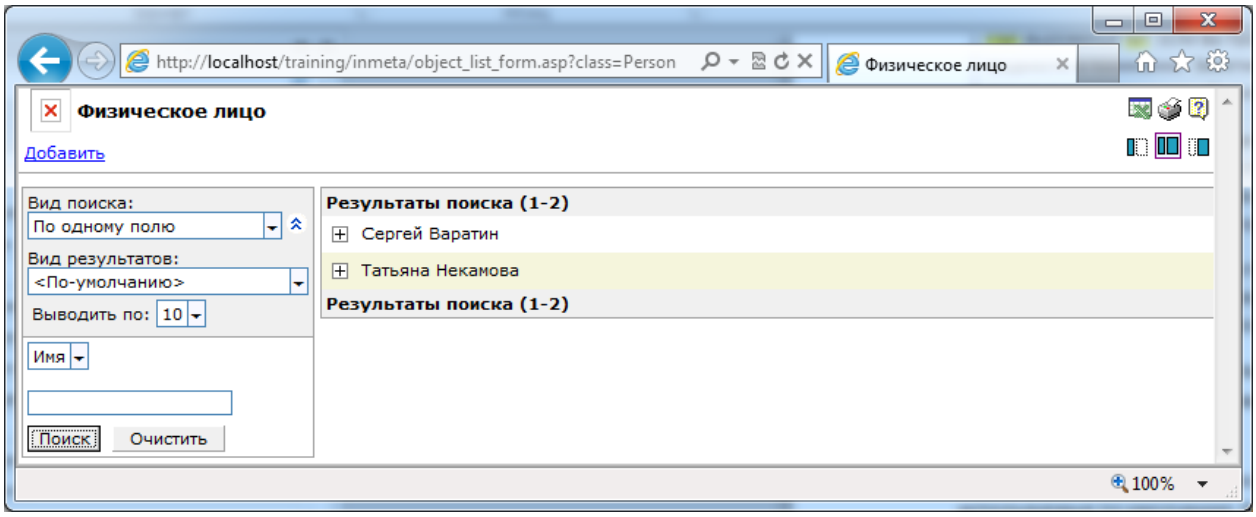
Пришло время определить нужное нам представление объекта для реестра физических лиц. Для этого внутри элемента класса добавим элемент *object-view*:

```
<?xml version="1.0" encoding="windows-1251" ?>
<inmeta xml-builder-skin="inmeta">
    <class name="Person" caption="Физическое лицо">
        <property name="FirstName" data-type="string" caption="Имя" data-length="100"/>
        <property name="LastName" data-type="string" caption="Фамилия" data-length="100"/>
        <property name="BirthDate" data-type="datetime" caption="Дата рождения"/>
        <object-view name="default" content-type="text">
            <![CDATA[<%=FirstNameProperty%> <%=LastNameProperty%>]]>
        </object-view>
    </class>
</inmeta>
```

В приведенном примере мы определили предопределенное представление объекта с именем *default*, и указали, что результатом работы функции является обычный текст (не HTML), в этом случае при отображении текста представления система будет заменять символы разметки HTML «<», «>» и пр., в соответствующие коды разметки «<», «>» и пр.

Для повышения читаемости текста представления мы использовали элемент *CDATA*, что позволило нам использовать в тексте управляющие символы XML «<» и «>».

Перезапустим приложение при помощи *reset.cmd* и откроем реестр физических лиц:



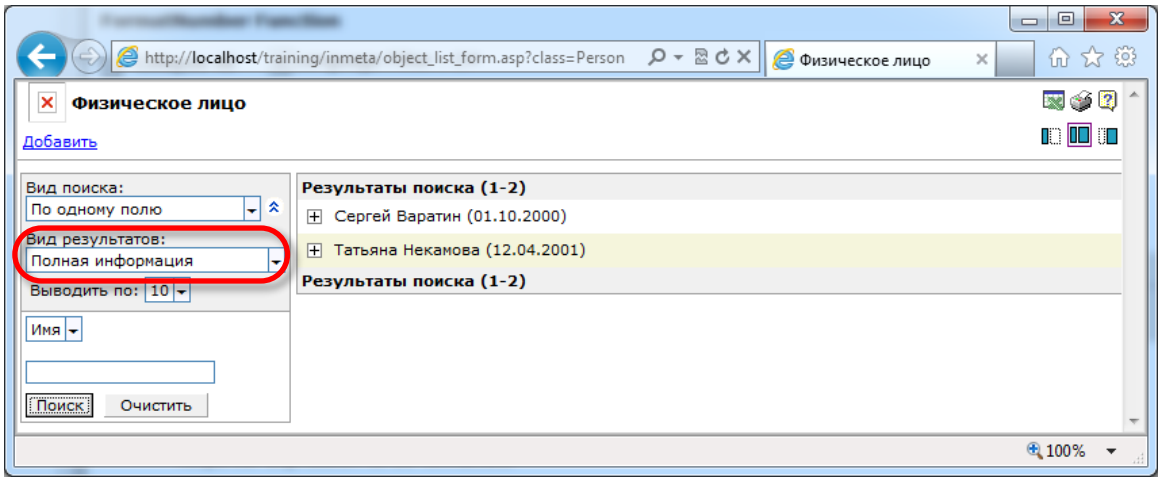
Получилось! Вместо чисел получили список имен и фамилий.

Теперь добавим еще одно представление объекта, которое будет возвращать полную информацию о физическом лице (включая дату рождения). Для этого в метаданные класса добавим еще один элемент *object-view*:

```
<object-view name="FullInfo" caption="Полная информация" content-type="text" is-internal="false">
    <![CDATA[<%=FirstNameProperty%> <%=LastNameProperty%> (<%=BirthDateProperty%>)]]>
</object-view>
```

Для нового представления мы задали имя «FullInfo», заголовок «Полная информация» и признак того, что представление будет доступно для выбора пользователем (*is-internal="false"*). В тексте представления мы добавили дату рождения, в скобках после имени и фамилии.

Перезагружаем приложение (*reset.cmd*), открываем поисковую форму, выбираем для отображения новое представление и смотрим результат поиска:



На этом месте отвлечемся от представлений объектов и перейдем к следующему разделу – создание поисковой формы.

Создание поисковой формы

По умолчанию система генерирует для нового реестра два способа поиска объектов: по одному полю и по нескольким полям. При поиске по одному полю у пользователя есть возможность выбрать поле и задать критерий поиска по этому полю. При поиске по нескольким поля пользователь может задать критерии поиска сразу для нескольких полей реестра, в этом случае будут найдены объекты, для которых удовлетворяются все заданные пользователем критерии.

Что делать, если необходим не предусмотренный системой вариант поиска? Например, в нашем примере, однострочный поиск по имени или фамилии физического лица.

В таких случаях ИнМета предоставляет возможность определить для класса нестандартные поисковые формы.

Реализация поисковой формы состоит из двух раздельных элементов метаданных:

- *Шаблон поиска* – текст с управляющими вставками, формирующий на выходе запрос SQL.
- *Шаблон формы поиска* – текст HTML, предназначенный для внедрения в панель реквизитов поиска.

Шаблон поиска

Шаблон поиска определяет запрос SQL, который должен вернуть список идентификаторов отобранных объектов. Помимо текста SQL, для шаблона поиска определяется набор именованных параметров. В зависимости от значений параметров может формироваться разный текст запроса SQL. Значения параметров доступны в программном окружении управляющих вставок текста шаблона.

В нашем примере внутри метаданных класса мы добавим новый элемент *sql-select-template*:

```
<sql-select-template name="ByFirstOrLastName" caption="По имени или фамилии">
    <![CDATA[
    SELECT [OID] id FROM Person
    WHERE [FirstName] LIKE '%<%=Name%>%' OR [LastName] LIKE '%<%=Name%>%'
    ]]>
    <param name="Name" data-type="string"/>
</sql-select-template>
```

Для запроса мы определили один параметр *Name* строкового типа, посредством которого будет передаваться значение фильтра по имени или фамилии. В тексте мы формируем запрос SQL, который извлекает идентификаторы из таблицы *Person*, для тех записей, в которых имя или фамилия подобна значению параметра *Name*.

Например, если параметр *Name* содержит строку «*сер*», в результате сформируется следующий запрос SQL:

```
    SELECT [OID] id FROM [Person]
    WHERE [FirstName] LIKE '%сер%' OR [LastName] LIKE '%сер%'
```

Результат выполнения запроса обязательно должен содержать столбец с именем *id* со значениями идентификаторов найденных объектов.

Имена полей и таблиц базы данных следует заключать в квадратные скобки.

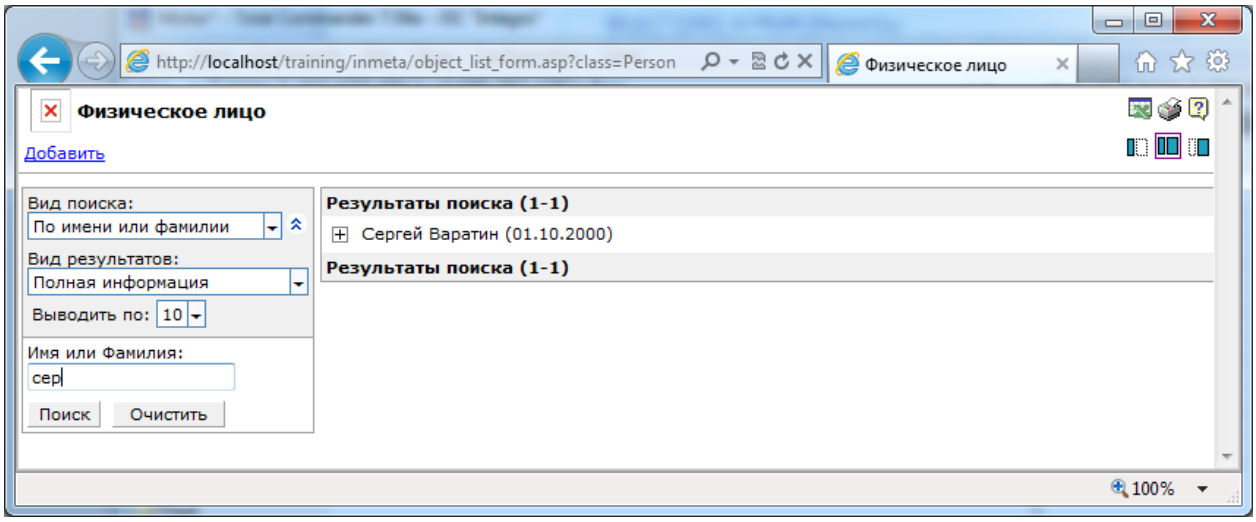
Шаблон формы поиска

Теперь необходимо спроектировать код разметки HTML для панели поиска. Для этого в метаданных класса добавим еще один элемент *search-form-template*:

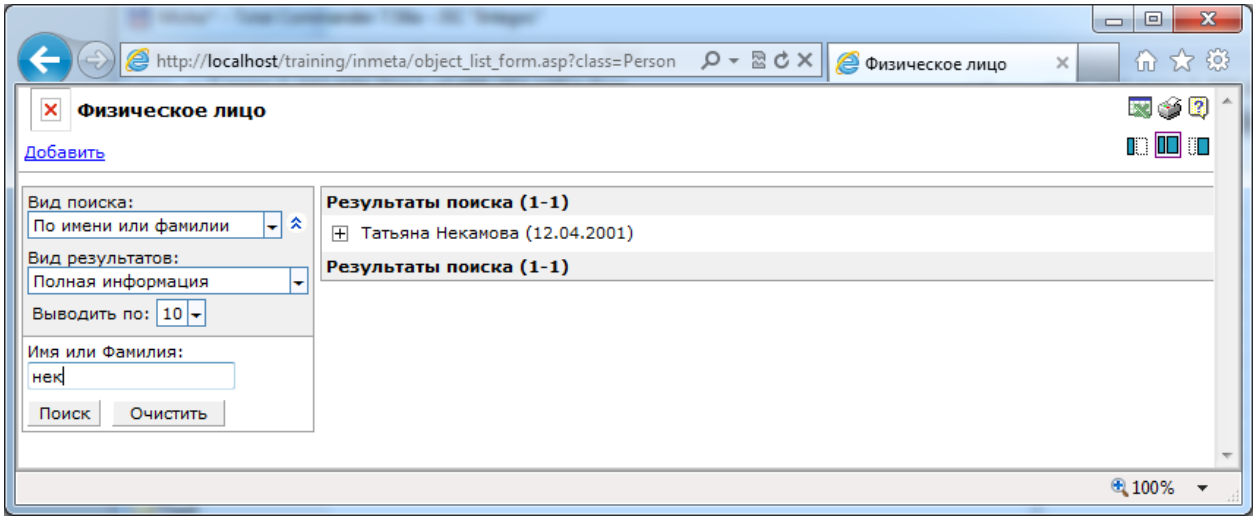
```
<search-form-template name="ByFirstOrLastName" caption="По имени или фамилии"
    sql-select-template="ByFirstOrLastName">
    <![CDATA[Имя или Фамилия:<br><input type=text inmeta-param=Name>]]>
</search-form-template>
```

Здесь мы определили шаблон формы поиска и указали, для поиска объектов будет использоваться шаблон поиска *ByFirstOrLastName*. В разметке формы мы задали поле ввода и при помощи атрибута *inmeta-param* связали это поле ввода с параметром запроса *Name*.

Перезагружаем приложение, открываем реестр, выбираем вид поиска «По имени или фамилии» и ищем по строке «сер»:



Теперь ищем по строке «нек»:



Наша форма поиска работает.

Несколько слов о том, как работает конвейер поисковой формы:

1. Пользователь выбирает вид поиска «По имени и фамилии». Система берет из метаданных соответствующую разметку и внедряет ее в панель реквизитов поиска.

2. Пользователь нажимает кнопку «Поиск». Система ищет в разметке формы все поля ввода, у которых задан атрибут *inmeta-param*. С каждого найденного поля снимается значение, введенное пользователем, и запоминается для передачи в качестве значения соответствующего параметра шаблона поиска.
3. Набор значений параметров, введенных пользователем, отправляется на сервер для выполнения запроса.
4. На сервере формируется текст запроса SQL на основе шаблона поиска с подстановкой значений параметров, присланных с клиента.
5. Сформированный запрос SQL выполняется на сервере СУБД и возвращает список идентификаторов найденных объектов.
6. Для всех найденных объектов формируются тексты представлений объектов и возвращаются клиенту.
7. На клиенте присланные результаты поиска отображаются в панели результатов формы поиска.

На этом будем считать создание простого реестра завершенным и перейдем к созданию более сложных конструкций.

Создание дочернего и справочного реестров

Усложним задачу: добавим для физического лица список адресов, при этом в адресе должен использоваться справочник улиц.

Для этого нам придется создать еще два реестра: реестр адресов, дочерний по отношению к физическим лицам и справочный реестр улиц.

Создание справочного реестра

Начнем с более простой задачи: создания справочного реестра улиц. Для этого добавим в файл метаданных «training.xml» класс *Street*:

```
<class name="Street" caption="Улица" list-caption="Реестр улиц">
    <property name="Name" caption="Название" data-type="string" data-length="100"/>
</class>
```

На главной странице приложения «Default.asp» добавим ссылку на новый реестр:

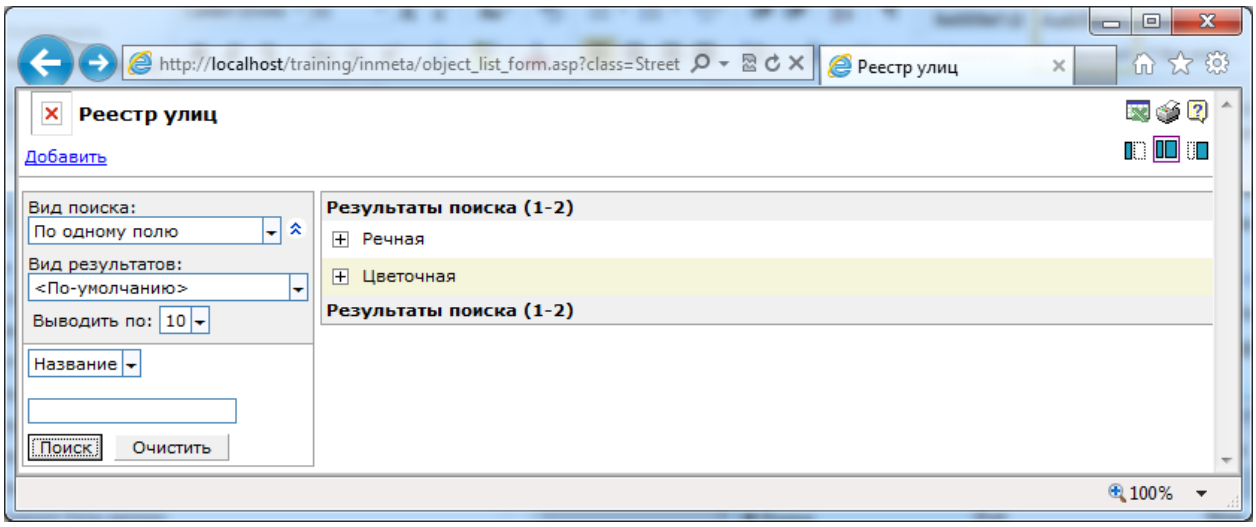
```
<html>

<head>
    <title>Система "Обучение"</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
    <meta http-equiv="Content-Language" content="ru">
</head>

<body>
    <h1>Пример приложения "Обучение"</h1>
    <a href="inmeta/object_list_form.asp?class=Person">Физические лица</a><br>
    <a href="inmeta/object_list_form.asp?class=Street">Справочник улиц</a>
</body>

</html>
```

Перезапускаем приложение при помощи «restart.cmd», синхронизируем базу данных при помощи «DBManager.exe», открываем главную страницу приложения «http://localhost/training», переходим в справочник улиц и добавляем две улицы:



Неплохо. Теперь у нас есть реестр улиц с двумя записями, который мы можем использовать в реестре адресов.

> Обратите внимание, что хотя мы не определяли представления объекта для класса *Street*, в поисковой форме отображаются названия улиц, а не имя класса с идентификатором объекта (как в случае с реестром физических лиц).
>
> Почему так получилось? Каждый раз при перезагрузке приложения происходит внутренняя компиляция метаданных. При компиляции система автоматически дополняет метаданные необходимыми атрибутами и элементами. В частности, для каждого класса формируется представление объекта с именем *default* (если оно не было определено в исходных метаданных).
>
> Если в классе определено свойство с именем *Name*, то будет сформировано представление, возвращающее значение этого свойства. В противном случае представление *default* возвращает заголовок класса и идентификатор объекта в квадратных скобках.

Создание дочернего реестра

Создание дочернего реестра адресов делается аналогично созданию обычного реестра. Для этого определим в метаданных новый класс *Address*:

```
<class name="Address" caption="Адрес" list-caption="Адреса">
    <property purpose="aggregation" name="Owner" caption="Владелец" ref-class="Person"/>
    <property purpose="association" name="Street" caption="Улица" ref-class="Street"/>
    <property name="Building" caption="Номер" data-type="string" data-length="20"/>
    <property name="Flat" caption="Квартира" data-type="string" data-length="20"/>
</class>
```

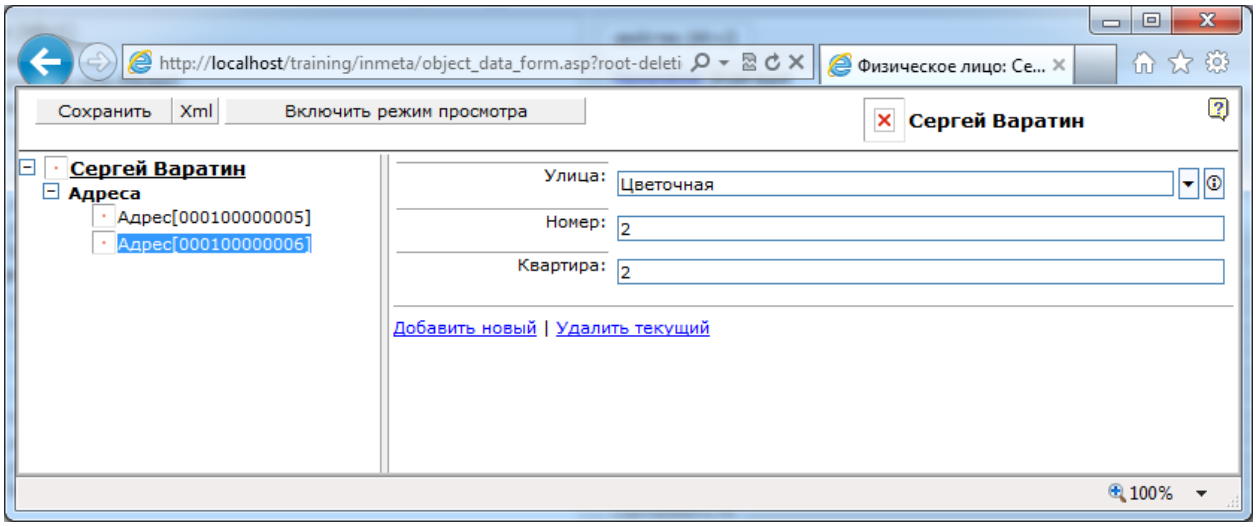
В приведенном примере мы определили класс Address с пятью свойствами. Рассмотрим подробнее, что именно мы определили.

Свойства *Building* и *Flat*: в них пользователь будет вводить номер дома и квартиры соответственно. Свойства простые строковые, на них заострять внимание не будем.

Свойство *Street*: описывает отношение *ассоциации* между реестром адресов и реестром улиц (ссылка в справочник улиц со стороны адреса). О том, что данное свойство является отношением, указывает атрибут *purpose* со значением *association*. Имя целевого класса для ссылки задается атрибутом *ref-class* со значением *Street*.

Свойство *Owner*: описывают отношение агрегирования между реестром адресов и реестром физических лиц (адреса являются дочерними объектами физического лица). На факт отношения указывает атрибут *purpose* со значением *aggregation* у свойства *Owner*. Очень похоже на отношение ассоциации *Street*.

Перезапускаем приложение, синхронизируем базу данных, открываем реестр физических лиц и заполняем адреса для Сергея Варатина:

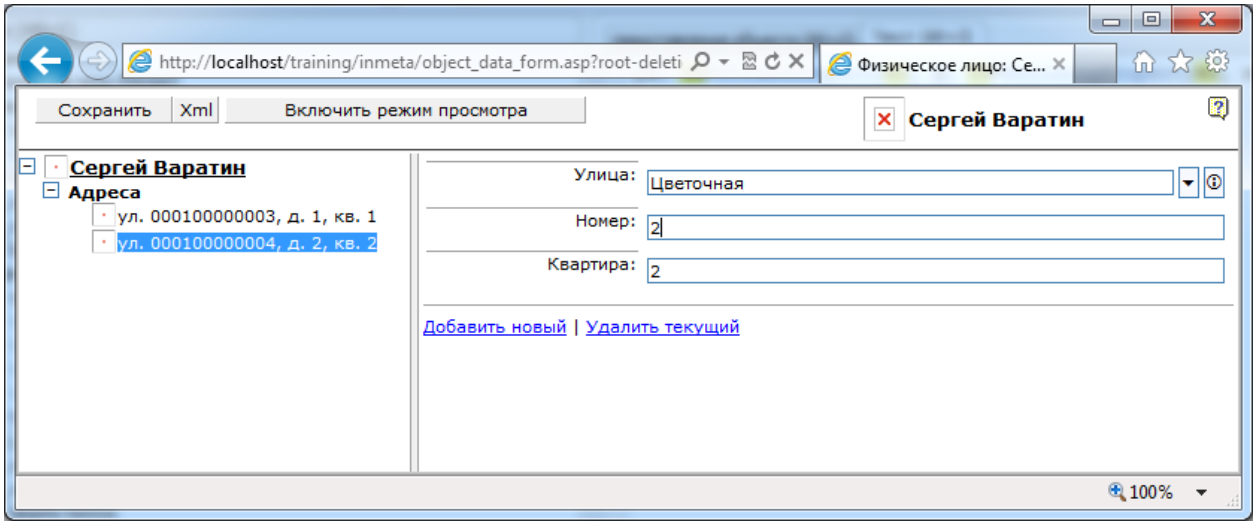


Непорядок. Вместо адресов опять выводится ненужная пользователю информация. Нужно поправить.

Определим для класса *Address* представление объекта *default*:

```
<object-view name="default">
    <![CDATA[ул. <%=StreetProperty%>, д. <%=BuildingProperty%>, кв. <%=FlatProperty%>]]>
</object-view>
```

Перезапускаем приложение, открываем форму с Варатиным:



Стало лучше, но не полностью: вместо названия улицы выводится идентификатор.

Создание представлений с информацией о справочном объекте

Как в представлении адреса вместо идентификатора улицы вывести ее представление?

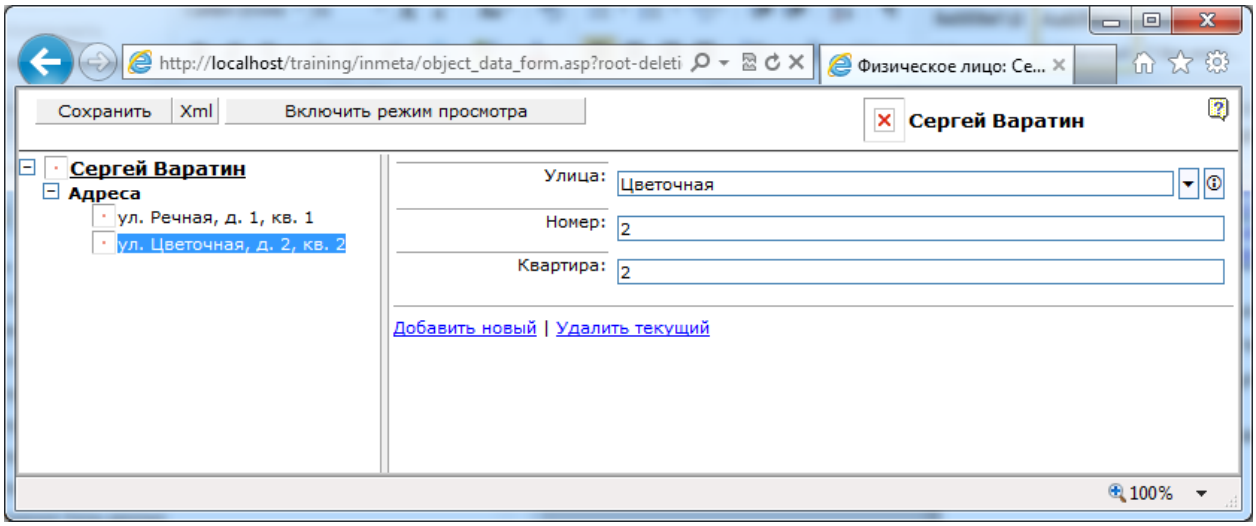
Для этого доработаем представление *default* у класса *Address*:

```
<object-view name="default">
    <virtual-property name="StreetView" association="Street" ref-view="default"/>
    <![CDATA[ул. <%=StreetViewProperty%>, д. <%=BuildingProperty%>, кв. <%=FlatProperty%>]]>
</object-view>
```

Мы добавили элемент *virtual-property* с именем *StreetView*. Мы указали, что для получения значения виртуального свойства нужно перейти по ассоциации *Street* (*association*="*Street*") и взять у связанного объекта представление *default* (*ref-view*="*default*").

Значение виртуального свойства будет доступно внутри управляющих вставок под именем *StreetViewProperty*.

Перезапускаем, смотрим результат:



То, что нужно.

Создание представлений с информацией о дочерних объектах

Еще небольшой штрих к нашему приложению: у физических лиц добавим представление, в котором отображаются адреса. Назовем его *NamesWithAddress* (для пользователя зададим заголовок «ФИО с адресами»):

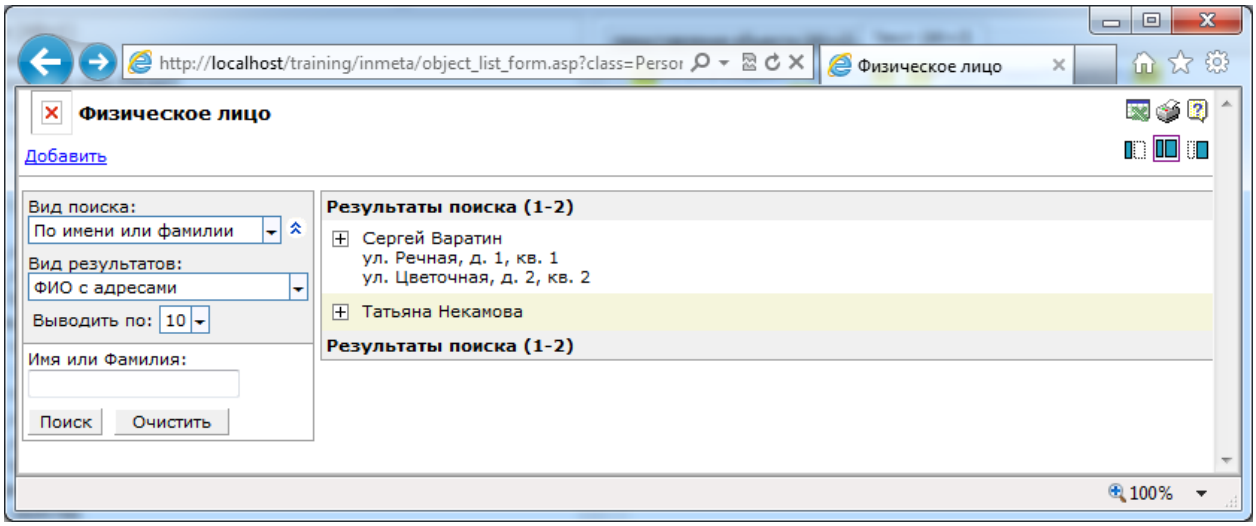
```
<object-view name="NamesWithAddress" caption="ФИО с адресами" is-internal="false">
    <virtual-property name="Address" aggregation="Address" ref-view="default"/>
    <![CDATA[
    <%=FirstNameProperty%> <%=LastNameProperty%>
    <%dim address%>
    <%for each address in AddressProperty%>
    <br><%=address%>
    <%next%>
    ]]>
</object-view>
```

Для того, что бы в управляющих вставках мы могли использовать связанные адреса, мы определили виртуальное свойство *Address*, для которого указали переход на агрегированные объекты класса *Address* (*aggregation*="*Address*"), и заказали представление *default* (*ref-view*="*default*").

Значения виртуальных свойств, привязанных к агрегатам, всегда являются массивом. Если у объекта нет ни одного дочернего объекта, значение виртуального свойства будет содержать пустой массив.

В тексте представления мы добавили скриптовые вставки для перечисления элементов массива с адресами и вывода каждого адреса в итоговый текст.

Перезапускаем, открываем реестр физических лиц, выбираем вид результатов «ФИО с адресами», жмем «Поиск», смотрим результаты:



Небольшой штрих удался. Сергей Варатин отображается вместе с адресами.

Создание поисковой формы с реквизитами дочернего объекта

Осталось добавить возможность поиска физических лиц по адресу.

Начнем с шаблона поиска. Добавим в класс *Person* новый шаблон поиска с именем *ByAddress*:

```
<sql-select-template name="ByAddress" caption="По адресу">
    <param name="Street" data-type="string"/>
    <param name="Building" data-type="string"/>
    <param name="Flat" data-type="string"/>
    <![CDATA[
    SELECT
        [Person].[OID] id
    FROM
         [Person]
        LEFT OUTER JOIN [Address] ON [Address].[Owner] = [Person].[OID]
        LEFT OUTER JOIN [Street] ON [Street].[OID] = [Address].[Street]
    WHERE
        [Street].[Name] LIKE '%<%=Street%>%'
        AND [Address].[Building] LIKE '%<%=Building%>%'
        AND [Address].[Flat] LIKE '%<%=Flat%>%'
    ]]>
</sql-select-template>
```

Для шаблона мы определили три параметра: *Street*, *Building* и *Flat* для поиска по улице, номеру дома и квартиры. Все три параметра – строковые.

Затем мы подготовили текст шаблона поиска, который после подстановки значений параметров преобразуется в текст запроса SQL. В запросе мы делаем выборку из трех присоединенных при помощи *JOIN* таблиц: *Person*, *Address* и *Street*.

> Все ссылки в ИнМете реализуются посредством внутренних идентификаторов объектов. В каждом реестре присутствует специальное поле *OID*, в котором содержится уникальный внутренний идентификатор реестрового объекта. Для каждого свойства, являющегося отношением ассоциации или агрегации, в таблице реестра заводится одноименное поле, которое содержит значение идентификатора связанного объекта.

В нашем примере мы присоединили записи из таблицы *Address*, у которых значение поля *Owner* совпадает с идентификатором физического лица (LEFT OUTER JOIN [*Address*] ON [*Address*].[*Owner*] = [*Person*].[*OID*]).

Затем мы присоединили записи из таблицы *Street*, идентификатор которых совпадает со значением поля *Street* таблицы *Address* (LEFT OUTER JOIN [*Street*] ON [*Street*].[*OID*] = [*Address*].[*Street*]).

Ну и наконец, мы задали условие отбора по интересующим нас полям.

Следует заметить, что запрос SQL должен возвращать набор идентификаторов из той таблицы, которая относится к реестру, для которого определяется шаблона поиска. В нашем случае это таблица Person, относящаяся к реестру физических лиц.

> Обратите внимание, что все названия таблиц и полей базы данных заключены в квадратные скобки. Это важно, так как перед выполнением запроса на сервере базы данных, ИнМета заменяет квадратные скобки ограничителями идентификаторов, принятыми в используемой СУБД, обычно это символы двойных кавычек.

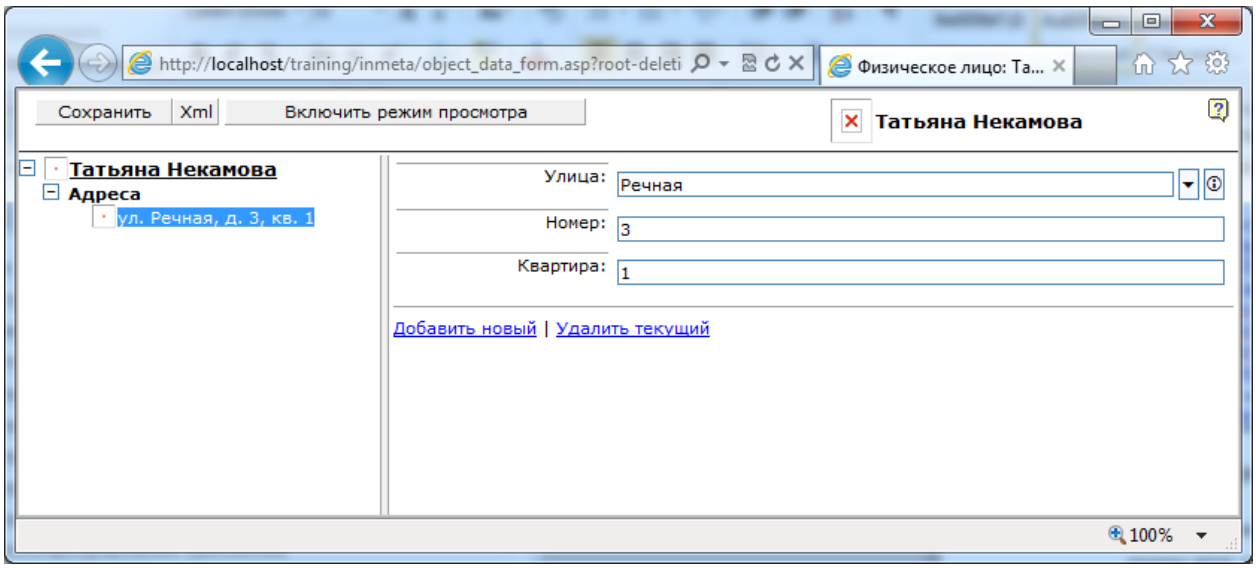
Шаблон готов, теперь определим саму форму поиска. Добавим в класс *Person* элемент *search-form-template*:

```
<search-form-template name="ByAddress" caption="По адресу" sql-select-template="ByAddress"
    object-view="NamesWithAddress">
    <![CDATA[
    <table border=0>
    <tr><td>Улица<td><input type=text inmeta-param=Street>
    <tr><td>Дом<td><input type=text inmeta-param=Building>
    <tr><td>Квартира<td><input type=text inmeta-param=Flat>
    </table>
    ]]>
</search-form-template>
```

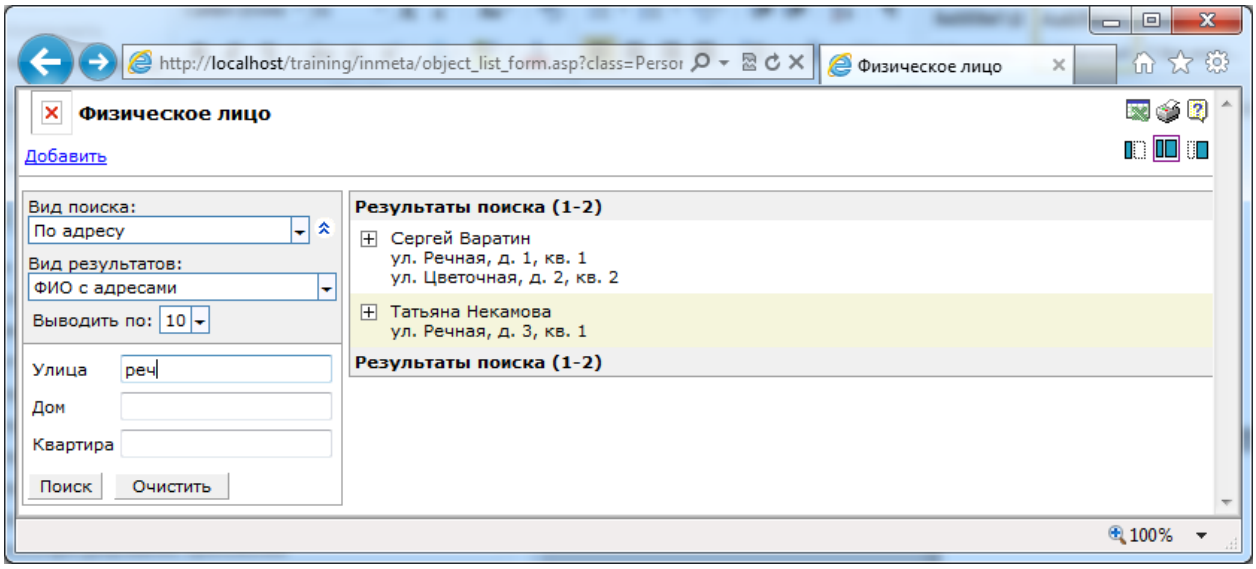
Здесь все просто – несложная разметка с привязкой полей ввода к параметрам шаблона поиска.

Единственное новшество – атрибут *object-view* со значением *ByAddress*. Если для формы поиска задан этот атрибут, то при выборе пользователем вида поиска на форме автоматически выберется указанный вид результатов.

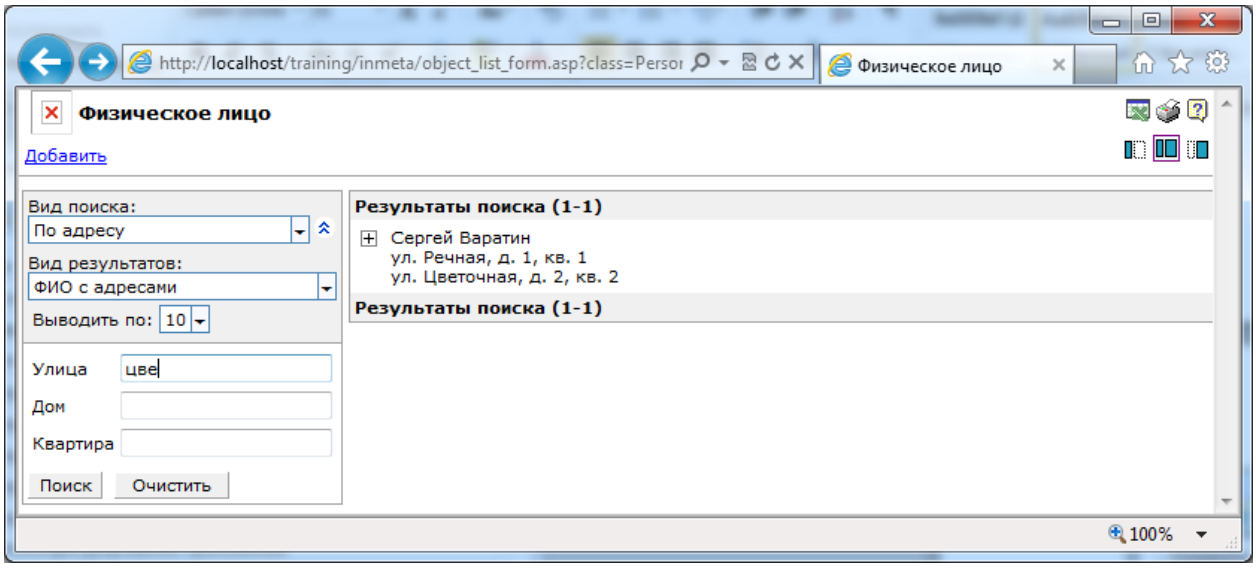
Перезапускаем и открываем реестр физических лиц. Откроем форму редактирования для Татьяны Некамовой и добавим один адрес:



Сохраняем изменения и возвращаемся на поисковую форму. Выбираем вид поиска «По адресу» и проверяем, как оно работает:



Теперь пробуем «цве»:



Оно работает.

Отлично, у нас есть небольшая, но работающая информационно поисковая система.