Тренинг **Разработка мобильного приложения на 1С с нуля – за 7 вечеров!**

Модуль 3. Использование планов обмена для мобильных баз данных

Бесплатные материалы проекта:
http://курсы-по-1с.рф/free/

Электронная почта: support@Kursy-po-1C.ru
Телефоны: **8-800 1000-613**, +7(495) 7777-096

Оглавление

Третий модуль (день). Введение

В предыдущем дне тренинга, мы научились переносить данные с клиента на сервер. При этом важно помнить, что скорость работы платформы на мобильном устройстве не идет ни в какое сравнение со стационарным компьютером.

Нами было перенесено некоторое количество элементов справочника и документов. Но как осуществить такой перенос для базы, где более 100 000 единиц товара, более 500 000 характеристик, и около миллиона штрихкодов? И главное – как это синхронизировать?

Очевидно, что переносить десятки мегабайт информации каждый раз при обмене, особенно через мобильный интернет – нелегкая задача. Если даже договориться, что мобильный агент (сотрудник) перед началом рабочего дня – приходит в офис и через Wi-Fi загружает все данные, то в любом случае – загрузка всех данных на мобильное устройство будет довольно длительной операцией. Поэтому необходимо выгружать только новую или изменившуюся информацию.

Некоторые фиксируют изменения при помощи регистра сведений, ставя обработчик *ПриЗаписи* и заполняя регистр сведений. Другие ставят жесткие отборы при выгрузке, и, допустим из 100 000 товаров выгрузятся только 2 000, но КАЖДЫЙ раз.

Но как только дело доходит до штрихкодов, да и вообще до регистров сведений, не подчиненных регистратору, возникает вопрос – как однозначно идентифицировать изменение записи в регистре?

Для справочников, документов и т.д. есть, как минимум, *UID*, по которому можно однозначно определить объект. А если был изменен штрихкод, то возникнет необходимость сохранять старую версию этой записи. Для того чтобы не создавать новую запись на клиенте, а изменить существующую. Именно это и позволяют делать планы обмена.

Но, с планами обмена тоже не все так просто. Если делать типовую выгрузку изменений плана обмена, то он выгрузит все реквизиты объекта. И при загрузке будет ожидать, что эти реквизиты есть у клиента.

Другими словами, типовой обмен данными возможен только между двумя идентичными (по структуре) базами данных. Нам же такой вариант не подходит. Т.к. хранить на телефоне все объекты стационарной конфигурации бессмысленно.

Возьмем, к примеру, типовую конфигурацию «Управление торговлей 10.3» и посчитаем количество реквизитов у справочника «Номенклатура». Реквизитов там около 30! И это не считая табличной части. Из них на мобильном устройстве нам понадобятся около трех, может чуть больше. Также, у этих реквизитов может отличаться тип и тогда, мы будем вынуждены перенести все справочники и т.д., что удастся сделать не всегда, так как в мобильной платформе доступны не все объекты.

Что же делать? Те, кто сталкивался с аналогичной проблемой ранее, могли использовать конфигурацию «Конвертация данных», основная цель которой – создание правил обмена между двумя разными базами данных. Но в мобильной платформе нет некоторых объектов, которые она использует (например, нет запросов). И соответственно, обработки по загрузке и выгрузке данных работать не будут.

Хотя можно, конечно, переписать ее таким образом, чтобы она работала с объектами, доступными на мобильной платформе, или же, чтобы она могла работать только с одной стороны. Т.е. по этим правилам формировалась не только выгрузка данных, но и загрузка. В таком случае, клиент бы считал, что он обменивается с идентичной конфигурацией.

Планы обмена

Далее, всю синхронизацию мы будем настраивать при помощи планов обмена. Посмотрим, что это такое.

В справке 1С сказано:

План обмена используются для реализации механизмов обмена данными. Планы обмена содержат информацию об узлах, которые могут участвовать в обмене данными, определяют состав данных, которыми будет производиться обмен, и указывают, следует ли задействовать механизм распределенной информационной базы при обмене.

В одном прикладном решении может существовать несколько планов обмена, каждый из которых может описывать свой порядок обмена данными. Например, если выполняется обмен данными с удаленными складами и удаленными офисами, то, скорее всего, будет существовать два плана обмена (один для обмена со складами, другой - для офисов), поскольку состав данных, которыми производится обмен со складами, будет значительно "уже", чем состав данных, предназначенных для обмена с офисами.

Если в двух словах, то план обмена можно представить в виде справочника со встроенным регистром сведений. Где элемент справочника – это узел базы данных (в нашем случае – мобильное устройство).

А во встроенном регистре находится информация о том, какие объекты необходимо выгрузить в конкретный узел. Этот регистр нельзя просмотреть, так как мы можем просмотреть регистры сведений и накоплений, и данные туда записываются и удаляются при помощи специальных методов.

Вся синхронизация базируется на основании номеров сообщений – номер принятого сообщения и номер отправленного.

Посмотрим, как это выглядит в действии.

Принцип работы плана обмена

Допустим, нам необходимо сделать синхронизацию между мобильным клиентом и сервером. В предыдущем дне тренинга мы это и делали, но выгружали все данные. Сейчас необходимо выгружать только измененные. Подумаем, как это сделать.

Идеальным был бы вариант, в котором в момент синхронизации мы могли бы обратится к некоему регистру и забрать из него только те объекты, которые нам нужно выгрузить на мобильное устройство. Почему это удобно?

Во-первых, это будет работать намного быстрее любых, даже фиксированных, отборов (так как при фиксированном отборе, необходимо каждый раз выполнять запросы для получения данных из разных объектов – справочников, документов и т.д.).

Во-вторых, намного удобней, так как мы работаем только с одним объектом, и знаем, что нужно выгрузить все, что находится в нем.

В-третьих, сервер нагружается только в тот момент, когда он записывает данные в регистр (если мы хотим данные выгружать выборочно). А при получении данных для выгрузки, на сервере будет минимальная загрузка.

В-четвертых, мы можем редактировать этот регистр, т.е. если мы что-то случайно туда выгрузили, мы можем это удалить.

В-пятых, всегда можно просмотреть текущую очередь выгрузки и отследить, если там что-то осталось после полного обмена (в обе стороны), то где-то была ошибка на клиенте и ее нужно исправить.

И т.д.

Итак, мы видим, что использовать регистр сведений очень удобно.

Рассмотрим его состав. Первые два поля, которые приходят в голову сразу – узел и объект. Ведь мы должны понимать, что и кому выгружать.

Теперь на базе этого попробуем сделать обмен. Мы внесли некий узел, который однозначно определяет мобильное устройство и записали в регистр два объекта для выгрузки.

Делаем первый вызов сервера. В запросе получаем все данные из регистра и передаем на устройство. Ничего не меняя, делаем второй вызов. Увидим, что получены те же данные, т.к. мы их не удалили.

Начнем все с начала, повторим тоже, что и раньше - делаем первый вызов, получаем данные, удаляем их из регистра, и при загрузке данных – на телефоне у нас появилась ошибка. Делаем еще один вызов на сервер, а в регистре очереди - пусто. Мы же данные удалили.

Таким образом мы подобрались к квитированию.

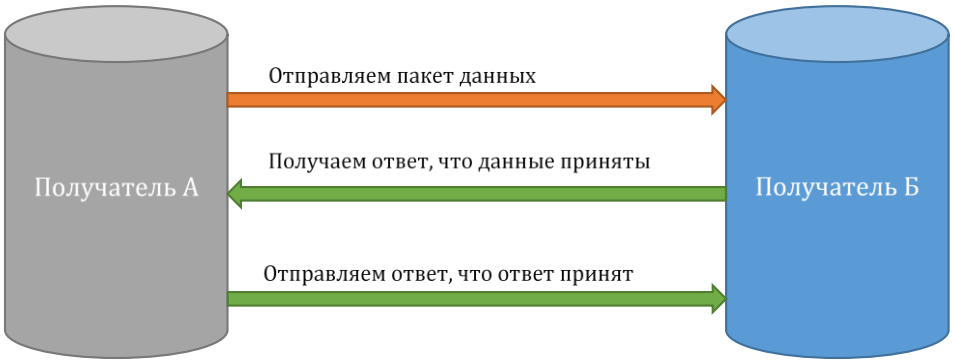
Квитирование (тройное рукопожатие)

Во всем мире активно используется механизм «тройного рукопожатия». Рассмотрим, как он работает.

Допустим, нам необходимо отправить срочное письмо другу и для нас очень важно знать, что он его получил. Ответ от друга будет являться подтверждением, что письмо дошло. Но и другу, в свою очередь, важно знать, что мы получили его ответ.

Мы будем постоянно отправлять письмо, раз за разом, пока не получим ответ. Как только мы его получили, но не ответили другу – он будет постоянно отправлять ответ нам. Так будет продолжаться, пока он не получит ответ от нас.

Схематически это выглядит так:



На что тут нужно обратить внимание, мы отправляем пакет данных, а получаем ответ. Т.е. мы можем послать пакет в несколько гигабайт, а получить ответ – «ОК». Это очень важно.

Вернемся к нашей задаче.

Прежде чем удалить данные нам необходимо получить подтверждение от клиента о том, что он принял эти данные.

Еще раз пройдемся по циклу:

1. Вносим данные в регистр
2. Запрашиваем данные с сервера
3. Получаем данные
4. Загружаем в телефон
5. Запрашиваем сервер еще раз и говорим, что данные приняли
6. Удаляем данные на сервере

Теперь посмотрим, что будет, если добавить между четвертым и пятым пунктом еще один – «Поставим новый объект в очередь для выгрузки». В нашем случае, сервер его удалит вместе со старыми данными.

Таким образом, мы пришли к тому, что в регистр необходимо добавить номер сообщения.

К примеру, всем записям изначально присваивается нулевой номер. Далее, при каждой выгрузке для всех записей формируется другой номер – на 1 больше предыдущего. Зафиксируем это, чтобы знать с какого номера начинать отсчет, ведь из регистра записи будут удалены. Пусть он хранится в реквизите нашего узла.

Создадим числовой реквизит элемента, и назовем его «Исходящий номер сообщения». По сути он будет показывать сколько раз была сделана выгрузка.

Теперь, если повторить ту же самую ситуацию, то, когда клиент заберет ответ от сервера (пункт 3), и на сервере запишут новый объект, он будет записан с номером 1.

Т.е. клиент забрал записи с номером 0, и теперь, когда он сообщит об этом серверу, сервер удалит все записи с номером сообщений 0, но не тронет записи с номером сообщений 1.

Если при загрузке сообщений 0 возникла ошибка, то клиент запросит все данные с этого регистра. В том числе, там будут данные с номером 1.

В этом случае, клиенту всегда нужно сообщать номер выгрузки.

Но каждый раз перезаписывая справочник и перепроводя документы необходимо делать точечное обращение к реквизиту узла (а узлов может быть много), всего лишь для того, чтобы получить номер сообщения. Это похоже на пустую трату ресурсов.

Тогда может быть имеет смысл сделать все наоборот? Т.е. все новые объекты или изменения вносить с номером 0, и если такой объект уже был в выгрузке с другим номером, то просто менять номер на 0. А в момент выгрузки, у всех существующих записей увеличивать номер на единицу. Так мы меньше нагружаем сервер.

Но в этом случае, необходимо знать последний номер сообщения, которое успешно принял клиент, и увеличивать его на один. Когда клиент сообщит серверу о номере успешной загрузки, сервер удалит все записи с более старшим номером. Так как теоретически, выгрузку мы можем делать сотни раз, а клиент может ответить успехом только один раз, причем какой именно – не известно.

Так выглядит работа плана обмена, если бы мы хотели сделать самописный обмен, на основании каких-либо своих регистров. С каждым разом, делая его все правильней и правильней, мы бы реализовали свой план обмена.

Однако у плана обмена есть возможность регистрировать записи регистра сведений. Что для нас является очень важным. Т.е. у себя в регистре он содержит информацию о том, как выглядела запись регистра до изменения, и как она выглядит после.

Варианты обмена ЦБ с мобильным клиентом

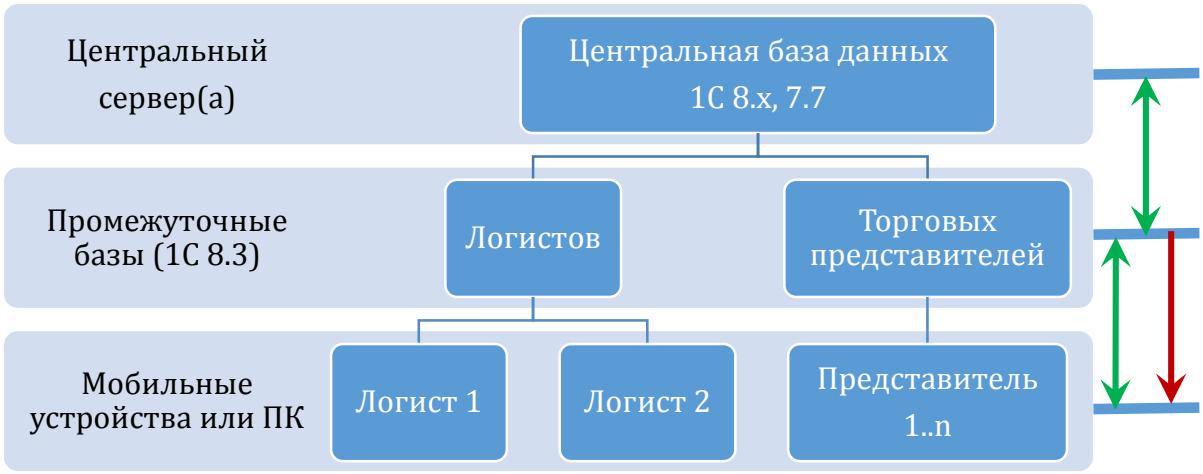
После того, как мы описали ряд проблем и вспомнили (или узнали), что такое планы обмена, нужно подумать о том, как можно решить нашу задачу, которая заключается в синхронизации данных. Для этого необходимо ответить на один вопрос:

Как можно настроить синхронизацию (обратите внимание, не просто обмен данными) между клиентами и центральной базой (ЦБ)?

Планы обмена + КД + Промежуточная база

Известно, что планы обмена работают только между двумя идентичными базами или при помощи правил конвертации данных (хотя существует и другой вариант – обращаться напрямую к записям регистрации плана обмена, но об этом позже). А мобильная конфигурация программируется на стационарной платформе. Тогда мы можем воспользоваться этой пустой базой данных в качестве промежуточной базы данных (ПБ)

В итоге у нас выйдет следующая схема:



На схеме зелеными стрелочками показан обмен данными, а красной – обмен конфигурацией. Таким образом, МП получает данные от ПБ, и от нее же получает конфигурацию. А ПБ обменивается данными с ЦБ.

Рассмотрим преимущества данной архитектуры:

- Одна и та же база для мобильного приложения и для стационарного;
- Гарантированно не будет возможности получить запрещенные данные из ЦБ в мобильное приложение;
- Нет необходимости конфигурировать ЦБ для реализации обменов, так как при обмене между ЦБ и ПБ – можно использовать типовой обмен данными при помощи «Конвертации данных»;

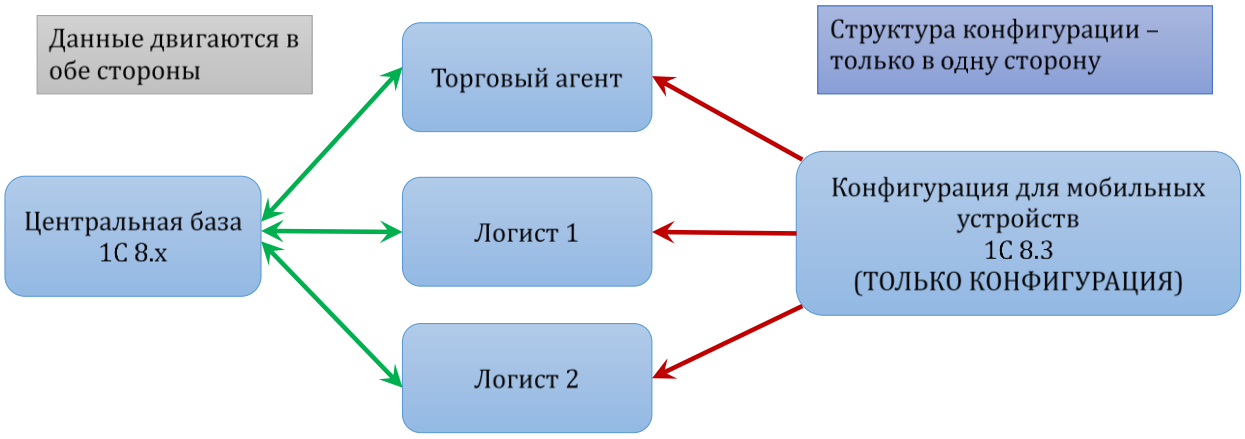
- Так как метаданные в МП и ПБ будут идентичными, то для обмена между МП и БП можно использовать полный план обмена;

Из недостатков:

- Использование ПБ – это еще одно звено, а чем меньше звеньев, тем стабильней;

Архитектура ЦБ(8.х) ↔ МП (самописный обмен)

Конечно же есть вариант использования самописного обмена. Его мы реализовали в рамках предыдущего дня тренинга.



Здесь зеленными стрелочками выделен обмен данными, а красными – обмен конфигурацией.

Что особенного происходит в данном случае? У нас есть центральная база, в которой настроены обмены (соответственно мы предаем данные из приложения в ЦБ и обратно), но одновременно у нас есть отдельная база, в которой мы пишем конфигурацию для мобильных телефонов.

Таким образом, ЦБ не должна располагаться именно на платформе 8.3, она может располагаться на любой платформе, просто в *планах обмена* (ну или в другом механизме, выбранном для переноса данных) нужно учитывать особенности разных версий платформ.

Минусы такого подхода очевидны – необходимо постоянно переделывать конфигурацию, нужно создавать новые *планы обменов* и/или *web-сервисы* и т.д.

Ее стоит забывать также, что база должна иметь выход в интернет, причем как первая, так и вторая (та, что с конфигурацией). Если же используется вариант фиксированной конфигурации, скомпилированной до *apk*-файла, то правую часть из схемы можно убрать. Но придется добавить другую – это сервис обновления приложения на мобильной платформе.

Почему ЦБ должна иметь доступ в интернет, почему ЦБ на платформе 8.х? Все очень просто – самый удобный способ обмена, это обмен при помощи *веб-сервисов*. А они появились с 8.х и требуют выход базы в интернет.

Гибрид

Выбор варианта обмена зависит от поставленных целей. Также есть возможность выбрать гибрид.

К примеру, у нас есть справочник товара, контрагентов и прочего, из этого практически ничего никогда не меняется и не обновляется. Но вот заказы – это динамическая информация. Тогда в промежуточную базу можно выгрузить всю статическую информацию, и подтягивать ее на мобильное приложение, а заказы передавать на прямую в ЦБ.

Таким образом получается ряд преимуществ:

- Не нужно грузить центральную базу при обновлении справочников, например, при первой авторизации мобильного телефона или в случае, когда нужно добавить какой-то новый реквизит.
- В промежуточную базу выгружаете штрихкода, цены и даже остатки можно, а с мобильника все это забирать.
- В ЦБ выгружаются только заказы, и, как показывает практика, после выгрузки редактировать их нельзя. Т.е. с ЦБ, в лучшем случае, можно получать только список статусов заказов (принят, отклонен, собран и т.д.).

Практически все будет передаваться между ЦБ и ПБ при помощи КД, и только заказы будут самописные. Таким образом, когда в мобильной платформе появятся запросы, можно очень быстро избавиться от ПБ и все перенести полностью на мобильный телефон, при малейшем редактировании ЦБ.

Какой вариант синхронизации выбрать?

Это достаточно сложный вопрос. Каждый решает вопрос исходя из своих способностей, возможностей и поставленных задач.

К примеру, если выбрать вариант самописного обмена, то необходимо понимать, что при любой доработке обмена придётся обновлять ЦБ, что весьма проблематично, если база работает 24/7. В этом случае может подойти второй или третий вариант.

При самописной конфигурации вряд ли удастся получить преимущество от использования конвертации данных. А если конфигурация базовая – вариантов не много. Поэтому каждый должен выбирать путь сам.

Реализация обмена на основе планов обмена

Будем считать, что мы выбрали архитектуру обмена «Планы обмена + КД + Промежуточная база». Т.е. когда мы из некой центральной базы данных настроили обмен в промежуточную, например, на основе конвертации данных. И наша текущая задача – настроить обмен между промежуточной базой данных и мобильным клиентом. Так как конфигурацию мы берем из промежуточной базы, то конфигурации мобильного клиента и промежуточной базы – идентичны.

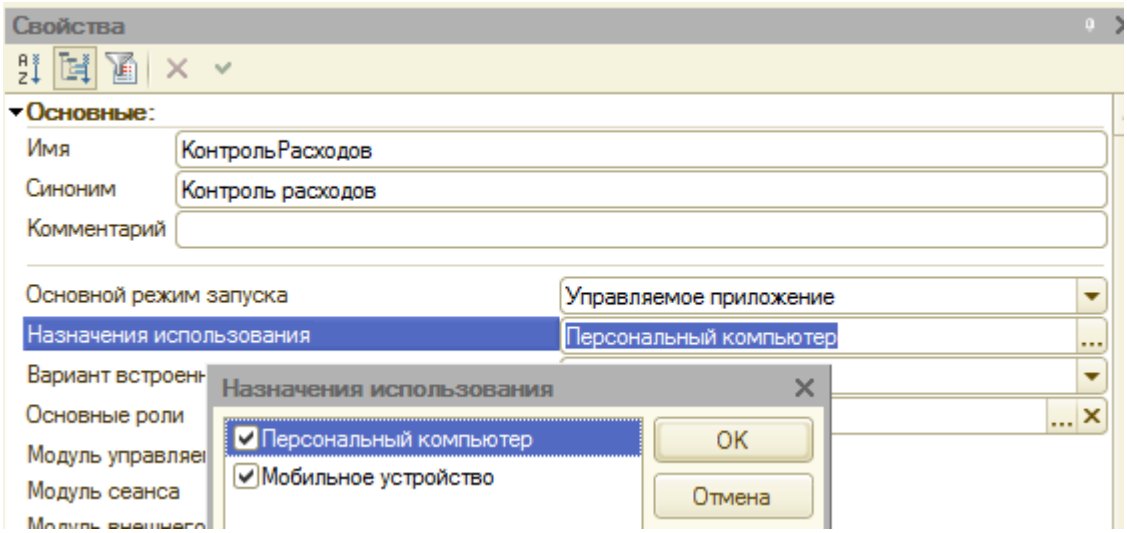
Создадим новую конфигурацию, и в нее добавим справочник «СтатьиРасхода». Сделаем документ «СписаниеДенежныхСредств» с реквизитами «СтатьяРасхода» и «СуммаРасхода».

Добавим регистр сведений «РекомендуемыеСуммыРасходаСтатей», в котором будет измерение «СтатьяРасхода», ресурс «Сумма», и он будет периодическим, в пределах дня.

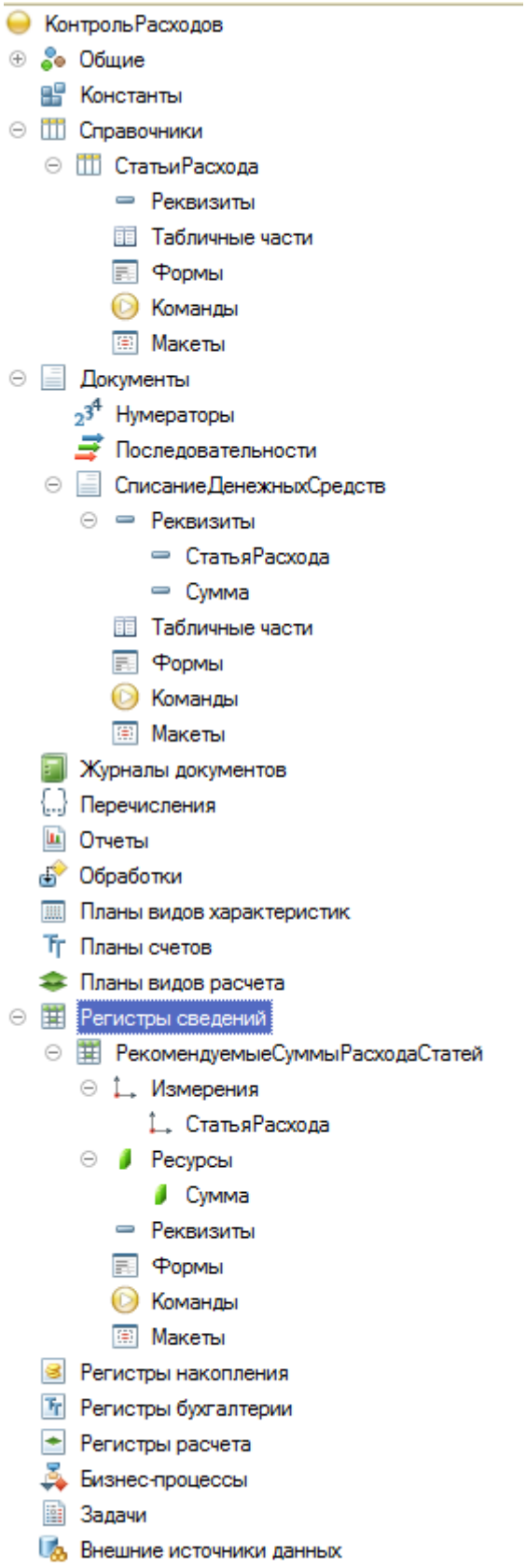
Напишем конфигурацию, в которой будем вести расходы. А изначально, при создании документа списания денежных средств, будет подставляться сумма из регистра сведений. Т.е. если стоимость проезда стоит 10 копеек в автобусе, то мы сэкономим время на внесение этой суммы каждый раз, когда мы делаем расход. Т.е. мы создали документ расхода – выбрали статью, и программа автоматом подставила сумму по умолчанию.

Конфигурацию назовем «Контроль расходов». Обратите внимание, назначение использования ставим и мобильное устройство, и стационарный компьютер. Так как будет

использоваться конвертация данных, а если выбрать только мобильное устройство, то обработка конвертации работать не будет.

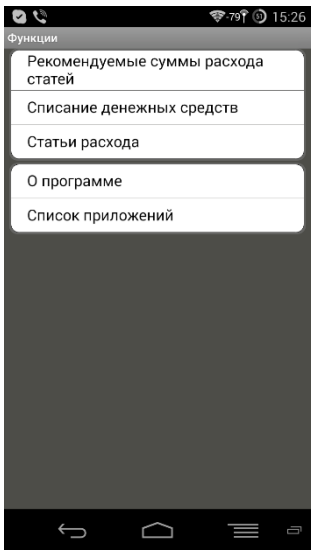


Древо конфигурации будет выглядеть вот так:

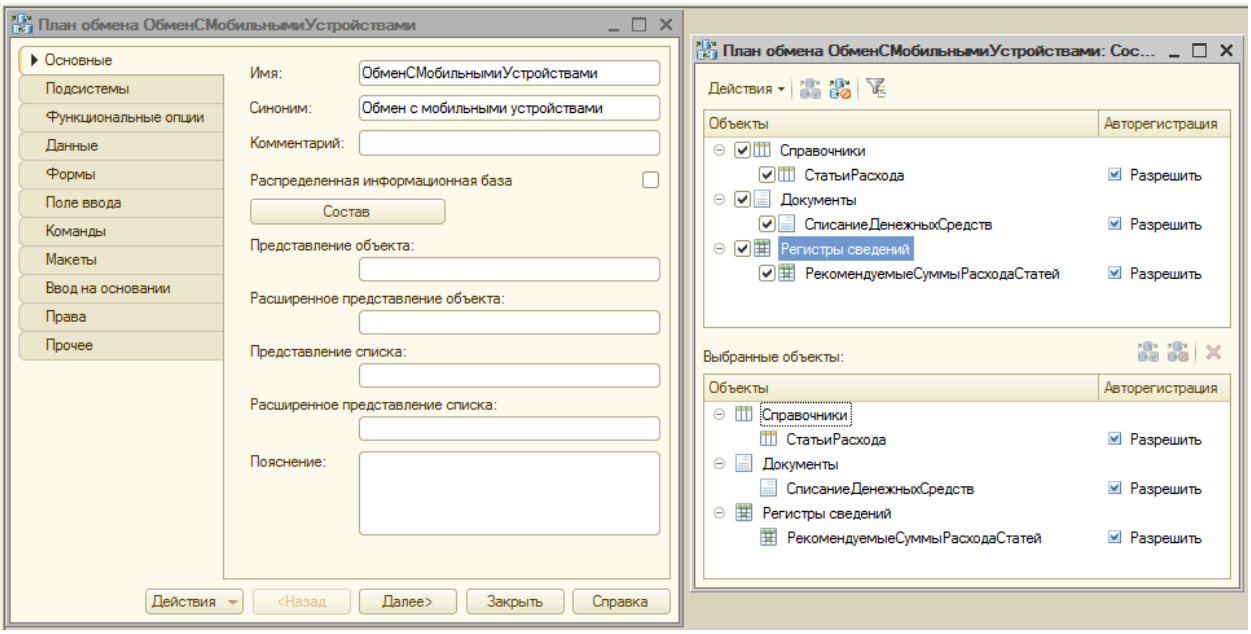


Обратите внимание, что все объекты конфигурации доступны.

Теперь публикуем конфигурацию и подключаем на мобильное устройство. Убедимся, что все правильно сделали. На мобильном устройстве должны увидеть вот такое меню:



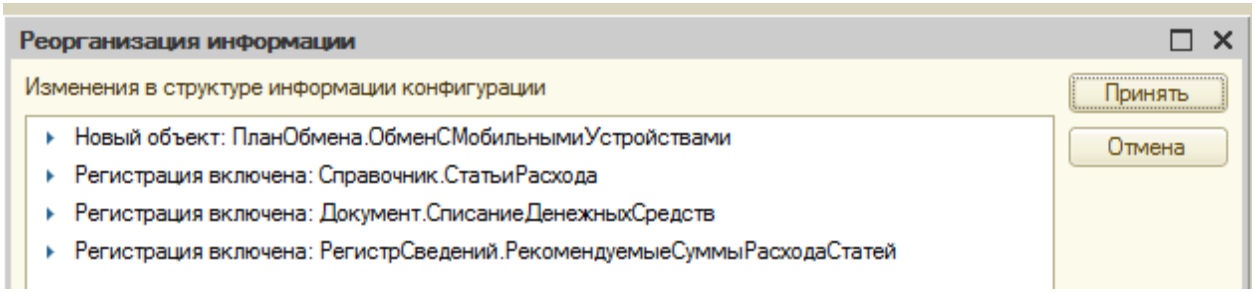
Так как мы не указали рабочий стол. Вроде все отлично, конфигурация подхватилась. Теперь нужно создать план обмена.



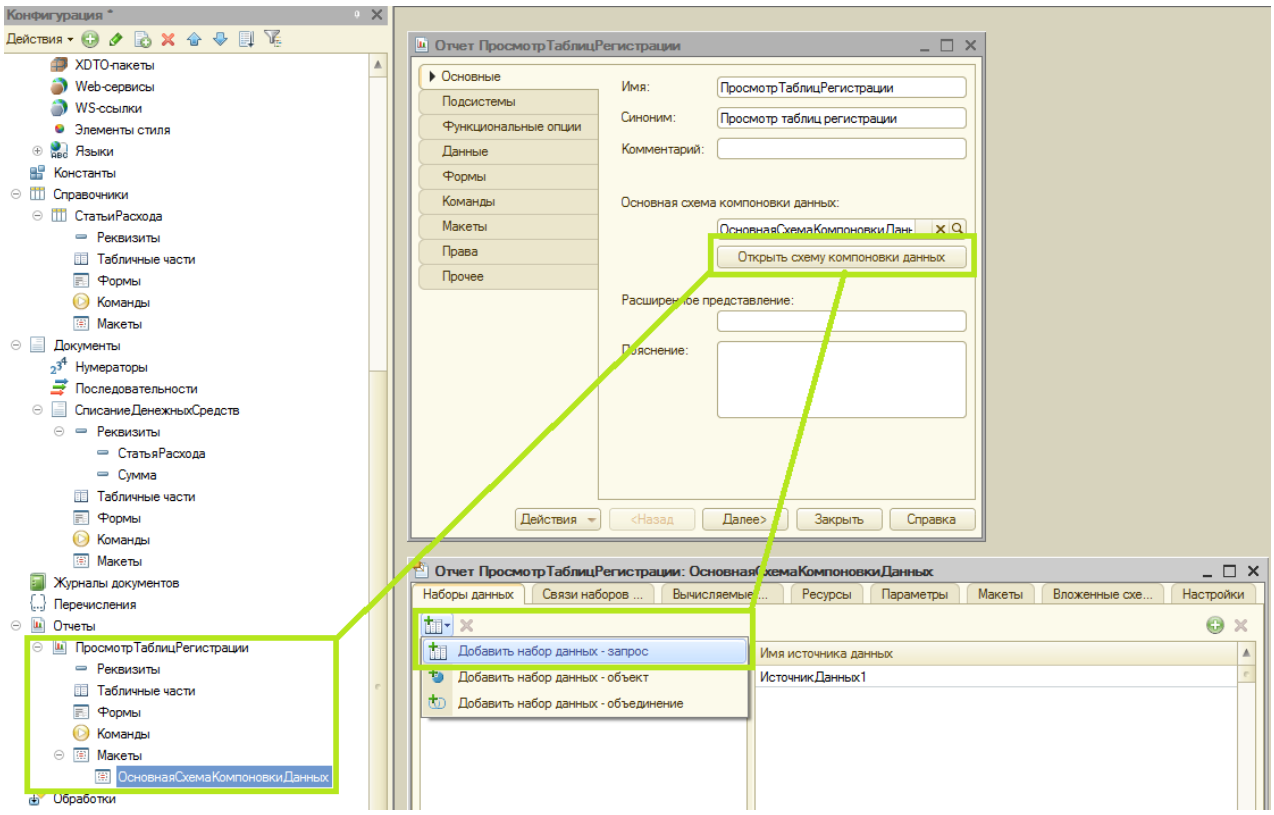
Создаем план обмена «ОбменСМобильнымиУстройствами» и выбираем состав. Т.е. какие объекты должны переноситься. Указываем галочками все объекты.

Колонка «Авторегистрация» говорит платформе, что данные объекты необходимо ставить в очередь по умолчанию (если авторегистрация разрешена) или игнорировать (если авторегистрация запрещена). В случае запрета, мы вынуждены сами контролировать очередь выгрузки, но об этом позже.

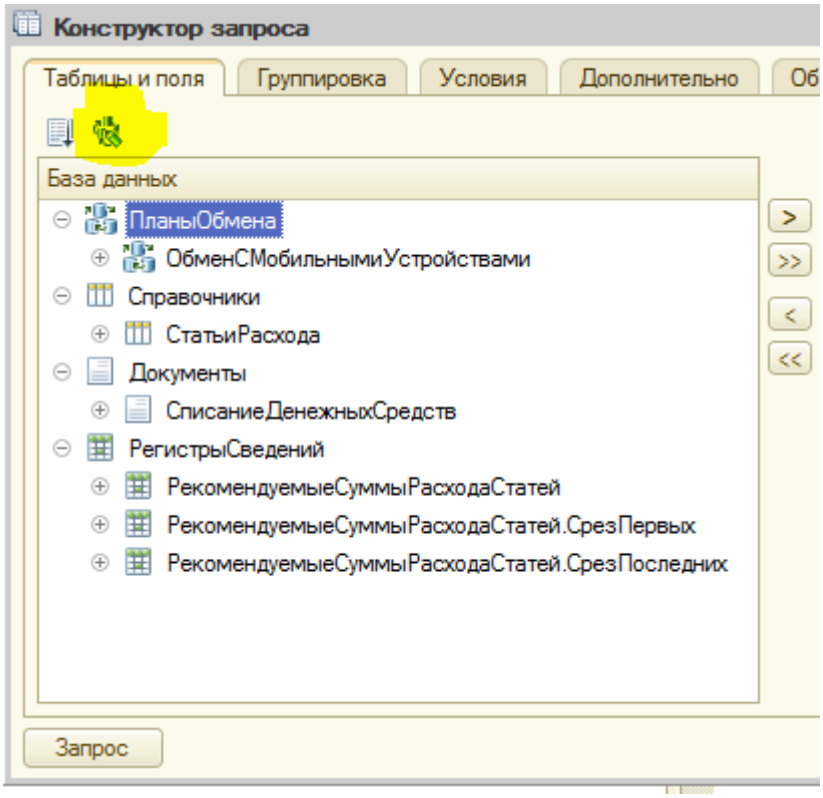
Обновим конфигурацию, при обновлении мы должны увидеть вот такое окно:



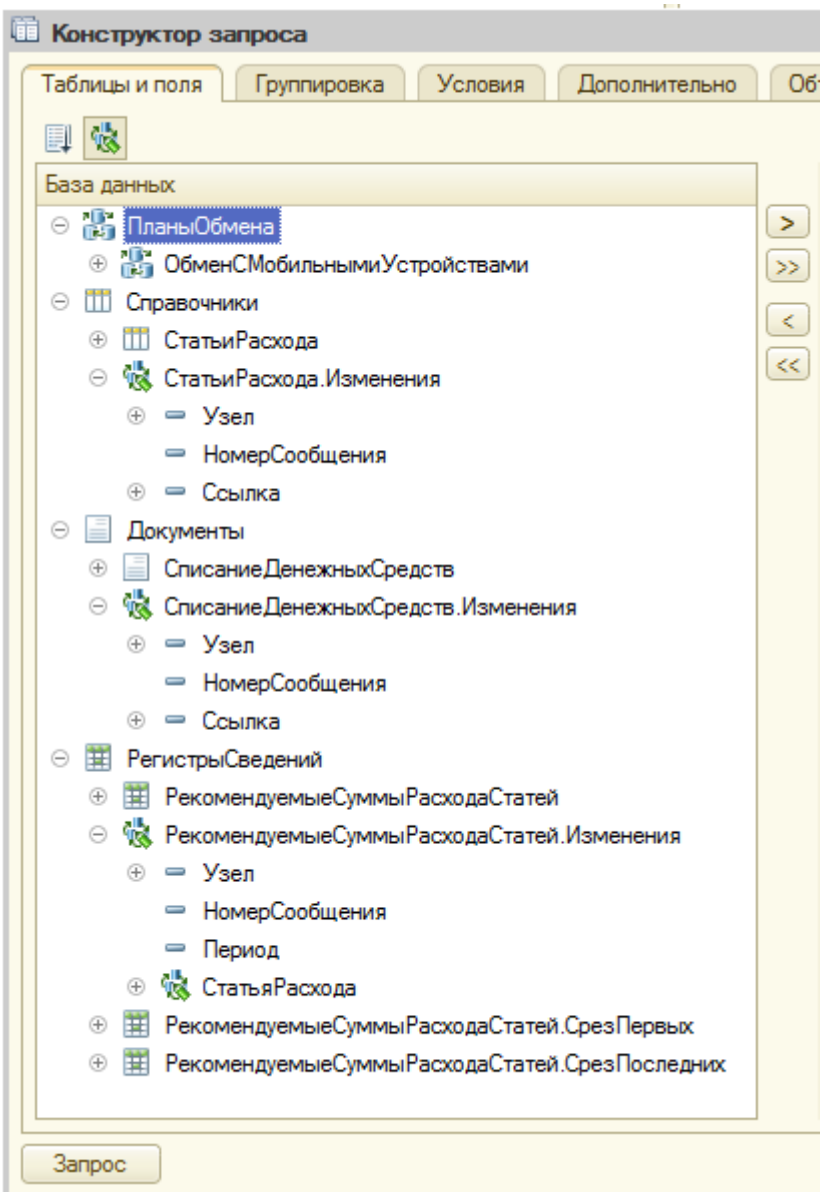
Оно говорит о том, что регистрация включена, и будут созданы специальные таблицы регистрации изменений для каждого объекта. Для их просмотра создадим отчет «ПросмотрТаблицРегистрации»:



Нажимаем «Конструктор запросов» и нам доступны только следующие таблицы:



Однако, как только нажмем на кнопку, выделенную желтым, появятся служебные таблицы регистрации изменений:

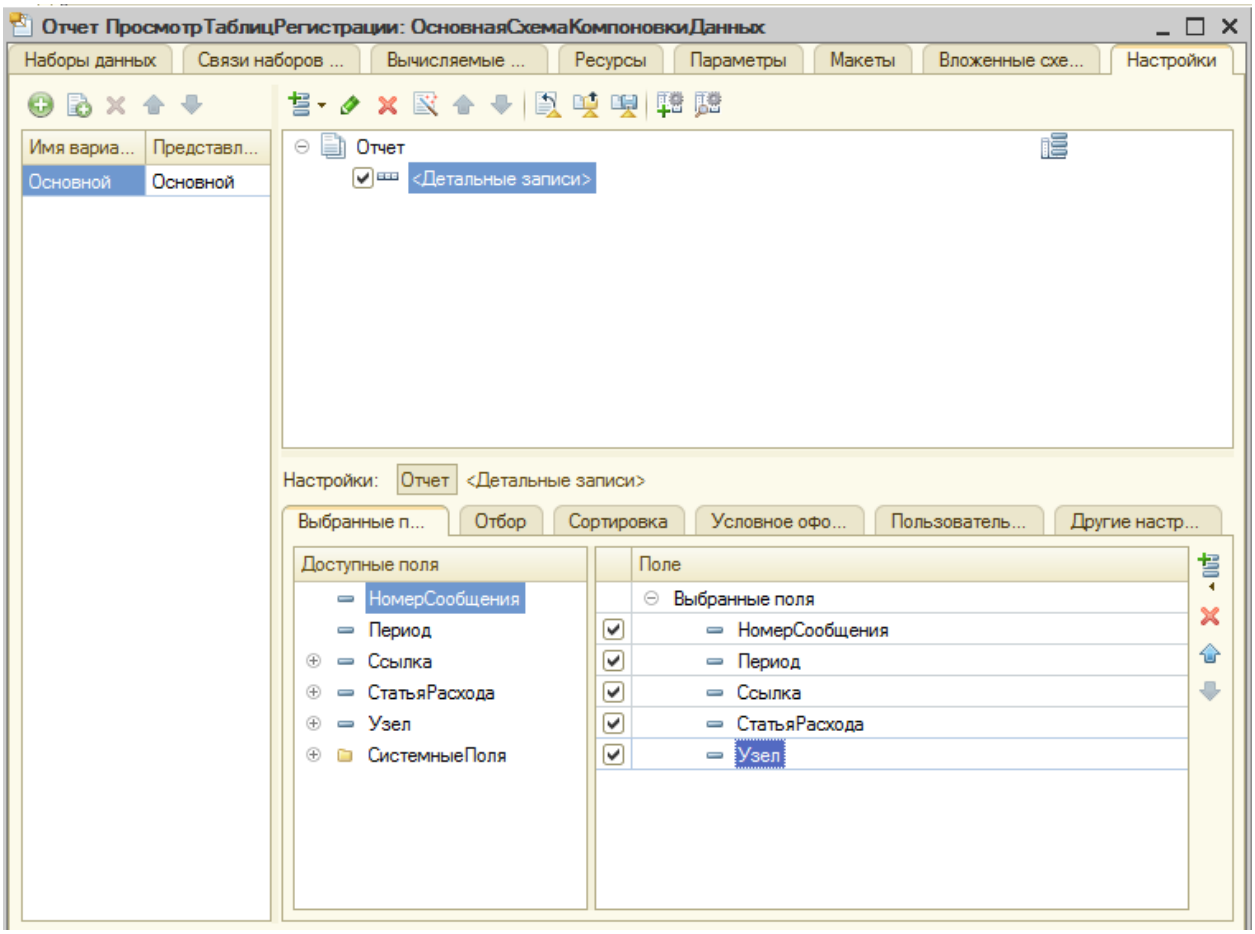


Сделаем вот такой запрос:

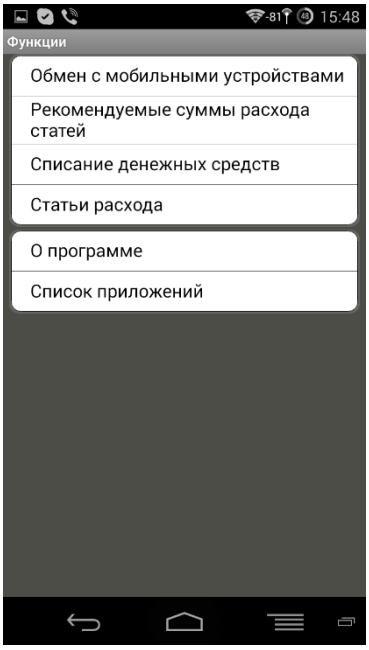
```
"ВЫБРАТЬ
   СтатьиРасходаИзменения.Узел,
   СтатьиРасходаИзменения.НомерСообщения,
   СтатьиРасходаИзменения.Ссылка,
   NULL КАК Период,
   NULL КАК СтатьяРасхода
ИЗ
   Справочник.СтатьиРасхода.Изменения КАК СтатьиРасходаИзменения

ОБЪЕДИНИТЬ ВСЕ
```

```
ВЫБРАТЬ
   СписаниеДенежныхСредствИзменения.Узел,
   СписаниеДенежныхСредствИзменения.НомерСообщения,
   СписаниеДенежныхСредствИзменения.Ссылка,
   NULL,
   NULL
ИЗ
   Документ.СписаниеДенежныхСредств.Изменения КАК СписаниеДенежныхСредствИзменения

ОБЪЕДИНИТЬ ВСЕ

ВЫБРАТЬ
   РекомендуемыеСуммыРасходаСтатейИзменения.Узел,
   РекомендуемыеСуммыРасходаСтатейИзменения.НомерСообщения,
   NULL,
   РекомендуемыеСуммыРасходаСтатейИзменения.Период,
   РекомендуемыеСуммыРасходаСтатейИзменения.СтатьяРасхода
ИЗ
   РегистрСведений.РекомендуемыеСуммыРасходаСтатей.Изменения КАК
РекомендуемыеСуммыРасходаСтатейИзменения"
```

Перейдем на последнюю вкладку, добавим пустую группировку и укажем какие поля необходимо выводить:



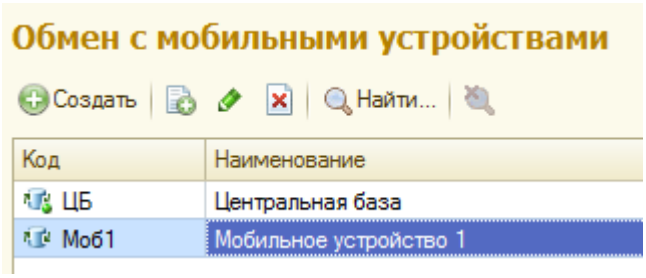
Обновим конфигурацию, и обновим ее на мобильном устройстве.



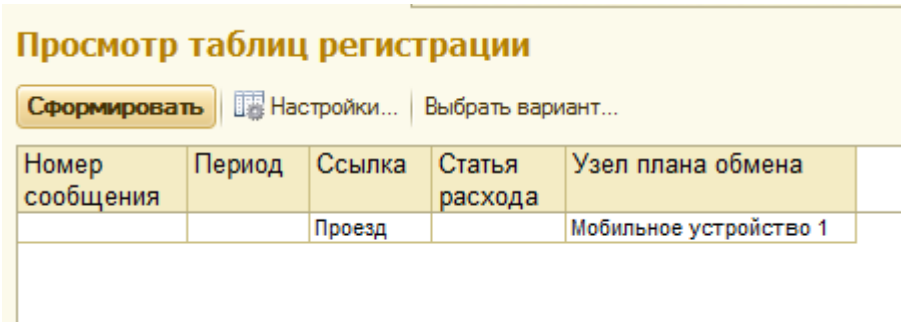
Обратите внимание, отчет не появится на телефоне, так как этот объект не доступен на мобильной платформе.

Запускаем стационарную платформу и построим этот отчет. Отчет сформируется пустым. Создадим новую статью – «Проезд». Сформируем отчет еще раз – тоже пусто, хотя должно было бы быть зафиксировано изменение. Почему отчет пустой? Потому что у нас не создано ни одного узла. Т.е. программа не понимает для кого регистрировать изменения. Создадим новый узел.

Изначально, в списке планов обмена, присутствует пустой узел, этот узел является текущей базой данных. Т.е. не центральной, а именно текущей. Заполним этот узел и создадим новый:



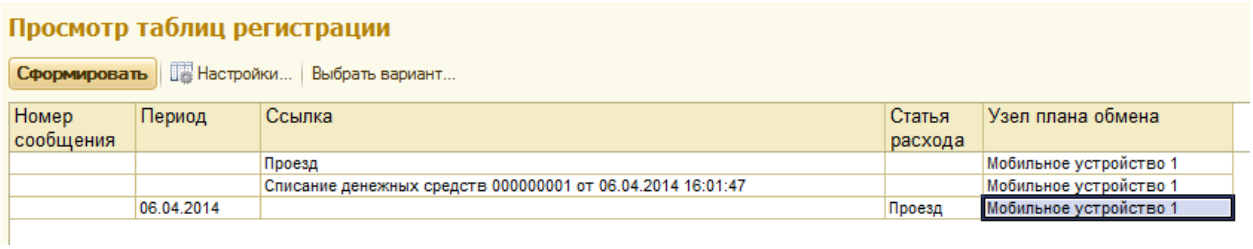
Построим отчет еще раз. И отчет снова пустой. Важно, при создании нового узла 1С не регистрирует изменения для него по умолчанию. Перезапишем нашу статью и снова сформируем отчет:



Теперь в отчете появилась запись. Обратите внимание, колонки *Период* и *Статья расхода* – это измерения регистра сведений, у которых стоит галочка «Основной отбор».

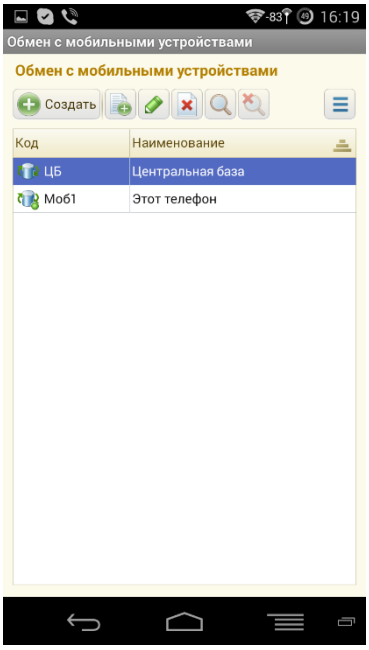
Т.е. набор этих измерений гарантирует уникальность записи, ведь у регистра нет уникального идентификатора, как у документа или справочника.

Теперь создадим документ и запись в регистре. Сформируем отчет:



Видим, что документ добавился как ссылка, а регистр сведений – как набор ключевых полей.

Теперь создадим на мобильном устройстве узлы обмена:



Здесь главным узлом будет мобильный телефон. Имя можно присвоить любое. Главное, чтобы код совпадал.

Обмен данными через веб-сервис

Настроим обмен, допустим, через веб-сервисы. Создадим веб-сервис «ОбменДанными» с операцией «Синхронизация» и одним входящим параметром «Данные». Но, тип этого параметра будет не строка, как мы делали ранее, а *ХранилищеЗначения*. И возвращать веб-сервис будет тоже хранилище.

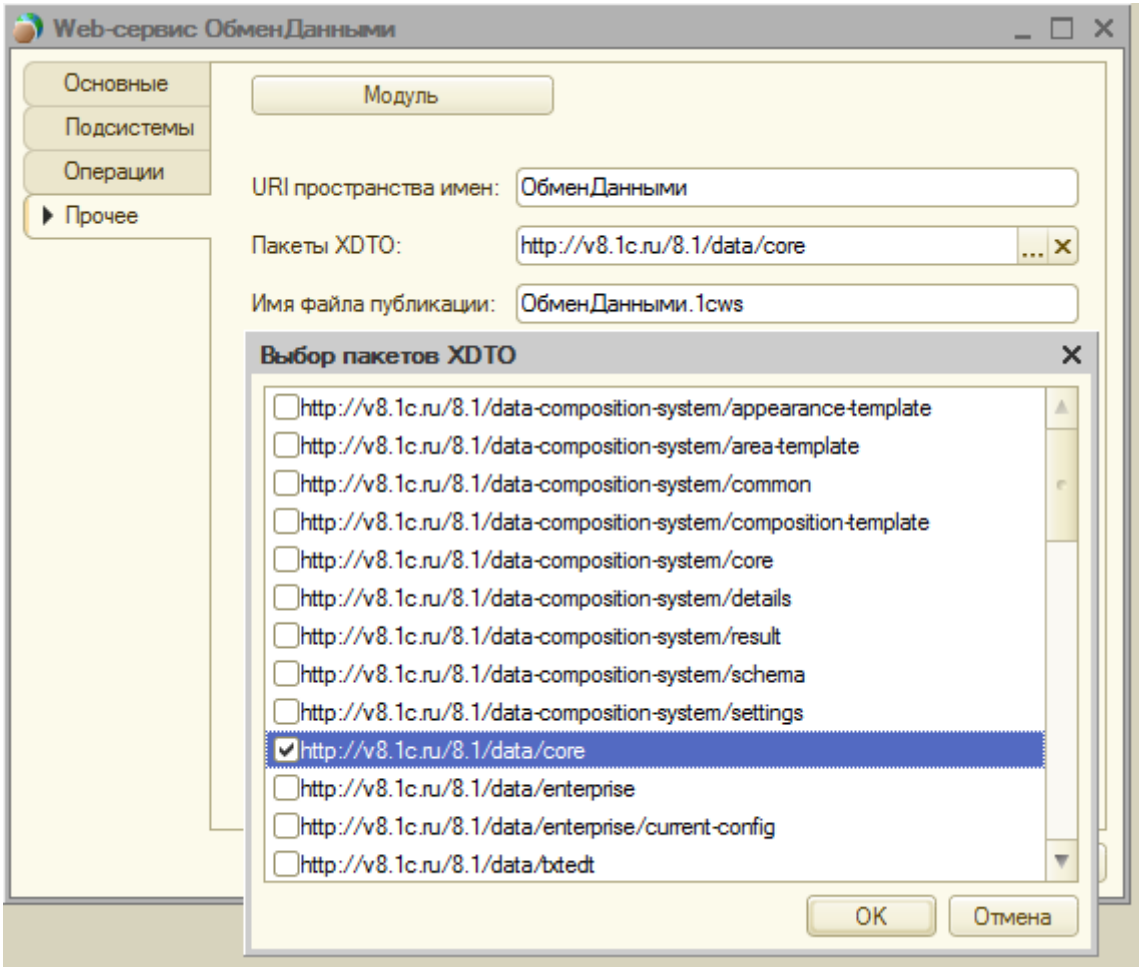
Так как хранилище может сжимать информацию, а это очень важно с учетом того, что на мобильных устройствах мобильный интернет слабее. И в хранилище можно передавать любую информацию.

Хранилище – это объект 1С, он не описан в мировом стандарте W3C. Значит нам надо узнать его пространство имен и имя типа. Открываем справку в 1С по этому объекту и читаем:

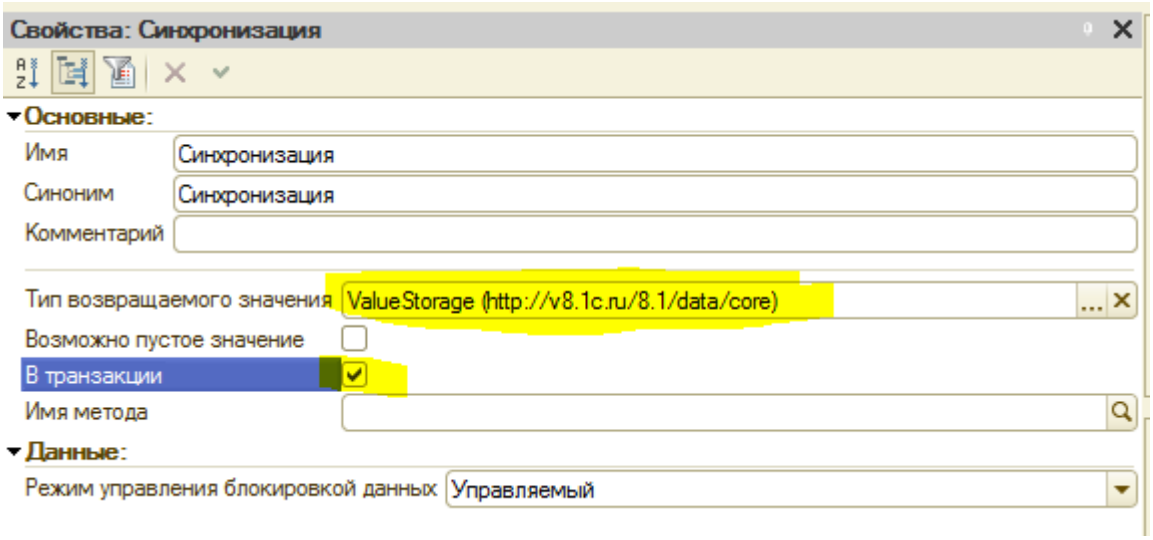
*Возможен обмен с сервером. Сериализуется. Данный объект может быть сериализован в/из XML. Может использоваться в реквизитах управляемой формы. Данный объект может быть сериализован в/из XDTO. Тип XDTO, соответствующий данному объекту, определяется в пространстве имен {**http://v8.1c.ru/8.1/data/core**}. Имя типа XDTO: **ValueStorage**.*

Нас интересует то, что я выделил красным.

При создании веб-сервиса, на вкладке *Прочее* выбираем пакеты *XDTO*, в которых содержится описание нужного нам типа:

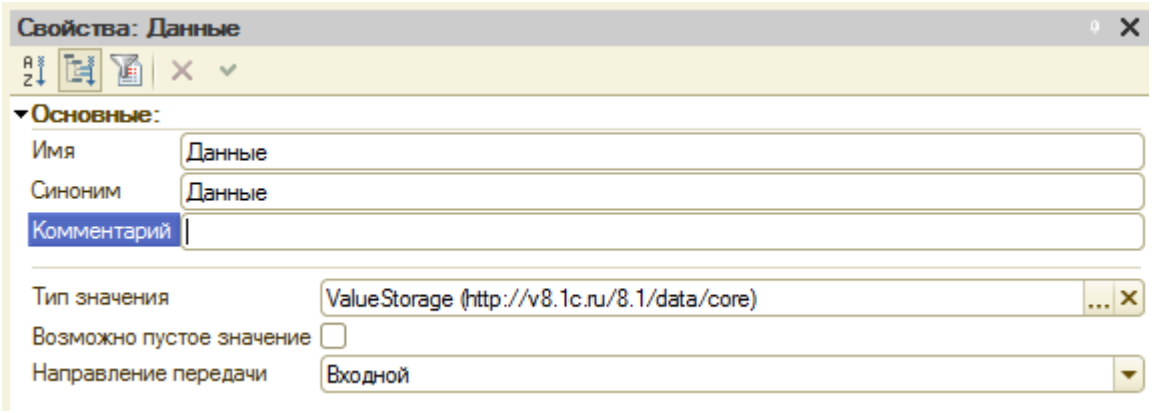


У самой операции указываем тип возвращаемого значения:



Кроме этого, укажем, что весь код нашей функции будет выполнятся в транзакции, так как если возникнет неконтролируемая ошибка, то может нарушиться весь обмен.

Создаем параметр «Данные» и указываем его свойства:

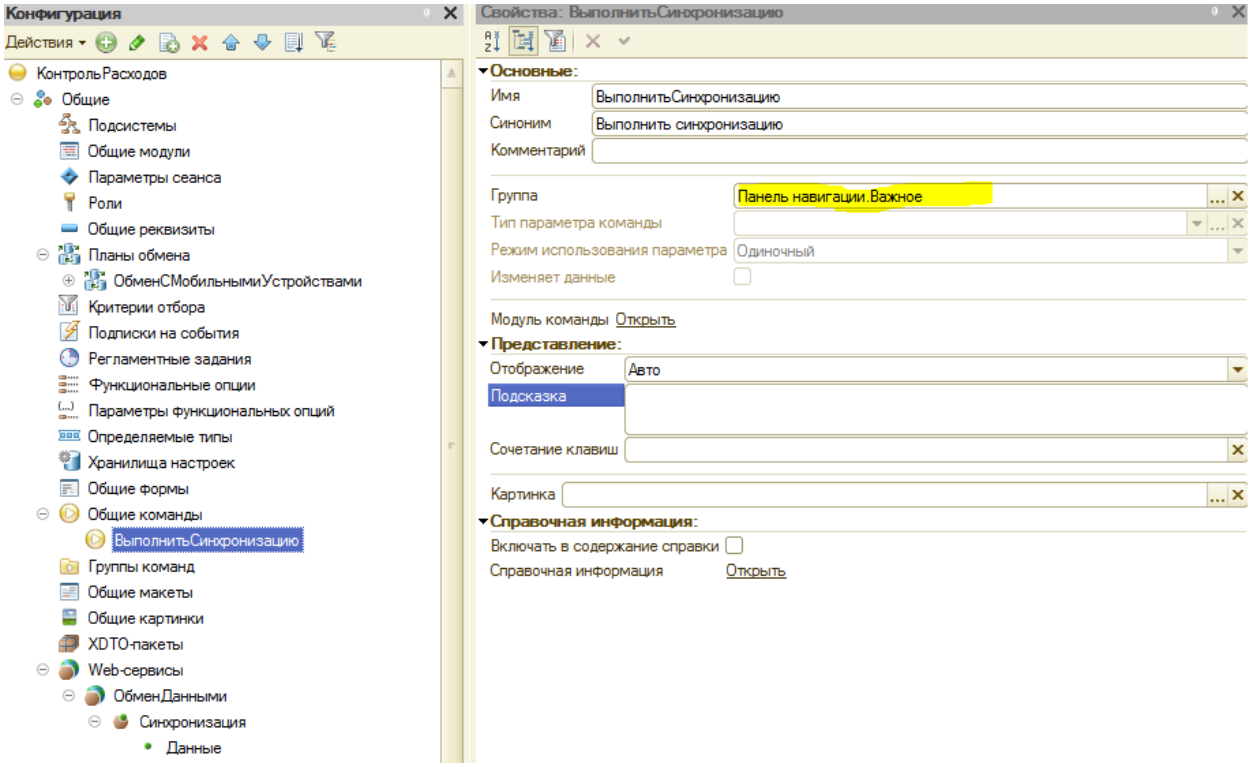


Возвращаемся в свойства самой операции и указываем имя метода, будет создана функция с параметром «Данные».

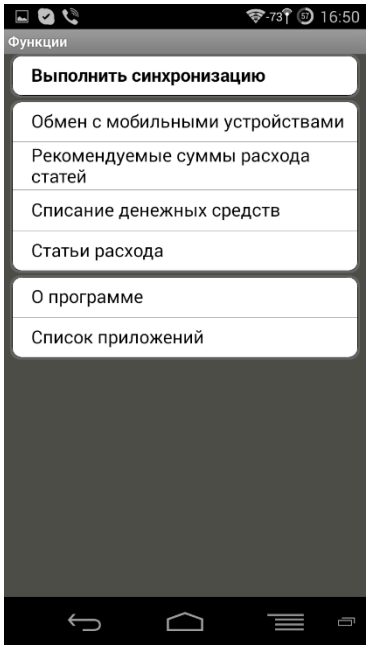
Теперь публикуем базу данных с именем «day3». Убедимся, что все работает. Для этого перейдем по ссылке: http://127.0.0.1/day3/ws/ОбменДанными.1cws?wsdl. И мы должны увидеть структуру *wsdl*-схемы.

Серверную часть сделана, теперь нужно описать клиентскую. Помним, что и сервер, и клиент мы делаем в одной конфигурации.

Создадим Общую команду «ВыполнитьСинхронизацию»:



Разместим ее в группе «Важное», обновим на телефоне базу и убедимся, что команда появилась:



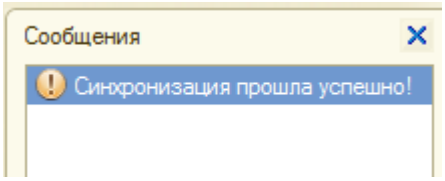
Теперь опишем код этой команды (функция синхронизации оставлена в этом модуле команды для удобства освоения материала, но так делать не правильно! Необходимо эту функцию вынести в общий модуль):

```
&НаКлиенте
Процедура ОбработкаКоманды(ПараметрКоманды, ПараметрыВыполненияКоманды)
   Если ВыполнитьСинхронизациюНаСервере() Тогда
      Сообщить("Синхронизация прошла успешно!");
   Иначе
      Сообщить("При синхронизации были ошибки!");
   КонецЕсли;
КонецПроцедуры

&НаСервере
Функция ВыполнитьСинхронизациюНаСервере()
   Попытка
      ВСОпределение  = Новый WSОпределения("http://192.168.1.2/day3/ws/ОбменДанными.1cws?wsdl");
      ВСервис     = ВСОпределение.Сервисы.Получить("ОбменДанными","ОбменДанными");
      ВТочкаВхода  = ВСервис.ТочкиПодключения.Получить("ОбменДаннымиSoap");
      ВОперация     = ВТочкаВхода.Интерфейс.Операции.Получить("Синхронизация");

      Данные = Новый ХранилищеЗначения("Некие данные", Новый СжатиеДанных(9));

      ДанныеXDTO = ВСОпределение.ФабрикаXDTO.Создать(ВОперация.Параметры.Получить("Данные").Тип,
Данные);

      ВСПрокси = Новый WSПрокси(ВСОпределение,
"ОбменДанными","ОбменДанными","ОбменДаннымиSoap") ;
      Ответ = ВСПрокси.Синхронизация(ДанныеXDTO);
      Возврат Истина
   Исключение
      Сообщить(ОписаниеОшибки());
      Возврат Ложь;
   КонецПопытки;
КонецФункции
```

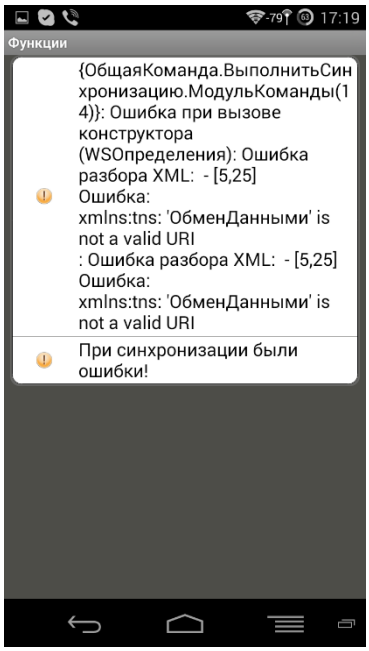
Конечно, путь к базе у каждого будет свой. В веб-сервисе пропишем вот такой код:

```
Функция Синхронизация(Данные)
   Возврат Новый ХранилищеЗначения("Ответ сервера", Новый СжатиеДанных(9));
КонецФункции
```

Запустим тонкий клиент и выполним эту команду, тонкий клиент сообщит:



Теперь выполним команду на мобильном телефоне:

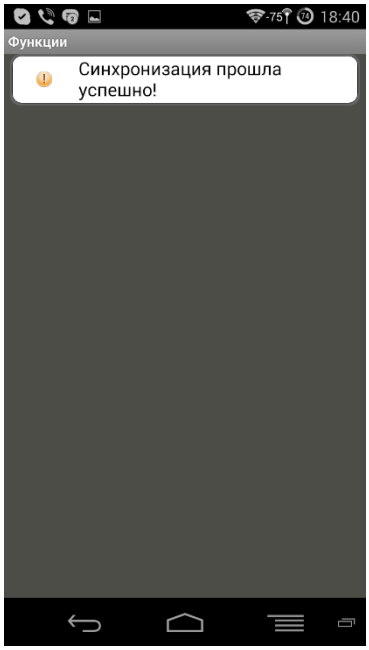


Мы получили ошибку из-за того, что изначально не следовали тем правилам создания веб-сервисов, о которых мы говорили ранее. Т.е. все, что выходит во внешний интернет, все имена параметров, пути подключения и т.д. должно быть на латинице, а еще лучше, если будет подчинятся правилам, по которым формируются имена переменных.

Давайте переименуем *URI* пространство имен веб-сервиса в “DataTransfer”, и отобразим это изменение в коде:

```
ВСервис  = ВСОпределение.Сервисы.Получить("DataTransfer","ОбменДанными");
ВСПрокси = Новый WSПрокси(ВСОпределение, "DataTransfer","ОбменДанными","ОбменДаннымиSoap") ;
```

Обновим конфигурацию на мобильном устройстве и выполним команду:



Все прошло успешно. Теперь мы можем перейти к написанию самого обмена.

Выгрузка изменений на клиент

Но, перед тем, как делать выгрузку, мы должны понимать – куда выгружать данные. Т.е. в какой узел. По этой причине мы должны на сервер передать код узла, для которого делается выгрузка. Для этого изменим немного код в нашей команде синхронизации:

```
   Данные = Новый ХранилищеЗначения(ПланыОбмена.ОбменСМобильнымиУстройствами.ЭтотУзел().Код,
Новый СжатиеДанных(9));
```

А в веб-сервисе пропишем такой код:

```
Функция Синхронизация(Данные)
   ОтветКлиенту = ЗарегестрироватьВыгрузку(Данные.Получить());
   Возврат Новый ХранилищеЗначения(ОтветКлиенту, Новый СжатиеДанных(9));
КонецФункции

Функция ЗарегестрироватьВыгрузку(КодУзла)
   ЗаписьXML = Новый ЗаписьXML;
   ЗаписьXML.УстановитьСтроку();
   ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();

   Узел = ПланыОбмена.ОбменСМобильнымиУстройствами.НайтиПоКоду(КодУзла);
```

```
  ЗаписьСообщения.НачатьЗапись(ЗаписьXML,Узел);
  ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(Узел,ЗаписьСообщения.НомерСообщения);
  Пока ВыборкаИзменений.Следующий() Цикл
    ОбъектОбмена = ВыборкаИзменений.Получить();
    ЗаписатьXML(ЗаписьXML,ОбъектОбмена);
  КонецЦикла;

  ЗаписьСообщения.ЗакончитьЗапись();

  Возврат ЗаписьXML.Закрыть();
КонецФункции
```

Обновим клиент и выполним синхронизацию еще раз. Поставим отладку в веб-сервисе и посмотрим, что он нам вернет:

```
<v8msg:Message xmlns:v8msg="http://v8.1c.ru/messages">
  <v8msg:Header>
    <v8msg:ExchangePlan>ОбменСМобильнымиУстройствами</v8msg:ExchangePlan>
    <v8msg:To>Моб1</v8msg:To>
    <v8msg:From>ЦБ</v8msg:From>
    <v8msg:MessageNo>1</v8msg:MessageNo>
    <v8msg:ReceivedNo>0</v8msg:ReceivedNo>
  </v8msg:Header>
  <v8msg:Body>
    <CatalogObject.СтатьиРасхода>
      <Ref>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</Ref>
      <IsFolder>false</IsFolder>
      <DeletionMark>false</DeletionMark>
      <Parent>00000000-0000-0000-0000-000000000000</Parent>
      <Code>000000001</Code>
      <Description>Проезд</Description>
    </CatalogObject.СтатьиРасхода>
    <InformationRegisterRecordSet.РекомендуемыеСуммыРасходаСтатей>
      <Filter>
        <Period>2014-04-06T00:00:00</Period>
        <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
      </Filter>
      <Records>
        <Record>
          <Period>2014-04-06T00:00:00</Period>
          <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
          <Сумма>20</Сумма>
        </Record>
      </Records>
    </InformationRegisterRecordSet.РекомендуемыеСуммыРасходаСтатей>
    <DocumentObject.СписаниеДенежныхСредств>
      <Ref>9c1e9c2a-bd8b-11e3-bf13-1c6f653dd5bd</Ref>
      <DeletionMark>false</DeletionMark>
      <Date>2014-04-06T16:01:47</Date>
```

```
      <Number>000000001</Number>
      <Posted>true</Posted>
      <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
      <Сумма>10</Сумма>
    </DocumentObject.СписаниеДенежныхСредств>
  </v8msg:Body>
</v8msg:Message>
```

Это структура сообщения. Давайте разберем ее подробно.

Разбор пакета сообщений

Вначале идет определение шапки сообщения:

```
  <v8msg:Header>
    <v8msg:ExchangePlan>ОбменСМобильнымиУстройствами</v8msg:ExchangePlan>
    <v8msg:To>Моб1</v8msg:To>
    <v8msg:From>ЦБ</v8msg:From>
    <v8msg:MessageNo>1</v8msg:MessageNo>
    <v8msg:ReceivedNo>0</v8msg:ReceivedNo>
  </v8msg:Header>
```

Тут указывается по какому плану обмена идет выгрузка, откуда и куда (вот почему код важен), и номер принятого и отправленного сообщения. Таким образом, мы не сможем повторно принять один и тот же пакет, так как при загрузке идет проверка, если номер последнего принятого больше или равен тому, что указан в документе программа не загрузит тогда этот пакет.

В следующем теге идет описание объектов.

Вот так выглядит выгрузка для справочника:

```
    <CatalogObject.СтатьиРасхода>
      <Ref>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</Ref>
      <IsFolder>false</IsFolder>
      <DeletionMark>false</DeletionMark>
      <Parent>00000000-0000-0000-0000-000000000000</Parent>
      <Code>000000001</Code>
      <Description>Проезд</Description>
    </CatalogObject.СтатьиРасхода>
```

Т.е. описание всего того, что есть в справочнике.

Вот так выглядит документ:

```
<DocumentObject.СписаниеДенежныхСредств>
```

```
    <Ref>9c1e9c2a-bd8b-11e3-bf13-1c6f653dd5bd</Ref>
    <DeletionMark>false</DeletionMark>
    <Date>2014-04-06T16:01:47</Date>
    <Number>000000001</Number>
    <Posted>true</Posted>
    <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
    <Сумма>10</Сумма>
  </DocumentObject.СписаниеДенежныхСредств>
```

Обратите внимание на тег *СтатьяРасхода*, там указан только *UID* объекта, но не указано, что это за объект. А мы знаем, что для того, чтобы искать по *UID,* необходимо указать объект поиска, т.е. конкретно, что это за справочник, или документ.

Именно поэтому, если реквизит в базе приемнике будет, к примеру, строковый, то в базу просто попадет *UID* объекта, а не его имя.

Но самым интересным является выгрузка регистра сведений:

```
  <InformationRegisterRecordSet.РекомендуемыеСуммыРасходаСтатей>
    <Filter>
      <Period>2014-04-06T00:00:00</Period>
      <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
    </Filter>
    <Records>
      <Record>
        <Period>2014-04-06T00:00:00</Period>
        <СтатьяРасхода>0828dc1b-bd8a-11e3-bf13-1c6f653dd5bd</СтатьяРасхода>
        <Сумма>20</Сумма>
      </Record>
    </Records>
  </InformationRegisterRecordSet.РекомендуемыеСуммыРасходаСтатей>
```

Обратите внимание, сначала идет фильтр, а потом данные. Таким образом, если мы изменим запись в регистре (именно изменим, например, поменяем статью), то в отборах будет старая статья, а в записи – новая. Таким образом, 1С однозначно понимает, какую запись добавить, а какую удалить.

Загрузка данных на клиенте

Теперь напишем загрузку данных. Для этого пропишем загрузку:

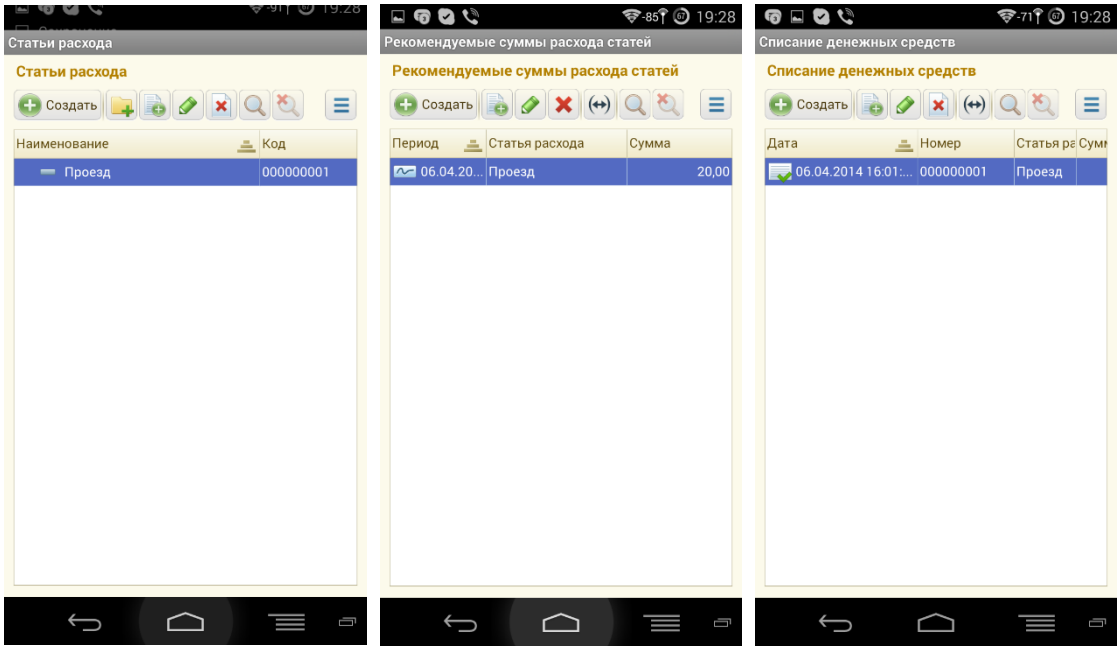
```
Процедура ПринятьИзменениеПоПлану(СтрокаСообщения);
```

```
  ЧтениеХМЛ = Новый ЧтениеXML;
  ЧтениеХМЛ.УстановитьСтроку(СтрокаСообщения);

  ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
  ЧтениеСообщения.НачатьЧтение(ЧтениеХМЛ);

  Пока ВозможностьЧтенияXML(ЧтениеХМЛ) Цикл
     Данные = ПрочитатьXML(ЧтениеСообщения.ЧтениеXML);
     Если Не Данные = Неопределено Тогда

        Данные.ОбменДанными.Отправитель = ЧтениеСообщения.Отправитель;
        Данные.ОбменДанными.Загрузка = Истина;

        Данные.Записать();

     КонецЕсли;
  КонецЦикла;

  ЧтениеСообщения.ЗакончитьЧтение();
КонецПроцедуры
```

И вызовем ее после получения ответа от веб-сервиса:

```
Ответ = ВСПрокси.Синхронизация(ДанныеXDTO);
ПринятьИзменениеПоПлану(Ответ.Получить());
Возврат Истина
```

Обновим клиент и запустим синхронизацию. Все должно выгрузиться на мобильное устройство:



Но, если мы нажмем на синхронизацию еще раз и отловим сообщение которые возвращает веб-сервис, то увидим, что он опять возвращает тоже самое. Только меняет номер выгрузки. Вспоминаем, что мы должны серверу теперь сообщить информацию о том, что мы приняли данные. Причем, мы можем выгрузить ответ серверу точно так же, как формировали ответ клиенту.

Однако, что если нам не надо ничего переносить с мобильного устройства на сервер? Например, в случае, когда мы используем гибридный вариант обмена, т.е. справочную информацию мы забираем по плану обмена из промежуточной базы, а вот заказы – передаем напрямую в ЦБ, минуя промежуточную базу.

Кроме этого, можем и на сервере прописать так, чтобы изменения не записывались вообще никакие с клиента, или только по условию. Реализуем это – будем с клиента загружать только те документы, у которых сумма документа больше 100.

Перед тем, как получить ответ от сервера мы должны послать ему наши данные, в которых, в том числе, будет и ответ. Тогда сервер удалит из выгрузки информацию, которую мы уже загрузили.

Добавим код в нашей команде:

```
Данные = Новый ХранилищеЗначения(СформироватьСообщениеСерверу(), Новый СжатиеДанных(9));
```

И опишем эту функцию:

```
Функция СформироватьСообщениеСерверу()
  ЗаписьXMЛ = Новый ЗаписьXML;
  ЗаписьXMЛ.УстановитьСтроку();
  ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();

  Узлы = ПланыОбмена.ОбменСМобильнымиУстройствами.Выбрать();
  Пока Узлы.Следующий() Цикл
    Если Узлы.Ссылка <> ПланыОбмена.ОбменСМобильнымиУстройствами.ЭтотУзел() Тогда
      Узел = Узлы.Ссылка;
    КонецЕсли;
  КонецЦикла;

  ЗаписьСообщения.НачатьЗапись(ЗаписьXMЛ,Узел);
  ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(Узел,ЗаписьСообщения.НомерСообщения);
  Пока ВыборкаИзменений.Следующий() Цикл
    ОбъектОбмена = ВыборкаИзменений.Получить();
    ЗаписатьXML(ЗаписьXMЛ,ОбъектОбмена);
  КонецЦикла;

  ЗаписьСообщения.ЗакончитьЗапись();

  Возврат ЗаписьXMЛ.Закрыть();

КонецФункции
```

В свою очередь, на сервере в веб-сервисе перепишем функцию:

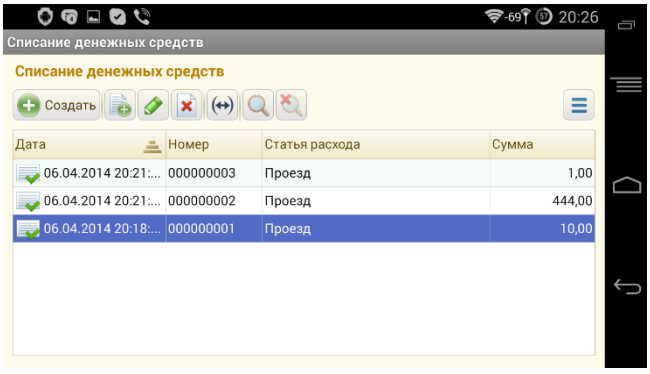
```
Функция ЗарегестрироватьВыгрузку(СтрокаСообщения)
  ЧтениеXMЛ = Новый ЧтениеXML;
  ЧтениеXMЛ.УстановитьСтроку(СтрокаСообщения);

  ЧтениеСообщения = ПланыОбмена.СоздатьЧтениеСообщения();
  ЧтениеСообщения.НачатьЧтение(ЧтениеXMЛ);
  Узел = ЧтениеСообщения.Отправитель;
  ПланыОбмена.УдалитьРегистрациюИзменений(
        ЧтениеСообщения.Отправитель,ЧтениеСообщения.НомерПринятого);

  Пока ВозможностьЧтенияXML(ЧтениеXMЛ) Цикл
    Данные = ПрочитатьXML(ЧтениеСообщения.ЧтениеXML);

    Если Не Данные = Неопределено Тогда
      Если ТипЗнч(Данные) = Тип("ДокументОбъект.СписаниеДенежныхСредств") Тогда
        Если Данные.Сумма < 100 Тогда Продолжить КонецЕсли;
      КонецЕсли;

      Данные.ОбменДанными.Отправитель = Узел;
      Данные.ОбменДанными.Загрузка = Истина;
```

```
        Данные.Записать();

    КонецЕсли;
КонецЦикла;

ЧтениеСообщения.ЗакончитьЧтение();

ЗаписьXML = Новый ЗаписьXML;
ЗаписьXML.УстановитьСтроку();
ЗаписьСообщения = ПланыОбмена.СоздатьЗаписьСообщения();

ЗаписьСообщения.НачатьЗапись(ЗаписьXML,Узел);
ВыборкаИзменений = ПланыОбмена.ВыбратьИзменения(Узел,ЗаписьСообщения.НомерСообщения);
Пока ВыборкаИзменений.Следующий() Цикл
    ОбъектОбмена = ВыборкаИзменений.Получить();
    ЗаписатьXML(ЗаписьXML,ОбъектОбмена);
КонецЦикла;

ЗаписьСообщения.ЗакончитьЗапись();

Возврат ЗаписьXML.Закрыть();
КонецФункции
```

Вот так выглядит конченый вариант регистрации данных на сервере. Обратите внимание – узел я теперь беру с входящего сообщения.

Отбор, который прописан при загрузке – может точно так же быть прописан и в момент формирования пакета выгрузки.

Создадим теперь еще два документа и засинхронизируем базы:



После синхронизации у нас добавился только один документ.

Итоги

Вот так, в двух словах, происходит обмен данными между конфигурациями. Конечно есть еще несколько моментов, которые тоже являются очень важными, например:

- выборочная регистрация изменений;
- префиксы информационных баз;
- первоначальная регистрация изменений для нового узла;
- и т.д.

Но все остальное не имеет смысла рассказывать в контексте мобильной платформы. Т.к. дальше, действуют те же правила, что и при обмене между обычными стационарными конфигурациями.

Кроме этого, мы узнали, что одна и та же конфигурация может использоваться как для мобильного устройства, так и для стационарного.

Где это можно применить?

Рассмотрим пример. На складе работают сотрудники, у них есть терминалы сбора данных с мобильной платформой. Они собирают товар, размещают и т.д. Управляющий склада получает доступ к этой конфигурации с компьютера, где он может строить отчеты, править документы и т.д. перед тем, как вся информация с промежуточной базы попадет в центральную базу. При этом, ему будут доступны все функциональные возможности стационарной платформы 1С.

Кроме этого, наличие промежуточной базы - удобно при отладке. Например, кто-то не может провести документ. Мы делаем полный обмен в демо базу с этим мобильным телефоном, и у нас появляются все его данные, далее мы можем намного проще и быстрее найти ошибки. Например, документ нельзя провести из-за того, что он помечен на удаление. Т.е. без наличия всех данных в некой стационарной базе найти ошибки, завязанные именно на данные – задача нелегкая.

Если Вам интересны аналогичные материалы

Рекомендуем пройти регистрацию на материалы по мобильной платформе: **kursy-po-1c.ru/mobile-apps**

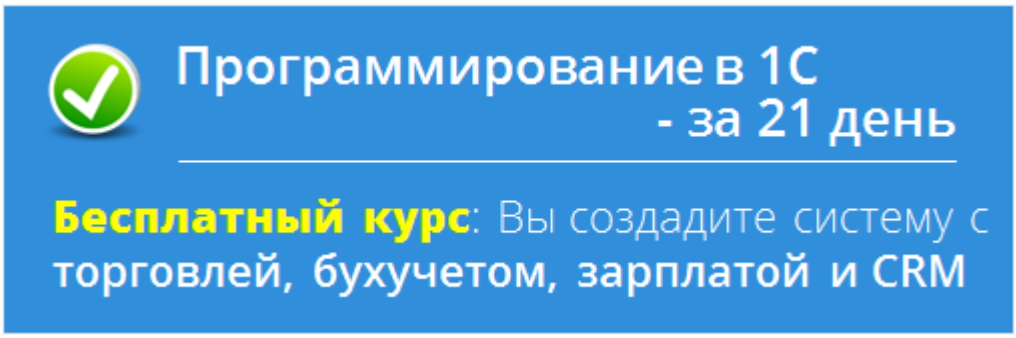
Регистрация для получения текущих и будущих бесплатных материалов проекта: **kursy-po-1c.ru/free**

Дополнительные материалы

Все статьи проекта **Курсы-по-1С.рф**: **http://курсы-по-1с.рф/blog/articles/**

Бесплатные материалы проекта **Курсы-по-1С.рф**: **http://курсы-по-1с.рф/free/**

Бесплатный курс «Программирование в 1С – за 21 день»: http://курсы-по-1с.рф/prog1C-21days/lp1/



Курсы по программированию в 1С v.8

Базовый и Продвинутый курсы по Программированию на Платформе 1С 8 – http://www.Spec8.ru/