Программный комплекс **«**Бизнес Семантика**»**

Документация

разработчика и администратора

версия **1.0**

Содержание

I. Архитектура системы

1.1. Основные компоненты системы

Набор программных продуктов «Бизнес Семантика» состоит из сервера и клиентов (или коннекторов). **Сервер** представляет собой автономное веб-приложение, которое в текущей версии реализовано на языке **PHP 5.2** и использует базу данных **MySQL 5.1** или старше. **Коннекторы** могут представлять собой полноценные приложения, или фрагменты кода, встраиваемые в код других систем. Первый тип клиентов называется автономными коннекторами, второй – встроенными. Ниже приведен список коннекторов, реализованных в текущей версии продукта:

Наименование	Тип	Язык реализации	База данных
Веб-коннектор	Автономный	**PHP**	**MySQL** (возможно – **PostgreSQL, Oracle**)
1С 8.1	Встроенный	**1С, C++**	Нет
1С 8.2	Встроенный	**1С, C++**	Нет
1С 7.7	Встроенный	**1С, C++**	Нет

В процессе реализации – коннекторы для **Oracle, MS SQL Server, Microsoft SharePoint.**

Сервер обслуживает запросы клиентов. Предусмотрены следующие варианты запросов:

Наименование	Назначение
login	Авторизация клиента
pushInfo	Передача данных о созданных/изменившихся объектах на сервер
readTriples	Запрос данных о созданных, изменившихся, удаленных объектах от сервера
requestObject	Запрос информации о конкретном объекте
deleteObject	Удаление конкретного объекта
sendStat	Отправка статистики сеанса
reportError	Сообщение об ошибке

Клиенты выступают как активные компоненты, которые обращаются к серверу для передачи информации о том, какие данные изменились в их информационной системе (далее ИС), а также для запроса данных, подлежащих передаче в их ИС.

1.2. Используемые технологии

Базовая терминология, используемая в ПО «Бизнес Семантика», включает следующие определения:

ИС – информационная система, чаще всего использующая базу данных (БД) для хранения обрабатываемой информации. ИС предоставляет определенную функциональность пользователю. Задача ПО «Бизнес Семантика» состоит в организации автоматического, прозрачного обмена данными между различными ИС.

Тип объекта – какой-либо вид информационных сущностей (например, клиенты), с которым работает та или иная ИС. Обычно в рамках ИС типу объекта соответствует справочник, журнал документов, реестр и т.д.

Объект – некая информационная сущность (например, клиент), присутствующая в какой-либо ИС. Чаще всего (но не обязательно!) объект соответствует одной записи в таблице БД.

Свойство – характеристика объекта (например, название или ИНН клиента). Обычно соответствует значению какого-либо столбца определенной записи в БД.

Триплет – выражение вида «подлежащее сказуемое дополнение», которое сообщает какой-либо факт о каком либо объекте. Пример триплета: «Клиент «ООО Астра» имеет ИНН, равный **6671034959**». Здесь «Клиент «ООО Астра»» - это объект, или подлежащее (на самом деле, в информационной системе объект идентифицируется не названием, а идентификатором ресурса – **URI**), «имеет ИНН» - сказуемое, обозначающее, что объект обладает определенным свойством, и «**6671034959**» - дополнение, которое в данном случае является литералом (а может являться, например, ссылкой на другой объект).

Набор программного обеспечения «Бизнес Семантика» использует стек следующих технологий:

SOAP – для передачи информации между клиентом и сервером (в приведенной выше таблице значение в столбце «Наименование» соответствует методу веб-сервиса со стороны сервера).

RDF – язык записи фактов (триплетов), при помощи которого выражены все передаваемые между сервером и клиентом данные.

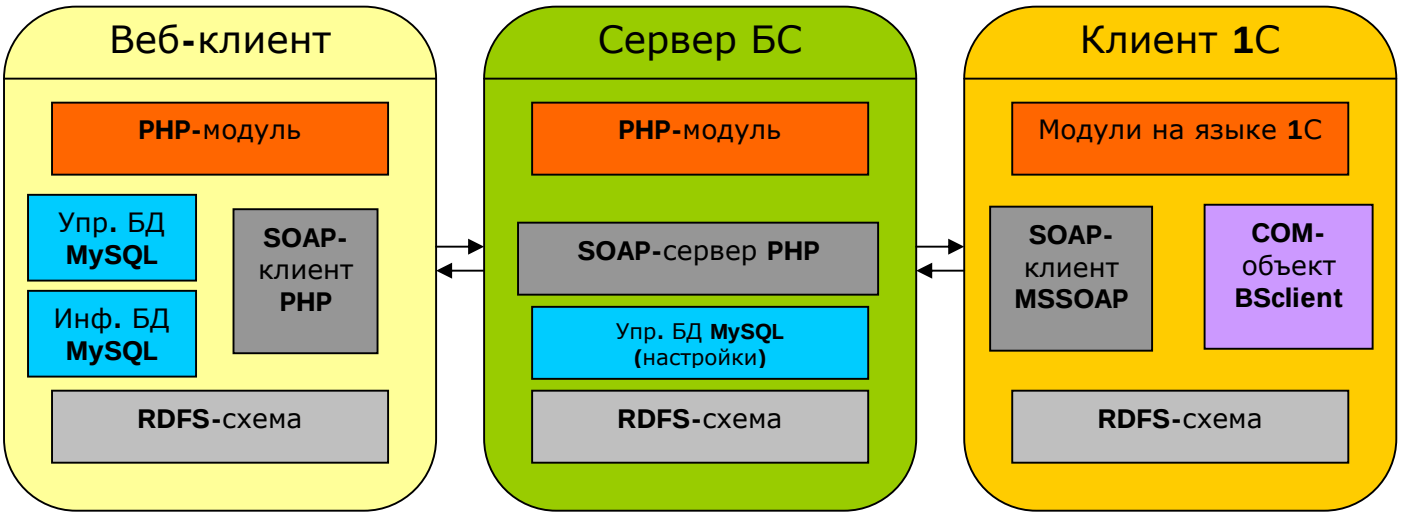
RDFS – язык описания схемы **RDF**. Содержит определения всех используемых каждой ИС объектов, свойств и их типов.

Turtle – один из синтаксисов записи **RDF**-выражений. В «Бизнес Семантике» в этом синтаксисе передаются данные об объектах между клиентом и сервером. Указанный выше факт в этом синтаксисе может быть записан так (предположим, что идентификатор сущности «клиент ООО «Астра» - **Client459**):

```
<#Client459> rdf:type Client ;
main:inn "ООО ""Альфа""";
.
```

COM – интерфейс взаимодействия с программными компонентами в **Microsoft Windows**. Коннекторы, написанные на встроенном языке **1С**, используют **COM**-объект **MSSOAP** для подключения к веб-сервису. Кроме того, для облегчения парсинга данных в синтаксисе **Turtle** существует написанный на **C++** компонент **BSclient**, который также можно использовать как **COM**-объект.

Диаграмма взаимодействия программных средств при передаче данных между веб-клиентом «Бизнес Семантика» и коннектором, встроенным в конфигурацию **1С**, посредством сервера «Бизнес Семантика», выглядит так:



Внутренняя структура, и взаимодействие программных компонентов внутри каждого модуля, детально рассматривается ниже. Данная диаграмма позволяет понять связь компонентов на макро-уровне, и подчеркнуть следующие факты:

- Каждый компонент содержит управляющий программный модуль, реализующий логику обмена данными. Модуль может быть написан на любом языке программирования.
- Каждый коннектор должен использовать модуль **SOAP**-клиента для того, чтобы иметь возможность обмениваться информацией с сервером.
- Каждый компонент должен содержать схему той информации, которой он обменивается с сервером. Схема обычно представлена в виде файла **RDFS**, но может загружаться в базу данных для удобства программного доступа. При этом схема, используемая каждым клиентом, является подмножеством схемы, используемой сервером. Таким образом, схема, которую использует сервер, должна содержать описания всех объектов и свойств, которыми обмениваются ИС.
- Веб-клиент использует две базы данных **MySQL**: в одной он хранит настройки, другая содержит собственно информацию, подлежащую интеграции с другими системами.
- Клиент, встраиваемый в конфигурации **1**С, использует **COM**-объект **BSclient** для удобства работы с данными в синтаксисе **Turtle**.

Принципиальная схема работы клиентского компонента представлена на следующем рисунке:

Таким образом, клиентский компонент преобразует сведения об изменениях в данных, представленных в реляционной форме (хранящихся в базе данных ИС), в семантическую форму, описанную выше, а также выполняет обратное преобразование.
Более подробная логическая схема способа работы клиентского компонента приводится далее.

Принципиальная схема сервера «Бизнес Семантика» выглядит так:

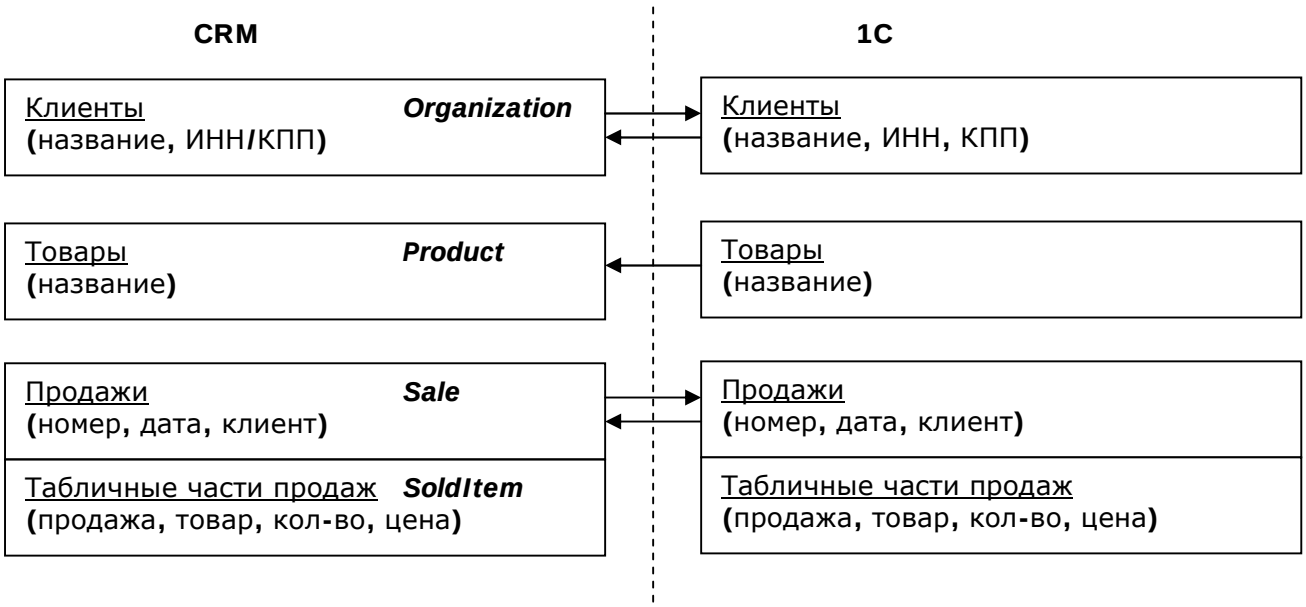
Модуль обмена данными со стороны сервера принимает от клиентского компонента ИС информацию, выраженную в виде триплетов (в качестве параметра какого-либо метода **SOAP**-интерфейса). Полученная информация сверяется со схемой и правилами, реализованными в модуле контроля прав доступа, и попадает в хранилище триплетов (может быть реализовано при помощи реляционной БД, или **SPARQL**-сервера). При поступлении запроса от клиентской ИС на получение данных, изменившихся в других ИС, сервер возвращает информацию из хранилища триплетов, которая ранее не передавалась данной ИС. При этом используется механизм транзакций (описывается ниже), и происходит сверка с настройками модуля контроля прав доступа. Сервер не производит никакого преобразования данных – он только помещает информацию, сообщенную клиентскими ИС, в хранилище триплетов, и по запросу извлекает ее оттуда. Информация выражена в семантической форме (триплетах) в соответствии с универсальной для данной системы онтологией, хранящейся на сервере.

Администратор системы, при помощи административного интерфейса, может задавать настройки модуля контроля прав доступа, а также создавать и редактировать онтологию, в которой выражены данные, с которыми работает сервер.

II. Проектирование архитектуры обмена данными

2.1. Составление RDFS-схемы

И сервер, и клиент БС должны иметь в своем распоряжении схему данных, которыми они ведут обмен. Чаще всего схема хранится в виде текстового файла; она может загружаться в БД при каждой обработке, или считываться из файла.
Начать работу лучше с составления диаграммы, которая покажет, какими данными в каком направлении должны обмениваться ИС. Для каждого вида данных нужно определить, какая (или какие) ИС будет являться его «хозяином», то есть будут обладать правом создания новых объектов такого типа, какие – смогут получать информацию об этих объектах. Дальше диаграмма детализируется на уровне свойств объектов – мы имеем возможность передавать разным ИС различные наборы свойств того или иного объекта. Например, диаграмма может выглядеть так:



Как видно из этой диаграммы, информация о клиентах и продажах (включая их табличные части – списки товаров) может передаваться как из **1С** в **CRM**, так и в обратном направлении, а информация о товарах – только из **1**С в **CRM**. Структура информации обо всех объектах практически идентична, за исключением нюанса, связанного с хранением полей ИНН/КПП: в **1**С это два разных поля, в **CRM** – одно.

Составим **RDFS**-схему для такой структуры обмена информацией.
Схема начинается со стандартных заголовков:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://localhost/schemas/common">
```

Затем следуют описания типов объектов (классов, в терминологии **RDFS**). Классы могут иметь иерархию. В нашем случае мы можем предположить, что наряду с клиентами-юридическими лицами могут существовать и клиенты-физические лица; мы определим класс **Agent**, который будет родоначальником для классов **Organization** (клиент – юр. лицо) и **Person** (клиент – физ.

лицо, который в контексте нашего примера мы пропустим). Названия классов показаны в правом верхнем углу блоков диаграммы выше.

```
<rdfs:Class rdf:ID="Agent"/>
<rdfs:Class rdf:ID="Sale"/>
<rdfs:Class rdf:ID="Product"/>
<rdfs:Class rdf:ID="SoldItem"/>

<rdfs:Class rdf:ID="Organization">
  <rdfs:subClassOf rdf:resource="#Agent"/>
</rdfs:Class>
```

Далее следует блок с определением типов данных. В настоящее время БС поддерживает четыре стандартных типа данных, поэтому блок всегда будет выглядеть одинаково:

```
<rdfs:Datatype rdf:about="&xsd;integer"/>
<rdfs:Datatype rdf:about="&xsd;string"/>
<rdfs:Datatype rdf:about="&xsd;double"/>
<rdfs:Datatype rdf:about="&xsd;date"/>
```

Ниже следует описание свойств всех объектов. Одни свойства могут представлять собой простые значения:

```
<rdf:Property rdf:ID="INN">
  <rdfs:domain rdf:resource="#Agent"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
```

Другие – быть ссылками на иные объекты:

```
<rdf:Property rdf:ID="client">
  <rdfs:domain rdf:resource="#Sale"/>
  <rdfs:range rdf:resource="#Organization"/>
</rdf:Property>
```

В конце файла необходимо закрыть тег:

```
</rdf:RDF>
```

Созданную таким образом схему нужно загрузить в сервер БС (этот процесс описан ниже). Затем из нее, путем удаления тех или иных фрагментов, получаем схемы для всех коннекторов (в нашем примере схемы для коннекторов будут идентичны; ситуация с различием схем для коннекторов может возникнуть только при трех и более интегрируемых системах).

2.2. Определение прав доступа

Права доступа определяют набор операций, который каждая ИС может выполнять с теми или иными объектами и их свойствами. Права доступа настраиваются в Панели управления сервера «Бизнес Семантика» (см. ниже, раздел «Конфигурирование сервера»). В некоторых коннекторах (например, в 1С) права доступа могут быть жестко определены в программном коде коннекторов.

III. Сервер «Бизнес Семантика»

В этом разделе мы детально рассмотрим архитектуру сервера «Бизнес Семантика» (далее просто «Сервер»), опишем процедуры его установки и настройки, форматы используемых данных, а также программный интерфейс сервера.

3.1. Программная архитектура сервера

Архитектура сервера достаточно проста, и включает следующие компоненты:

- Веб-сервис, реализованный на языке **PHP**. Доступен по ссылке **/services/BSS.php**, **wsdl** для него – по ссылке **/services/BSS.wsdl**. Методы этого сервиса детально рассматриваются ниже.
- Веб-страницы, предназначенные для конфигурирования сервера («Панель управления»). Доступны по ссылке с главной страницы (**/**). Программные модули, реализующие различные элементы Панели управления, находятся в папке **/config**.
- База данных **MySQL**, в которой хранятся настройки обмена данными, и права доступа.
- **RDFS**-схема – загружается из текстового файла в базу данных в процессе настройки сервера.

Рассмотрим структуру базы данных, в которой хранит свои настройки сервер «Бизнес Семантика»:

[*доступно в полной версии документации* ***]***

Описание формата **RDFS** можно найти здесь: **http://www.w3.org/TR/rdf-schema/**;
Описание формата **RDF**: **http://www.w3.org/RDF/**; **http://xmlhack.ru/texts/06/rdf-quickintro/rdf-quickintro.html**
Описание синтаксиса **Turtle**: **http://www.w3.org/TeamSubmission/turtle/**

3.2. Установка

Сервер устанавливается как обычное веб-приложение.

1. Для него необходимо создать отдельный виртуальный хост на веб-сервере (рекомендуем использовать **Apache** любой версии, и **PHP 5.2** или старше), и разместить файлы сервера в корневом каталоге виртуального хоста.
   Внимание: сервер не может функционировать в подпапке, или на порту, отличном от 80!
2. Нужно настроить **mod_rewrite**, указав следующие строки в конфигурационном файле **Apache (httpd.conf**, или отдельный файл, соответствующий данному хосту):
3. Необходимо создать базу данных **MySQL 5.1** или старше (обязательно использование **storage engine InnoDB!**). Кодировка по умолчанию для базы данных – **cp1251**. Если имя базы данных – **main**, то из консоли **MySQL** нужно выполнить следующую команду:
   CREATE DATABASE main DEFAULT CHARSET=cp1251;
   Затем нужно импортировать в базу данных дамп **main.sql** следующей командой (пусть **main** – имя базы):
   mysql –uroot –p –default-character-set=cp1251 main < main.sql
4. Нужно указать реквизиты доступа к БД в конфигурационном файле **/core/config.inc**
5. Необходимо защитить доступ к серверу, установив пароль **Apache** на доступ к корневой папке (при этом папка **/services** должна быть доступна без пароля!). Для этого нужно:
   - создать файл с паролями командой
     htpasswd.exe -cb .passwd root pass (здесь **root** – логин, **pass** – пароль)
     Скопировать полученный файл **.passwd** в корневую папку сервера.
   - В **httpd.conf**, или файле конфигурации виртуального хоста, написать следующее (предположим, что **c:\work\bss_server** – путь к каталогу, куда установлен сервер):

```
<Directory "c:/work/bss_server">
    Require valid-user
    AuthName "BSS Server"
    AuthType Basic
    AuthUserFile c:/work/bss_server/.passwd
</Directory>
<Directory "c:/work/bss_server/services">
    Satisfy Any
    Allow from all
</Directory>
```

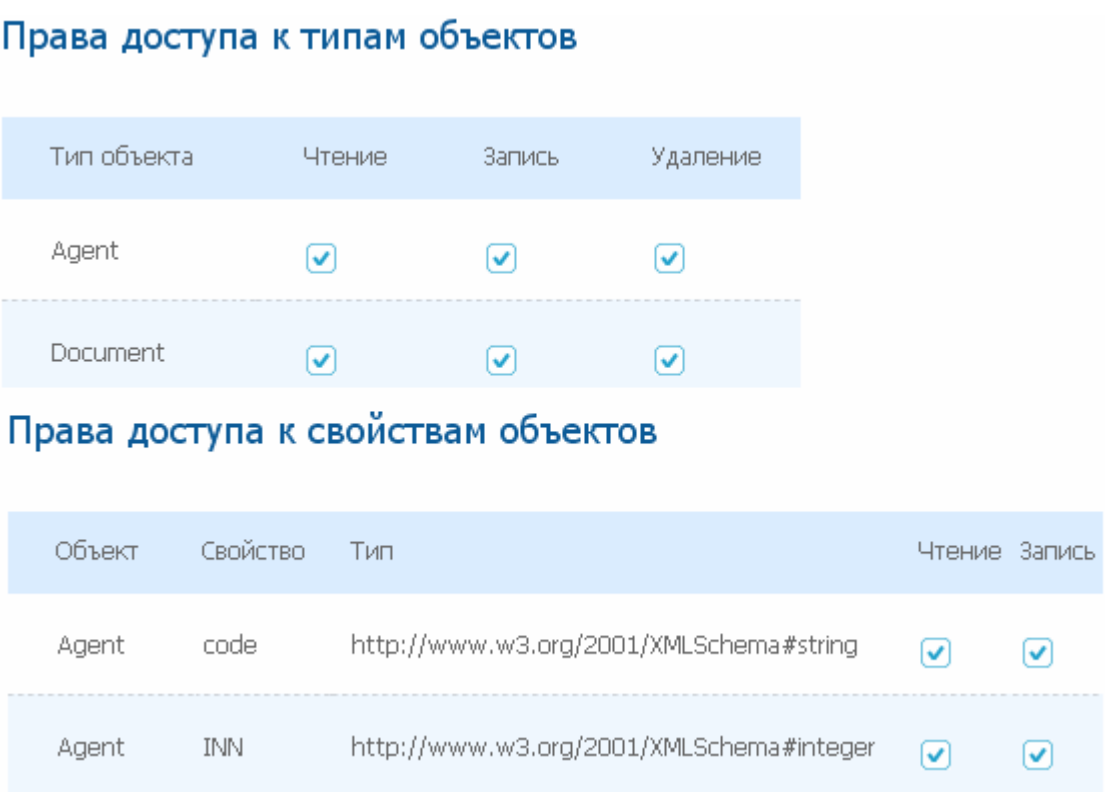
После этого нужно перезапустить службу **Apache.** Теперь сервер запущен, и вы можете войти в Панель управления, набрав в браузере имя виртуального хоста, и логин-пароль на доступ к нему, чтобы перейти к настройке сервера.

3.3. Конфигурирование сервера

После авторизации пользователь попадает на главную страницу веб-интерфейса сервера, где расположены две таблицы: список подключенных информационных систем, и настройки сервера. Первая таблица выглядит так:

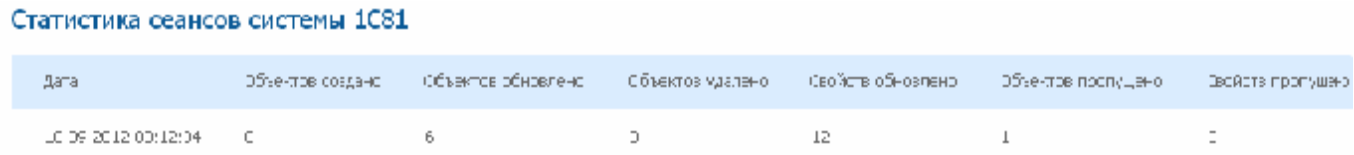
Ссылка «Добавить систему» открывает форму подключения новой информационной системы. В этой форме нужно ввести название системы, и токен, который будет использоваться для авторизации. Также можно ввести **IP-**адрес: если его указать, система сможет осуществлять подключение только с данного адреса. Если поле оставить пустым, то система сможет подключаться к серверу с любых адресов.

Если в форме «Подключенные информационные системы» нажать на ссылку на названии какой-либо системы, откроется форма редактирования ее свойств. Ниже нее расположены формы настройки прав доступа данной информационной системы:



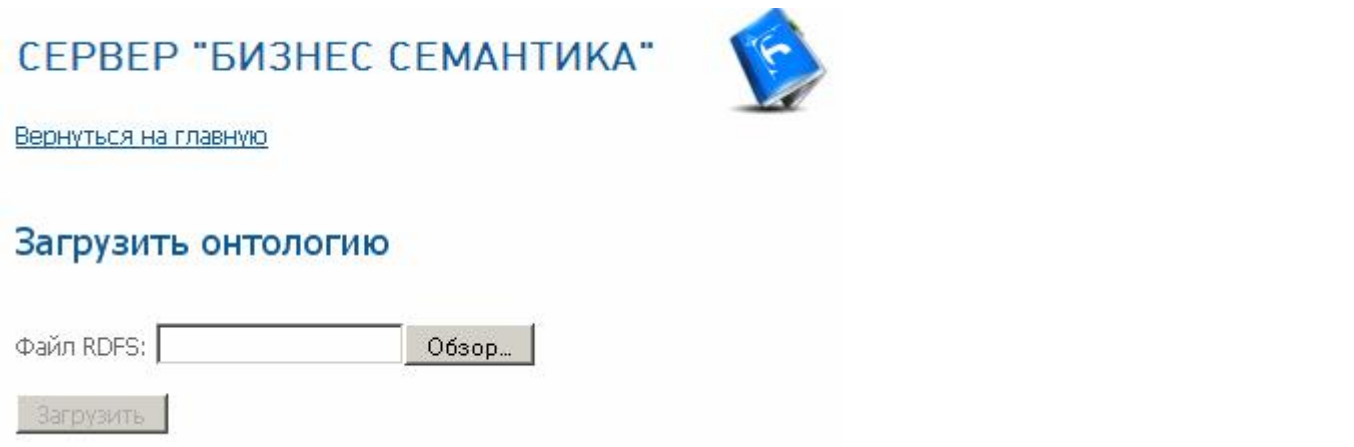
В первой таблице перечислены все виды объектов, определенные в онтологии. Для каждого типа объектов указывается при помощи переключателя, может ли данная информационная система получать информацию об объектах этого типа, создавать и изменять такие объекты, удалять их.
Во второй таблице, «Права доступа к свойствам объектов», перечислены все определенные в онтологии свойства тех объектов, к которым имеет доступ данная информационная система. Для каждого свойства указывается его наименование и тип. Здесь администратор сервера может указать, имеет ли данная информационная система право получать информацию о значении данного свойства объектов, и изменять ее.
Сделанные в этих формах настройки необходимо сохранить при помощи кнопки, расположенной внизу страницы.

Ссылка «Показать журнал» в списке информационных систем на главной странице веб-интерфейса сервера отображает список **10** последних событий (подключений) данной системы, а также статистику **10** последних сеансов. В таблице «Статистика сеансов» выводится информация о количестве объектов и свойств, затронутых в ходе сеанса:



Статистическую информацию серверу сообщают клиенты-коннекторы.

В форме «Настройки сервера» на главной странице веб-интерфейса отображается информация о загруженной онтологию. Администратор может перейти по ссылке «Загрузить онтологию», чтобы увидеть следующую форму:



В этой форме необходимо выбрать файл формата **RDFS**, содержащий онтологию, в которой будут выражены все данные, с которыми работают сервер и клиентские информационные системы. После загрузки онтологии можно будет перейти к описанным выше настройкам прав доступа информационных систем.

3.4. Программный интерфейс сервера

SOAP-интерфейс сервера БС реализует следующие методы:

[доступно в полной версии документации]

IV. Коннекторы «Бизнес Семантика»

4.1. Общий алгоритм работы коннектора

В этой главе рассматривается общий алгоритм построения коннектора «Бизнес Семантика». Все конкретные коннекторы, так или иначе, реализуют этот алгоритм.
Программный код коннектора состоит из двух относительно независимых блоков: блока импорта и блока экспорта. И тот, и другой имеют дело с подмножеством данных информационной системы – с теми видами информации, которыми, согласно схеме, должен происходить обмен с другими ИС. Наборы данных для импорта и экспорта могут не совпадать – например, **CRM**-система может экспортировать информацию о клиентах, а получать информацию о реализациях. Каждому из объектов, передаваемых серверу, присваивается уникальный код «Бизнес Семантика» (обычно состоит из английского наименования типа объекта, к которому добавлен его идентификатор в «родной» ИС, например, **«Sale19»**). Под этим кодом объект будет сохранен на сервере БС, а также распространен в другие ИС. Каждая ИС, получившая объект, обязана хранить и не изменять его уникальный код.

Задача блока экспорта состоит в отслеживании изменений, происходящих в тех данных, которыми нужно обмениваться с другими ИС. Отслеживание может происходить при помощи разных инструментальных средств: «Подписки на события» в **1С**, триггеров на таблицах – в веб-коннекторе. Обнаружив изменение в данных (или удаление какого-либо объекта), блок экспорта выполняет следующие операции:

- Помещает данный объект в очередь на отправку (этот шаг не обязателен – в **1**С, например, отправка происходит в синхронном режиме);
- Находит все объекты, зависимые от данного, информацию о которых тоже необходимо передать серверу (например, при передаче объекта «Реализация» может передаваться также информация о клиенте, которому был продан товар);
    - Для этого по схеме данных определяются свойства объекта, которые являются ссылками на другие объекты;
    - Определяются сами объекты, на которые ссылается передаваемый объект;
    - Если у какого-то из этих объектов нет кода БС – ему присваивается код, и объект также отправляется серверу.
- Формирует текстовый файл в синтаксисе **Turtle**, содержащий информацию о всех измененных объектах;
- Авторизуется на сервере БС;
- Передает объекты серверу БС.

Блок импорта устроен сложнее. Он представляет собой периодически вызываемую процедуру (это может достигаться разными инструментальными средствами – «Регламентными заданиями» в **1**С, планировщиком задач в случае веб-коннектора). Процедура обращается к серверу для получения изменившихся данных, получает их и обрабатывает. Для этого она выполняет следующие действия:

- Авторизуется на сервере БС;
- Получает информацию об изменившихся или удаленных объектах;
- Начинает транзакцию (может быть отменена, если при обработке возникнет ошибка). Наличие механизма транзакций критически важно для правильной работы механизма, поскольку в противном случае может возникнуть ситуация частичного импорта пришедших данных – до момента возникновения ошибки.
- Очень важно, чтобы при создании или изменении объектов в ходе импорта не срабатывали обработчики событий, которые экспортируют данные об изменившихся объектах – в этом случае сервер получит обратно все те же данные, которые только что отправил. Способ подавления обработчиков зависит от платформы, на которой реализуется коннектор. Подавление обработчиков нужно отменить в конце обработки, независимо от того, закончилась ли она успехом, или возникновением исключения. Также необходимо заблокировать изменение тех хранилищ данных, куда будет вносить изменения процедура импорта, чтобы пользователь не мог внести в них изменений во время обработки.
- Проходит по списку полученных от сервера объектов, находя по коду БС каждый из них, или создавая пустой объект при его отсутствии;
- Одновременно происходит удаление объектов, об удалении которых сообщил сервер;
- Может быть предусмотрена специальная процедура – определенный пользователем обработчик – которая будет вызываться на этой стадии, для выполнения каких-то

дополнительных операций с объектом (на данный момент возможность определения таких процедур предусмотрена только в веб-коннекторе);

- На второй стадии обработки полученных данных система проходит по массиву полученных от сервера объектов и свойств, присваивая свойства объектам (то есть, на этом шаге происходит наполнение объектов конкретной информацией: например, «Дата продажи **Sale19** равна **10.09.2012»**).
    - Если какое-либо свойство содержит ссылку на другой объект – система находит объект, получает его идентификатор в «родной» системе, и сохраняет этот идентификатор в соответствующем свойстве обрабатываемого объекта;
    - Если объект, на который ссылается свойство обрабатываемого объекта, не найден – коннектор запрашивает у сервера информацию об этом объекте, а также об обрабатываемом (чтобы получить ее повторно, после того, как появится нужный объект);
    - Для некоторых свойств объектов схемой могут быть предусмотрены специальные программные обработчики, определенные пользователем – они вызываются на этой стадии обработки;
- Сообщает серверу статистику обработки;
- Завершает транзакцию;
- В случае ошибки на любом предшествующем этапе – отменяет транзакцию, и возвращает серверу информацию об ошибке.

Для работы коннектору необходима схема, которая содержит описание взаимосвязей объектов, а также отображение **(mapping)** логических свойств объектов, которыми ИС обменивается с сервером, физическим свойствам (например, полям БД в соответствующей таблице). Далеко не всегда можно установить точное свойство между логическим объектом (например, «Продажа») и какой-либо одной таблицей БД или программным объектом (например, **«sales»** или Документы.РеализацияТоваровИУслуг), или между свойством логического объекта и полем в БД (например, ИНН и КПП могут представлять собой два разных свойства логического объекта, но храниться в одном поле БД). Для решения таких ситуаций предусмотрен механизм определяемых пользователем обработчиков – реализованный в универсальном виде, как в веб-коннекторе, или встраиваемый непосредственно в код обработки, как в 1С.
Очень важно корректно реализовать в коннекторе транзакции и обработку ошибок (лучше – при помощи механизма обработки исключений).

4.2. COM-объект BSclient

COM-объект предназначен для использования в составе коннекторов на платформе **Windows.** Он обеспечивает функции загрузки и парсинга схемы, а также парсинга данных в формате **Turtle. COM-**объект реализует следующие методы:

Метод	Параметры	Возвращаемое значение	Описание
LoadSchema	**BSTR* filename**		Загружает из файла **RDFS** схему
Parse	**BSTR* ttl**		Разбирает строку в формате **Turtle**, загружая данные в свои внутренние структуры. Для доступа к данным используются следующие методы.
getNextProperty	**BSTR* prop** **BSTR* value** **LONG* ref**	**LONG* result** (**true**, если возвращено очередное свойство, **false**, если свойства закончились)	Вызывается в цикле, используется для получения следующего свойства (из списка свойств, загруженных методом **Parse**). Возвращает значения в переменные, являющиеся параметрами: **prop** – имя свойства, **value** – его значение, **ref** – является ли ссылкой на объект (**1** если да)
getNextObject	**BSTR* name** **BSTR* type**	**LONG* result** (**true**, если	Вызывается в цикле, используется для получения следующего

		возвращен очередной объект, **false**, если объекты закончились)	объекта (из списка объектов, загруженных методом **Parse**). Возвращает значения в переменные, являющиеся параметрами: **name** – код БС объекта, **type** – его тип
getPropertyType	**BSTR* obj_type** **BSTR* prop**	**LONG* result**	Возвращает тип свойства **prop** объектов **obj_type**. Тип является константой: **1** = целое число, **2** = строка, **3** = число с плавающей точкой, **4** = дата, **99** = ссылка на объект
getTargetType	**BSTR* type** **BSTR* prop**	**BSTR* result**	Возвращает тип объекта, на который ссылается свойство **prop** объектов типа **type**
findObject	**BSTR* type** **BSTR* name**	**LONG* result**	Возвращает индекс объекта с кодом БС равным **name**, типом **type** (**-1**, если не найден)
getObjectByNumber	**LONG number** **BSTR* name** **BSTR* type**	**LONG* result**	Возвращает информацию об объекте по его индексу. **Number** – индекс, **name** – код БС объекта, **type** – его тип
cleanPrefix	**BSTR* stmt**		Удаляет префикс из типа объекта или свойства, возвращает значение в ту же строку
getProperty	**BSTR* obj_id** **BSTR* prop_name** **BSTR* prop_value** **LONG* prop_is_obj**		Возвращает значение свойства **prop_name** объекта с идентификатором **obj_id**. Значение записывается в переменную **prop_value**, а в **prop_is_obj** записывается **1**, если свойство указывает на другой объект (в этом случае **prop_value** будет содержать его код БС)

Приведем несколько упрощенный пример использования **COM**-объекта **BSclient** из обработок **1С**.

```
// Создаем COM-объект:
Сервис=Новый COMObject("BSClient.BSClient");

// Загружаем RDFS-схему
Сервис.LoadSchema("c:\work\bss_client\main.rdfs");

// Разбираем данные в формате Turtle, содержащиеся в строке Стр
Сервис.Parse(Стр);

// Проходим циклом по всем объектам, получая имя и тип каждого объекта в одноименные переменные
Пока Сервис.getNextObject(имя, тип)

    // Очищаем название типа объекта от префикса
    Сервис.cleanPrefix(тип);

    // Получаем значение свойства "date" объекта с кодом БС, равным «имя».
    // Значение возвращается в переменную Дата
    Сервис.getProperty(имя,"date",Дата,ссылка_объект);

    // Пробегаем по всем свойствам текущего объекта
```

```
    Пока Сервис.getNextProperty(проп, значение, ссылка_объект)

        // Получаем тип объекта, на который ссылается свойство «проп» объектов «тип»
        ТипОбъекта=Сервис.getTargetType(тип, проп);

        // Получаем тип свойства «Свойство» объекта «ТипОбъекта»
        ТипСвойства=Сервис.getPropertyType(ТипОбъекта, Свойство);

        // Получаем внутренний индекс объекта с кодом «кодБС», типом «ТипОбъекта»
        инд=Сервис.findObject(ТипОбъекта,кодБС);

        // По внутреннему индексу получаем имя и тип объекта
        Сервис.getObjectByNumber(инд, имя_объекта, тип_объекта);
    КонецЦикла;
КонецЦикла;
```

4.3. Коннекторы для 1С

В настоящей версии продукта предусмотрены коннекторы для платформы **1**С версий **7.7**, **8.1**, **8.2.** Рассмотрим особенности коннектора для **8.1**, затем остановимся на специфике коннектора для **7.7.**

Программный код коннектора необходимо включить в конфигурацию **1**С. Детально этот процесс описан в инструкции на коннектор. Отметим следующие основные особенности:

- Для экспорта данных используется механизм оповещения о событиях. Устанавливается «подписка» на сохранение и удаление объектов – методы ПередЗаписьюДляЭкспорта и ПередУдалениемДляЭкспорта, размещенные в общем модуле «ЭкспортСемантика». Модуль, в зависимости от типа измененного объекта, вызывает одну из функций (например, «ОтправитьРеализацию»), размещенных в обработке «ЭкспортСемантика». Эта функция добавляет данные в буфер отправки. Затем, при необходимости, она вызывает функции для отправки объектов, зависящих от данного (например, клиента, которому совершена продажа). По завершении последовательности вызовов функций Отправить[Объект], метод ПередЗаписьюДляЭкспорта отправляет сформированный буфер на сервер БизнесСемантика.
- Для блокировки подписки на события, на время обработки импорта, а также на время присвоения кодов БС в ходе экспорта, используется переменная сеанса ПараметрыСеанса.ОбработкаБС. Реализовать механизм полной блокировки справочника или журнала документов на время, когда установлен данный флаг, в **1**С невозможно, поэтому в будущем необходима более корректная реализация блокировки подписки на события: вместо установки одного общего флага, в некий буфер должны записываться коды обрабатываемых объектов, и блокировка подписки должна распространяться только на них.
- Для взаимодействия с сервером БС используется **COM**-объект **MSSOAP**:

  ```
  АдресСервиса="http://bs_server/services/BSS.wsdl";
  Сервис = Новый COMObject("MSSOAP.SoapClient");
  Сервис.MSSoapInit(АдресСервиса);
  Result = Сервис.login("1C81","password");
  и т.д.
  ```

- Для периодического опроса сервера с целью получения входящих данных, используется механизм регламентных заданий. Регламентное задание с установленной периодичностью вызывает процедуру ИмпортДанных, находящуюся в общем модуле ЭкспортСемантика (служебные функции импорта определены в обработке ИмпортСемантика).
- Поскольку средства работы со строками в **1**С оставляют желать много лучшего, для парсинга схем и данных в формате **TTL** используется **COM**-объект **BSclient.** Обобщенный пример его использования, как раз в контексте **1**С, см. в предыдущем разделе. При помощи методов этого объекта коннектор пробегает по всем полученным от сервера БС объектам и их свойствам. Обработка выполняется в три прохода:

  - На первом проходе коннектор запоминает объекты и их свойства в массив, находит или создает объекты.
  - На втором проходе обрабатываются специальные данные – в тестовом примере это объект **SoldItem**, представляющий информацию об одной строке из табличной части документа «Реализация». Особая обработка для него необходима потому, что этот информационный объект не имеет прямого аналога – объекта 1С (соответствует строке табличной части документа).
  - На третьем проходе происходит присвоение значений, при этом, в частности, происходит поиск объектов-адресатов для свойств, которые хранят ссылки на другие объекты.
- С целью сокращения количества одинакового кода, информация о более или менее стандартных модулях 1С, куда вносится информация, объединена в массив «НаборСправочников». При помощи объектов под названием «Проп» и «Соответствие» устанавливается соответствие свойств, описанных в схеме, свойствам этих объектов. Это позволяет (правда, путем использования нелюбимого 1С метода «Выполнить») избавиться от многократного повторения кода типа «Клиент = Справочники.Контрагенты.Создать()», «Клиент.Название=значение».

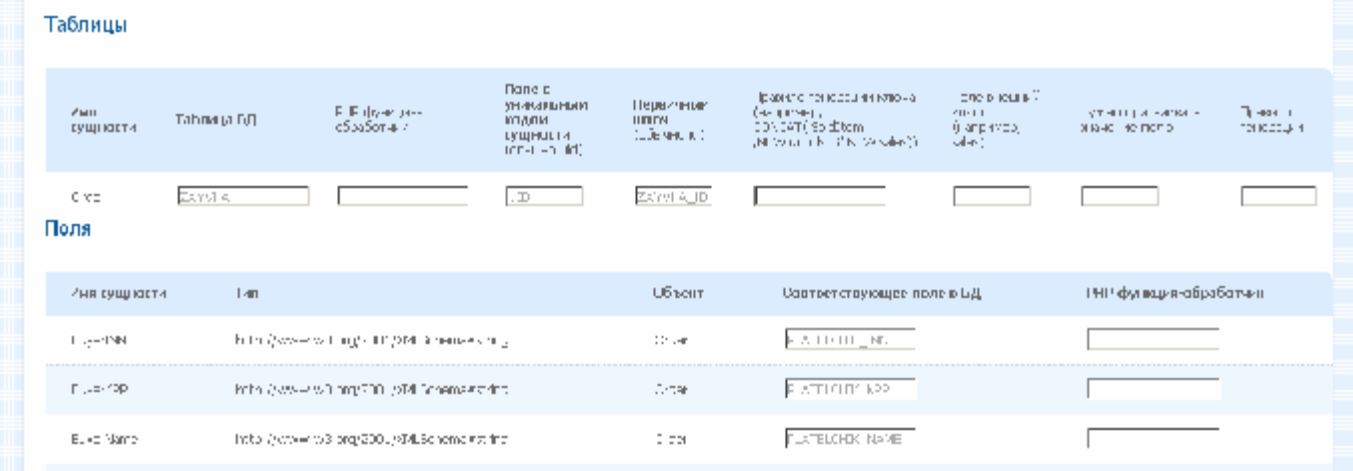
Теперь опишем особенности коннектора для платформы **7.7**. Поскольку в ней нет механизма подписки на события, необходимо добавить свой код в методы глПриУдаленииЭлемента, а также в модули формы элемента, в процедуру «ПриЗаписи()», для всех обрабатываемых справочников. Для журналов документов соответствующий код можно добавить в процедуру ОбработкаПроведения.
В остальном код экспорта данных идентичен коду, используемому в 1С **8.1**.

4.4. Веб-коннектор (PHP / MySQL+Oracle)

Механизм работы веб-коннектора основан на отслеживании изменений в базе данных при помощи триггеров. В панели управления (которая описывается ниже) настраивается сопоставление типов объектов, определенных в схеме, таблицам базы данных (когда это возможно); веб-коннектор создает на этих таблицах триггеры, отслеживающие добавление, изменение и удаление записей в таблицах. Сведения о таких данных помещаются в слежуюные таблицы-очереди, которые веб-коннектор создает в базе. При каждом очередном сеансе обмена информацией с сервером происходит передача ему сообщений об этих записях. Сеансы инициируются вызовом по расписанию (средствами операционной системы сервера) скрипта **services_client.php**, входящего в состав веб-коннектора.

Веб-коннектор хранит в базе данных собственную служебную информацию (настройки). Для этого он может использовать ту же самую базу, в которой хранятся синхронизируемые данные, или отдельную конфигурационную базу. Реквизиты доступа к обеим базам хранятся в файле **/core/defines.ext**. Специальная константа, **$store_queue_inside_client** (определенная в том же файле), определяет, в какой из двух баз будут храниться таблицы с очередью записей для отправки. Эти таблицы называются **bs_update_queue** и **bs_delete_queue**. Все остальные настройки делаются в веб-интерфейсе коннектора.

Веб-коннектор имеет собственную панель настроек, работающую практически идентично панели настроек сервера, описанной выше. Главная специфика работы веб-коннектора связана с сопоставлением сущностей и свойств схемы таблицам и полям базы данных. Для этого предназначена специальная страница веб-интерфейса панели настроек:



Таблицы базы данных должны иметь поле для хранения уникального идентификатора каждого объекта **(URI).** Обычно это поле называется **uid**; поле нужно создать в базе данных.
Также каждая таблица может иметь первичный ключ, который является локальным уникальным идентификатором записей (обычно это поле **id**); в случае, если его нет (например, описывается таблица-связка), необходимо задать способ генерации уникального локального идентификатора. Такой идентификатор может представлять собой произвольную строку, которую генерирует **SQL**-выражение (например, **CONCAT('SoldItem',NEW.sales)**); выражение должно однозначно идентифицировать каждую запись в таблице. Данное выражение будет использоваться в триггере, срабатывающем при создании/изменении записей в таблице. Если невозможно и это, нужно создать поле, в котором будет формироваться уникальный локальный идентификатор по произвольному правилу, не зависящему от свойств записи. В поле «Правило генерации» нужно вписать одно из предопределенных значений, описывающих функцию-генератор ключа (в данный момент доступен только один вариант – **"md5"**), и указать название поля, которому будут присваиваться сгенерированные таким образом значения.

Значение в колонке «Поле-внешний ключ» нужно указывать, если данная таблица является подчиненной по отношении к другой (например, табличная часть продажи – **sales_goods** – является подчиненной по отношению к таблице продаж, **sales**). Установка этого значения влияет на рекурсивную обработку записей при их удалении (и только при удалении!).

Ниже, в таблице «Поля», надо указать для каждого свойства сущности из схемы данных, какому полю БД оно будет соответствовать. Если установить однозначное соответствие невозможно, нужно указать название **PHP** функции-обработчика.

Функции-обработчики бывают двух видов: уровня сущностей (типов объектов), и уровня полей. Обработчики уровня сущностей используются в том случае, если нельзя установить однозначное соответствие таблице БД для этого типа объектов, или необходимо выполнять какую-либо дополнительную обработку при операциях с этими объектами.

Функции-обработчики уровня сущностей определяются в файле **services_client.php**, как методы класса **ParseHandler**. Обработчик получает два параметра: массив с информацией о данном конкретном объекте, и массив с описанием типа объекта. Первый массив содержит элементы:

- **Objid – URI** объекта;
- **Objtype –** определение типа объекта;
- **Data –** вложенный массив, каждый элемент которого содержит значения **prop** и **value**. Они хранят, соответственно, определения и значения свойств данного объекта.
- **Uid –** присвоенный **URI** объекта, если обработчик вызывается при его создании.

Метод должен вернуть **URI** объекта. Если наряду с обработчиком определено и соответствие таблице БД, то к моменту вызова обработчика он будет уже присвоен, и методу надо будет только вернуть элемент **uid** из описываемого массива. Наряду с этим, обработчик может выполнить любые требуемые операции с сущностью.
Второй массив содержит элементы, соответствующие настройкам, сделанным в панели настроек веб-клиента. Поле **dtable** хранит название таблицы БД (если задано), поле **codefield –**

название поля для хранения **URI** объекта, поле **reffield –** название поля, хранящего локальный идентификатор объекта, и т.д.

Функции-обработчики уровня полей тоже определяются в файле **services_client.php**, как методы класса **ParseHandler.** Обычно, для каждого поля нужно объявить отдельную функцию-обработчик. Структура типичной функции-обработчика выглядит так:

```
function name($field,$data,$direction="write") {
  if($direction=="read") {
   return $data["name"]);
   }
  elseif($direction=="write")
   return $data["value"];
  }
```

Входные параметры: **$field –** имя поля, **$direction –** направление, в котором происходит передача данных **(“read”** или **“write”), $data –** массив с значениями, подлежащими обработке. При операции **read** он содержит ассоциативный массив со всеми свойствами записи в БД **(**результат работы запроса **SELECT * FROM …),** а при операции **write –** ассоциативный массив с переданными с сервера значениями, имена элементов в котором соответствуют именам свойств в схеме.

Внутри функция разделена на два блока: срабатывающий при чтении значения **(**его передаче серверу**),** и при записи значения в БД. Если установить однозначное соответствия свойства полю нельзя, то в случае записи значения код должен сам выполнить необходимые **SQL-**запросы. Если соответствие установить можно, но значение нужно преобразовать перед записью, то в настройках наряду с названием функции-обработчика указывается название поля; в этом случае функция-обработчик может просто вернуть значение, которое ему необходимо присвоить.
При операции **read** может возникнуть необходимость объединить несколько значений в одно. Для этого обработчик каждого конкретного поля должен запомнить данные из своего поля в переменные-члены класса **ParseHandler.** Перед завершением формирования информационного пакета, соответствующего этой записи, движок вызовет метод **FinishStatement,** который вернет итоговую строку **(**если в процессе обработки данного объекта был вызван хотя бы один обработчик уровня поля**).** Например, так:

```
function FinishStatement($stat,$direction="write") {
  if($direction=="write")
   return "`name`=TRIM('".$this->fn." ".$this->ln."')";
  }
```

Параметр **$stat –** строка с текущим сформированным информационным пакетом.

Обработчик уровня сущностей вызывается только в случае чтения **(**передачи информации на сервер**),** перед началом обработки очередной записи. Он получает в качестве параметров имя поля-идентификатора в соответствующей локальной таблице БД, и его значение. Если обработчик вернет **false,** обработка прекратится, и информация о данном объекте не будет передана серверу. Если он вернет строку – она должна представлять собой полностью сформированный информационный пакет с данными об этой записи.