Среда разработки платформы КРАФТ – «быстрый старт».

Платформа КРАФТ™ (разработка компании «Бином Софт» – www.binomsoft.ru) является одновременно средой исполнения и системой быстрой разработки бизнес-приложений. Ее отличительной особенностью является функциональная завершенность и независимость от других средств разработки. Платформа КРАФТ является основой для тиражной системы автоматизации предприятий **КРАФТ™ ERP** (www.crafterp.ru) и системы для учета в оффшорных компаниях **ELF** (www.elferp.com). Платформа разработки КРАФТ может быть использована для быстрой и экономичной разработки клиент-серверных приложений любого уровня сложности, включая построение общей информационной системы предприятия. Платформа решает типичные проблемы, стоящие перед разработчиками информационных систем, и предлагает способ для эффективного создания надежных, функциональных, современных и профессиональных приложений, также значительно облегчая их последующее сопровождение.

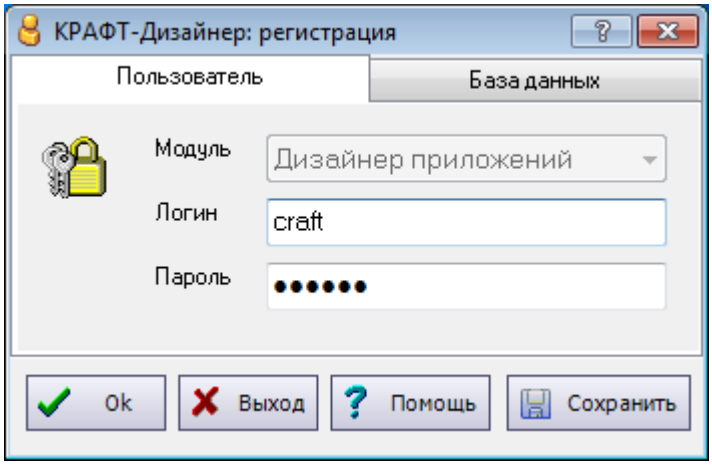
Цель данной статьи – провести краткий обзор средств, которые предоставляет платформа КРАФТ для разработчика при создании приложений или кастомизации уже существующих. Конечно, в рамках одной статьи невозможно рассмотреть все возможности платформы ввиду объемности материала (одних только функций встроенного языка насчитывается более 500). Однако, для начала реальной работы в платформе КРАФТ необходим минимум знаний, которые затем можно расширять по мере необходимости, используя подробное описание в пользовательской документации.

Попробуем решить поставленную задачу на достаточно простом примере доработки существующей конфигурации. Рассмотрение же в рамках статьи полного цикла создания законченного приложения отняло бы слишком много времени (однако некоторые аспекты этого процесса будут затронуты). Кроме этого, данная статья не претендует на курс обучения созданию промышленного программного обеспечения, так как это подразумевает более серьезную проработку схемы данных, бизнес-логики приложения и интерфейса пользователя.

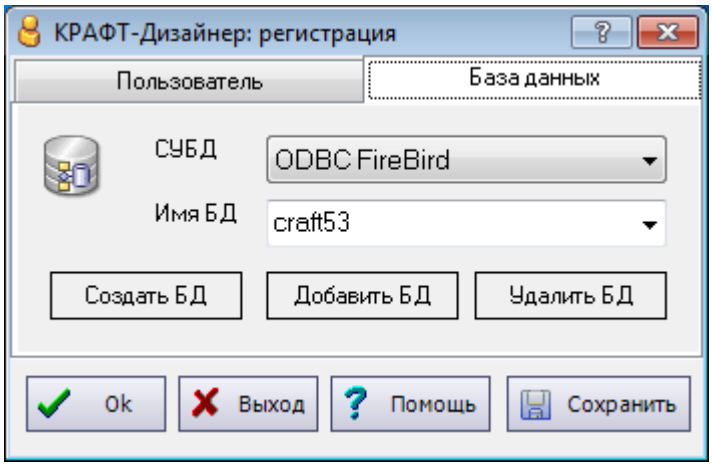
В качестве примера возьмем следующую задачу. Предположим, необходимо доработать существующее приложение CRM (Customer Relationship Management), а именно – добавить в него новую сущность – «Проект», который является планом выполнения работ (задач). «Проект» должен использовать в качестве опорных пунктов плана уже имеющуюся в конфигурации приложения сущность «Задача». Итак, приступим...

1. Создание БД и вход в программу

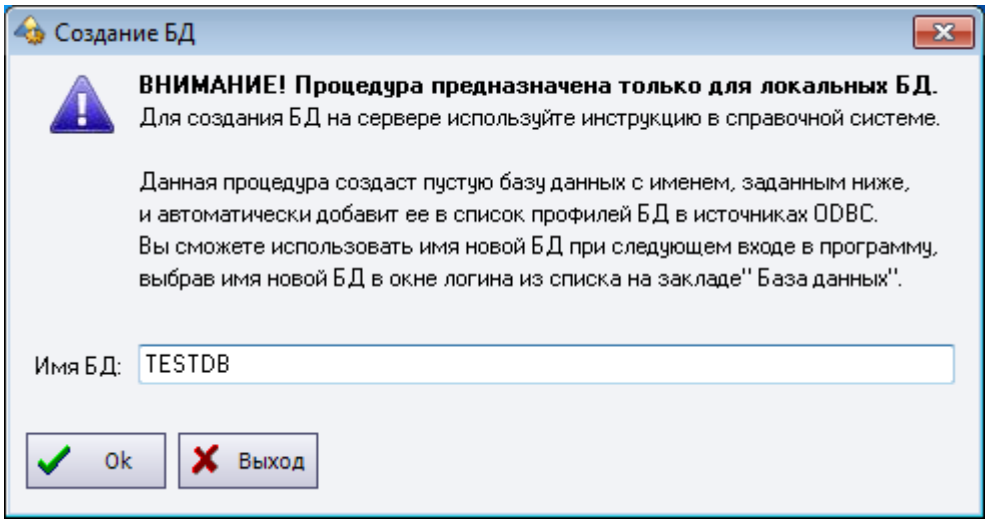
Запускаем модуль «КРАФТ-Дизайнер» 
Откроется диалог регистрации в БД:



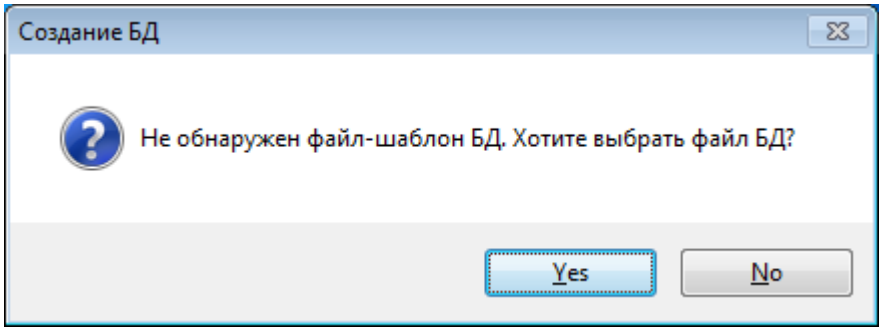
Переключаемся на страницу «База данных» и выбираем тип СУБД (в нашем случае это «ODBC Firebird»).



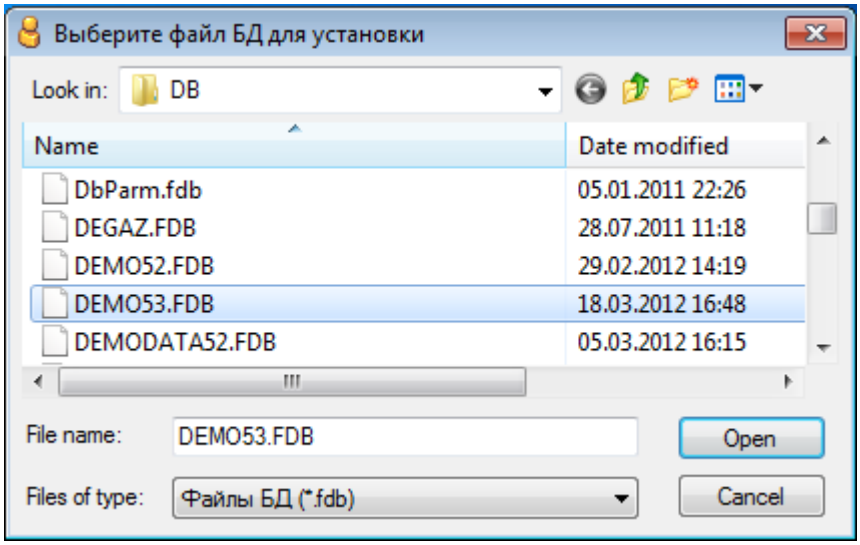
Теперь можно выбрать имя БД для входа из выпадающего списка или создать новую БД. Процедура создания БД также автоматически создает новый профиль в источниках ODBC Windows. Для создания БД нажимаем на кнопку «Создать БД» - откроется диалог, в котором вводим имя для новой БД:



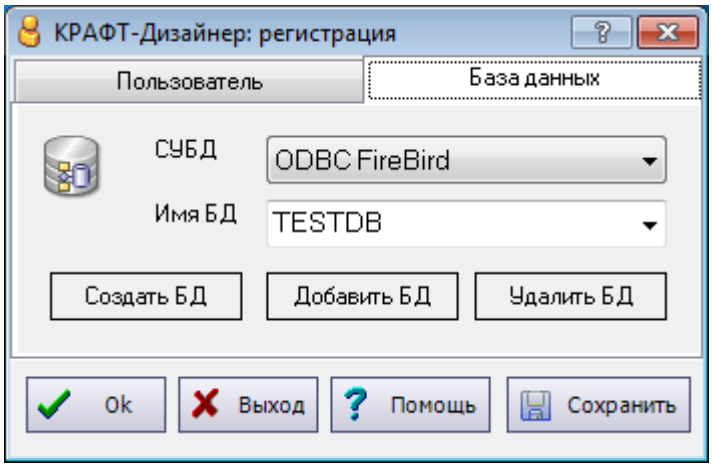
Система попробует найти файл-шаблон для создания новой БД, если он не будет найден, то выведется окно с вопросом:



Если выбрать **Yes**, то будет выведен диалог для выбора файла-шаблона для создания новой БД:



В результате после создания БД имя новой БД будет добавлено в список для выбора в диалоге регистрации, а также будет установлено в поле «Имя БД»:



Теперь остается только нажать на кнопку **Ok** для входа в программу. После загрузки системных данных откроется среда разработки модуля Мастер-сервис.

2. Определение модуля в метаданных системы

Любое приложение, разрабатываемое на платформе КРАФТ, должно быть зарегистрировано в метаданных в виде модуля. Модулем в КРАФТ называется совокупность объектов (визуальных или реализующих бизнес-логику) и объединенных для выполнения конкретной задачи. У каждого модуля имеется свое меню, при помощи которого все объекты приводятся в действие.

Набор визуальных компонентов (бизнес-объектов) платформы КРАФТ включает в себя:

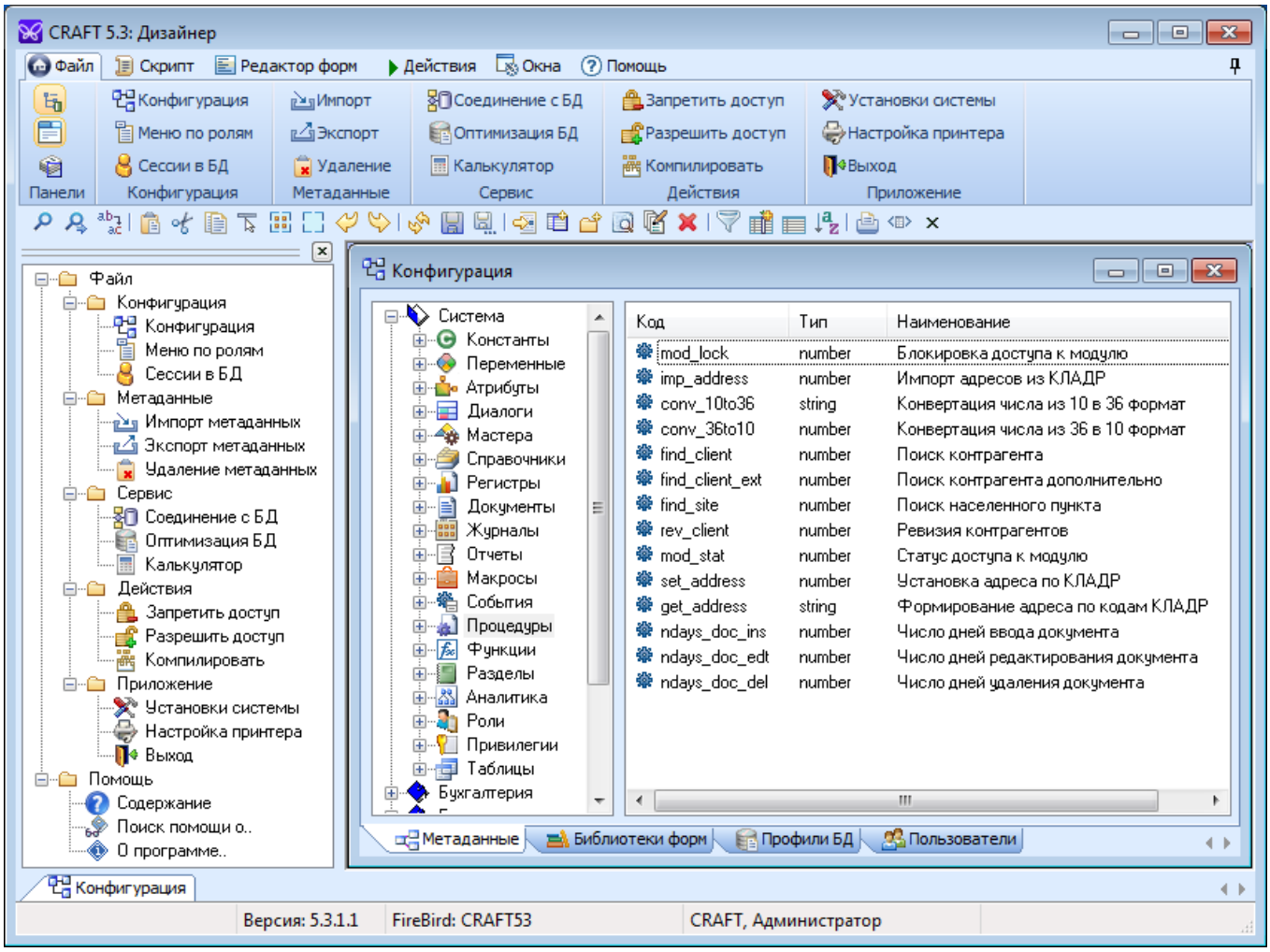
- Диалоги
- Мастера (Wizards)
- Справочники
- Регистры
- Документы и журналы документов
- Отчеты

Для создания бизнес-логики модулей используются:

- Глобальные константы и переменные
- Глобальные макросы (SQL-запросы)
- События и процедуры модуля
- События и процедуры объектов
- Системные функции

Все эти элементы составляют конфигурацию (метаданные модуля), которая хранится в общей БД. Используя среду разработки, программисты могут легко создавать и модифицировать любой элемент конфигурации модуля. Среда разработки платформы КРАФТ предоставляет для программиста мощные и удобные средства для навигации по всем компонентам конфигурации прикладного бизнес-модуля. Окно управления конфигурацией модулей позволяет работать с метаданными модуля, библиотеками интерфейсных форм для бизнес-объектов, профилями БД, а также со списком пользователей системы.

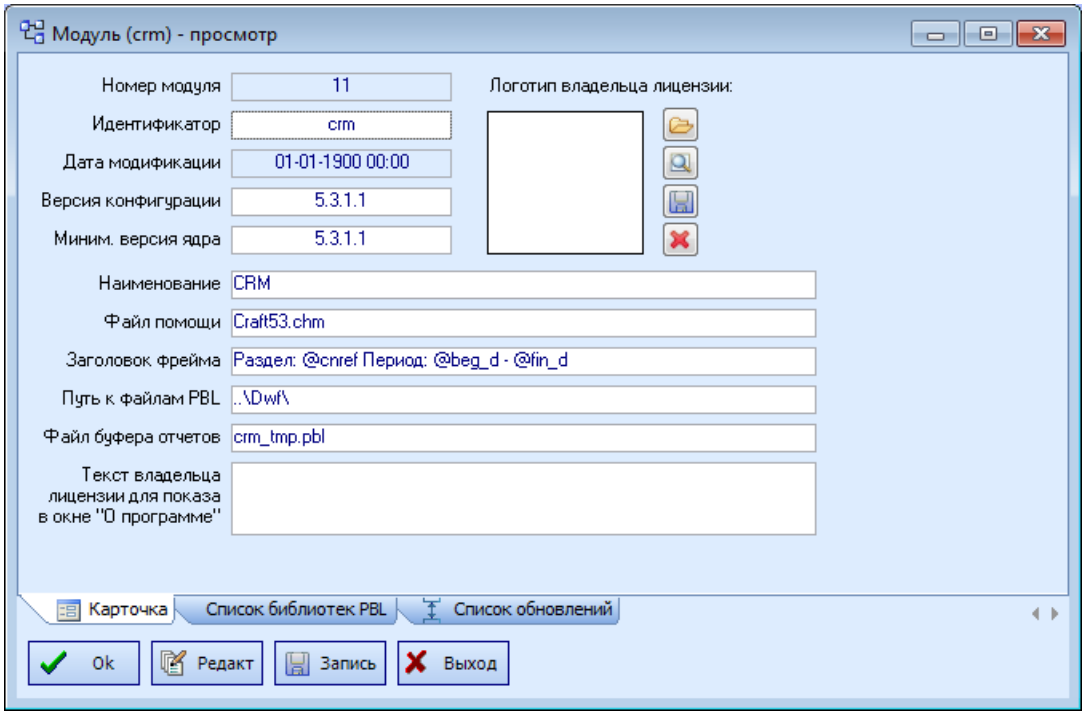
Типичное окно среды разработки (с открытой конфигурацией) показано ниже:



На закладке «Метаданные» в дереве навигатора окна конфигурации представлен список всех зарегистрированных модулей, а также набор объектов каждого модуля. Переключаясь между пунктами дерева навигатора, можно увидеть например, список всех глобальных процедур модуля или список всех справочников модуля. Каждый объект модуля имеет признак доступа: «Модуль» (по умолчанию) или «Общий». Если указан «Модуль», то данный объект доступен только из этого модуля, если «Общий» - то объект доступен из любого модуля.

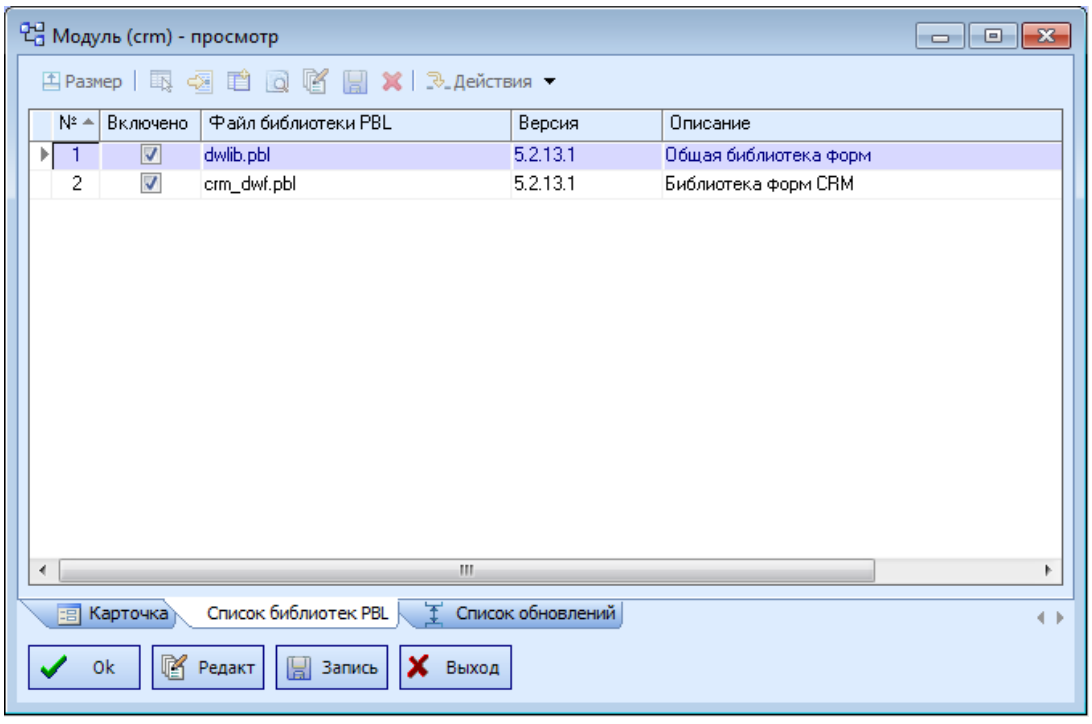
Одной из уникальных возможностей платформы КРАФТ является наличие ссылок на объекты. Посредством ссылок можно организовать выборочный доступ к объектам конфигурации из других модулей, например справочник товарной номенклатуры может быть доступен в модулях «Торговля» и «Производство», а в других модулях – недоступен.

Итак, для нашей задачи мы будем использовать уже готовый модуль CRM. Кратко рассмотрим, как выглядит определение модуля в метаданных (для этого выделим необходимый нам модуль в дереве навигатора конфигурации и вызовем функцию просмотра объекта из контекстного меню по правой кнопке мыши или нажмем на клавишу F3 – Просмотр):



У модуля есть ряд обязательных атрибутов, такие как: номер модуля, идентификатор, наименование. Для модуля может быть указана версия конфигурации данного модуля, а также минимально необходимая версия ядра системы КРАФТ. Также, для модуля может быть указано выражение для текста заголовка окна основного фрейма (тут могут быть использованы ссылки на глобальные константы и переменные конфигурации). Наконец, для модуля должен быть указан путь к библиотекам форм (файлам PBL) и PBL-файл для буфера отчетов. Специально для независимых разработчиков добавлена возможность указать свой логотип и дополнительный текст, который будет показан в стандартном диалоге «О программе».

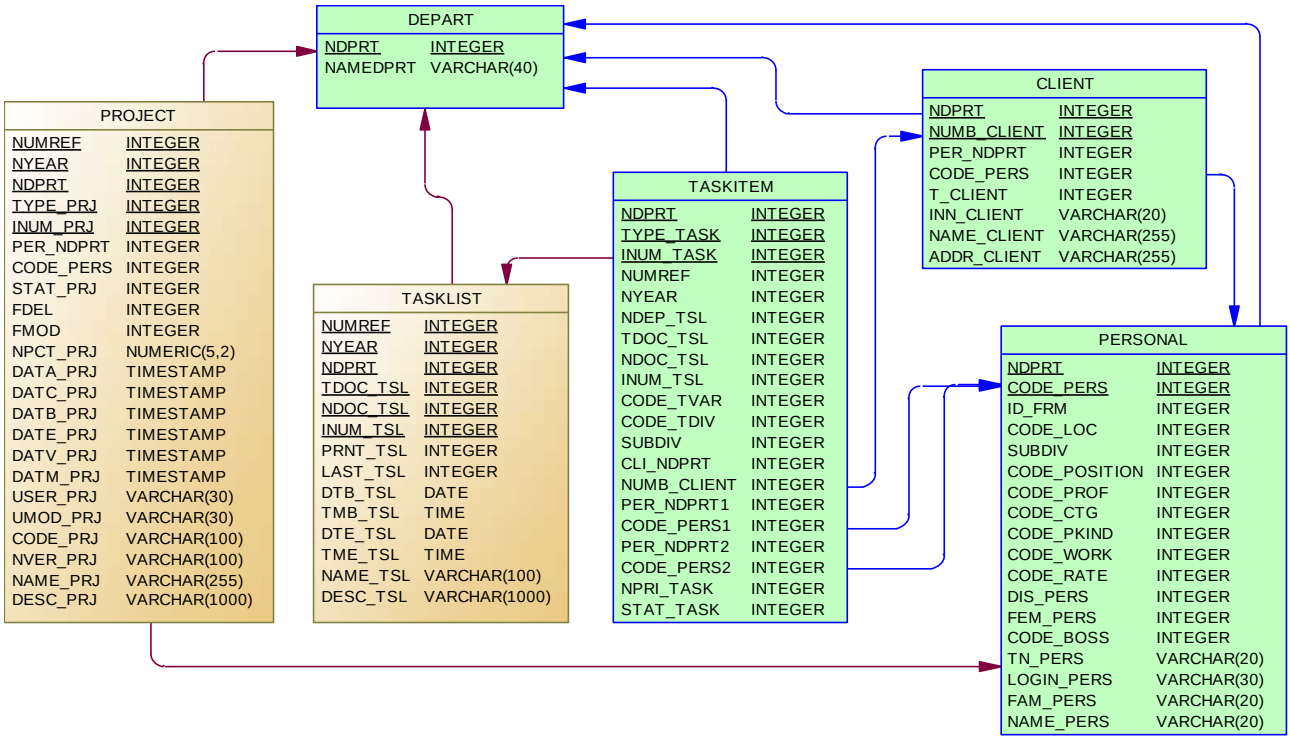
На закладке «Список библиотек PBL» должны быть зарегистрированы все файлы библиотек форм, которые используются данным модулем:



Необходимо, чтобы версии библиотек PBL соответствовали (или были выше) версии конфигурации модуля – это проверяется системой при старте. Версия библиотеки PBL хранится в специальном объекте внутри файла и считывается системой при подключении библиотеки к модулю.

3. Определение структуры данных модуля и создание таблиц

Для решения поставленной задачи необходимо спроектировать соответствующие структуры данных в БД. На диаграмме ниже показаны некоторые таблицы, которые нам понадобятся в работе. Таблицы, выделенные желтым цветом, относятся непосредственно к сущности «Проект», а таблицы зеленого цвета – уже существующие в модуле таблицы, которые будут нами использоваться. Подчеркнутые поля в таблицах являются первичными ключами.

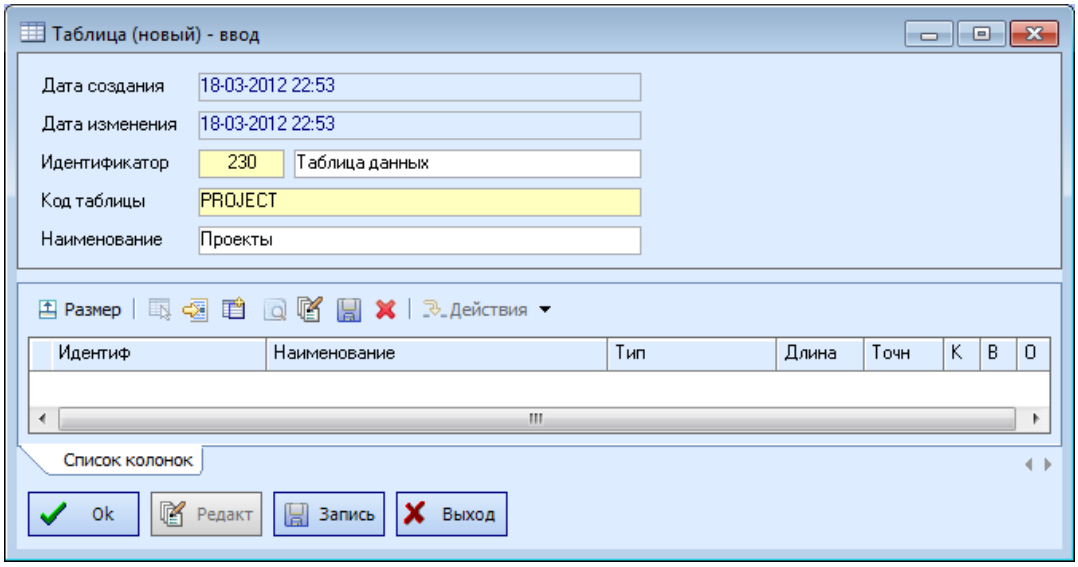


Таблицы на диаграмме имеют следующее назначение:

- PROJECT – основная таблица сущности «Проект»
- TASKLIST – список задач по проекту (план)
- TASKITEM – основная таблица сущности «Задача»
- DEPART – справочник подразделений
- CLIENT – справочник организаций
- PERSONAL – справочник сотрудников

Для создания таблиц в БД можно воспользоваться любой утилитой для администрирования. В случае Firebird SQL для этой цели отлично подходит бесплатная программа IB Expert.

После того, как необходимые таблицы в БД созданы (в нашем случае это таблицы PROJECT и TASKLIST), необходимо зарегистрировать их в метаданных системы. Для этого в окне конфигурации необходимо перейти на модуль Мастер-сервис и выбрать пункт «Таблицы». Далее с помощью контекстного меню по правой кнопке мыши вызвать функцию ввода или нажать клавиши Ctrl+Ins (Ввод). Откроется окно для ввода новой таблицы (идентификатор будет назначен системой автоматически).

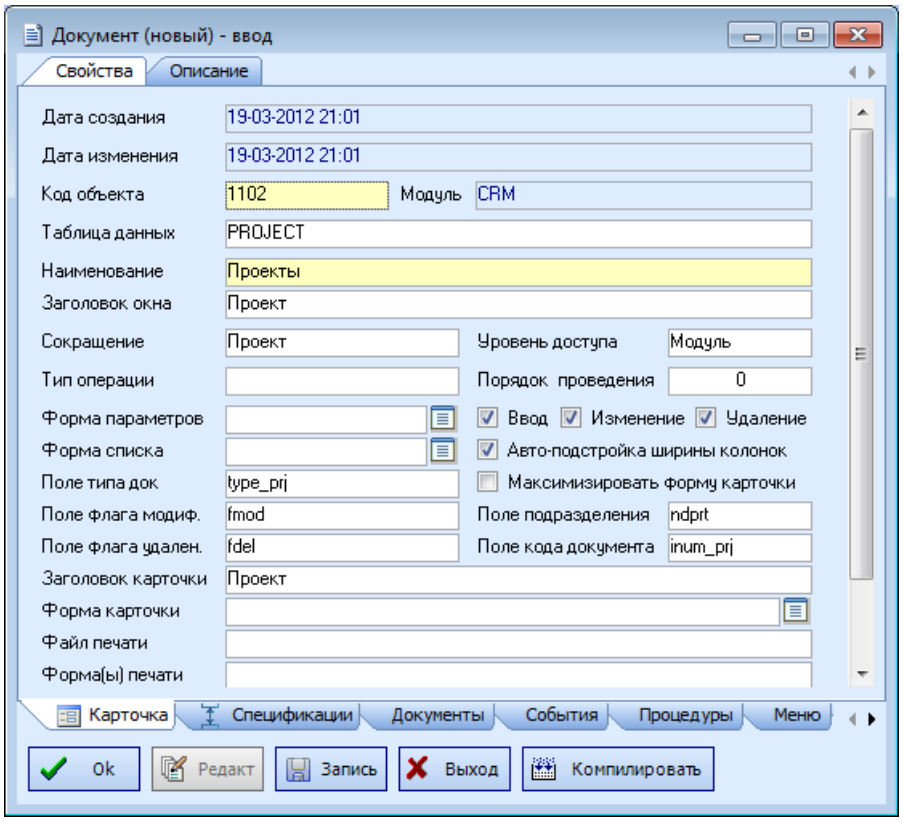


В данном окне обязательно нужно ввести код таблицы (имя в БД) и наименование (описание). Список колонок заполнять необязательно.

Также следует поступить и с таблицей TASKLIST.

4. Создание объектов в конфигурации модуля

Теперь можно приступить к непосредственному созданию бизнес-объектов в конфигурации модуля. Для этого в дереве объектов навигатора конфигурации перейдем на раздел «Документы» в модуле CRM и вызовем функцию ввода из контекстного меню или нажмем клавиши Ctrl+Ins (Ввод). Откроется окно ввода нового объекта «Документ»:



Код объекта был уже назначен системой по умолчанию, но во время ввода мы можем его изменить. Также, необходимо выбрать из списка зарегистрированных в репозитории таблиц основную таблицу для нашего объекта – PROJECT.

Установим основные свойства для нового документа:

- Наименование: «Проекты» (используется для заголовка окна списка)
- Заголовок окна: «Проект» (используется для заголовка окна при редактировании)
- Заголовок карточки: «Проект» (используется для заголовка карточки объекта)
- Сокращение: «Проект» (используется для вывода наименования в сокращенном виде)
- Уровень доступа: «Модуль» (по умолчанию)

Далее нужно указать наименования некоторых полей из таблицы PROJECT, которые несут специальные функции (большинство из них опциональные):

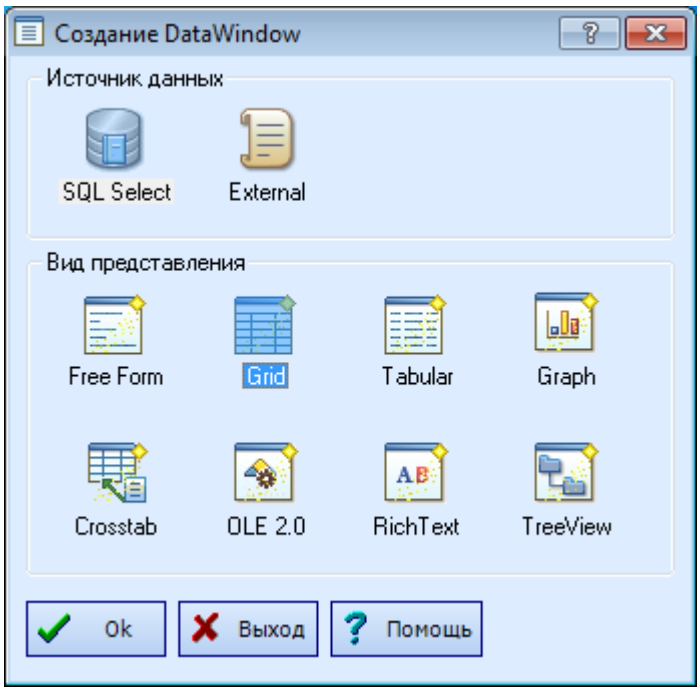
- Поле типа документа: **type_prj** (обязательно, используется для идентификации вида объекта)
- Поле флага модификации: **fmod** (опционально, используется для блокировки при изменении)
- Поле флага удаления: **fdel** (опционально, используется для пометки на удаление)
- Поле подразделения: **ndprt** (опционально, используется для показа номера объекта)
- Поле кода документа: **inum_prj** (опционально, используется для показа номера объекта)

В завершение нажимаем на кнопку **Ок** в диалоге, чтобы сохранить изменения.

Теперь мы можем перейти к созданию форм для объекта «Проект». Так как этот объект имеет тип «Документ», то у него должны быть как минимум две формы: форма списка и форма карточки, идентификаторы которых нужно будет указать в описании объекта. Начнем с формы списка.

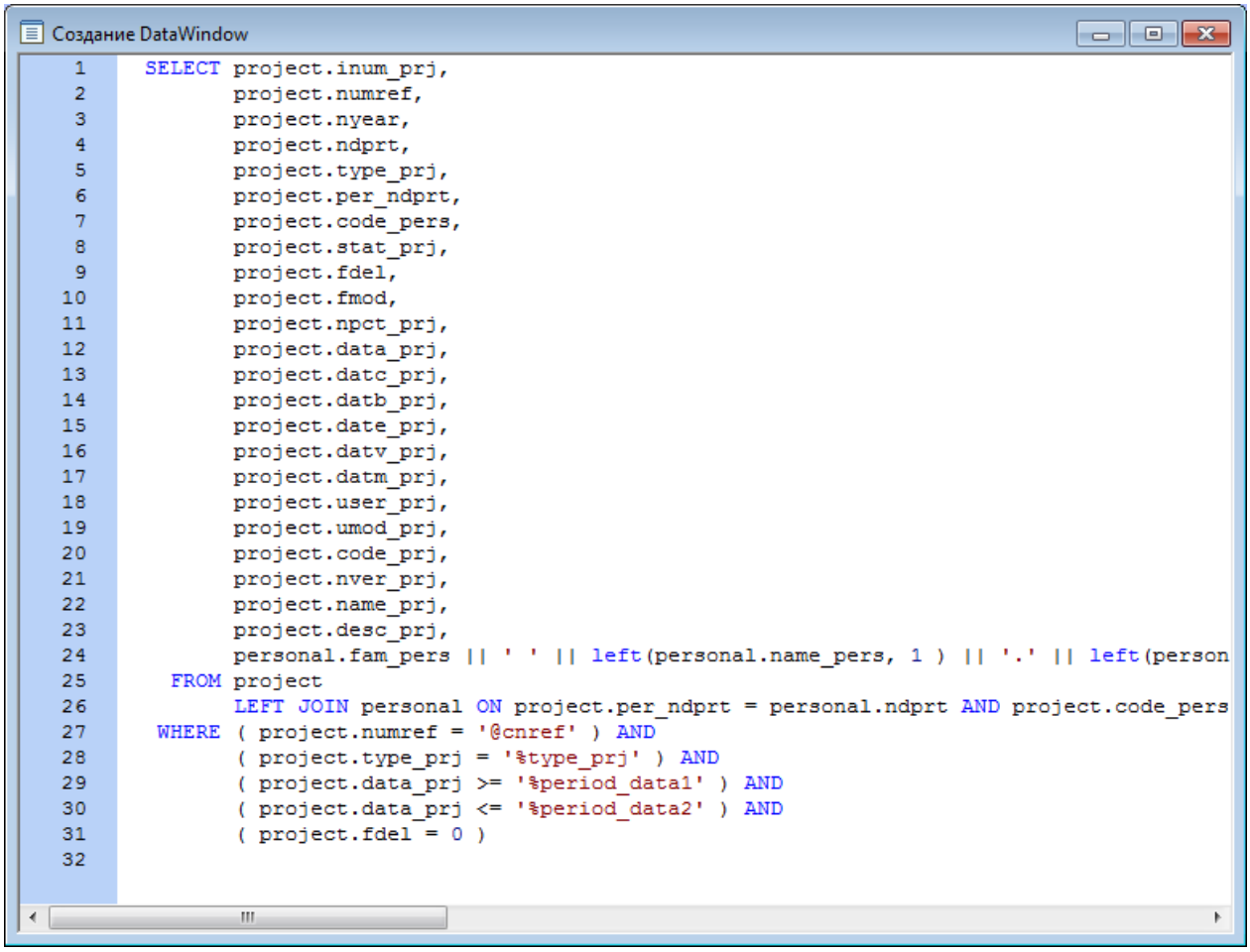
Для создания новой формы можно поступить двумя способами:
1. Закрыть карточку объекта и перейти на закладку «Библиотеки форм» в окне конфигурации, там выбрать зарегистрированную библиотеку модуля и вызвать функцию добавления новой формы.
2. Можно создать новую форму, не выходя из карточки объекта. Для этого нужно нажать на кнопку с пиктограммой «Редактировать форму» справа от поля «Форма списка» в карточке объекта.

Независимо от способа создания новой формы, на экран будет выведен диалог, в котором нужно выбрать тип создаваемой формы:



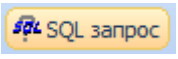
В даиалоге необходимо выбрать два параметра:
1. Источник данных (SQL Select, External) – выберем SQL Select
2. Вид представления данных (Free Form, Grid, Tabular и т.п.) – выберем Grid

Нажимаем кнопку **Ok** и откроется окно дизайнера форм ввода, в котором будет предложено ввести SQL запрос для формы:

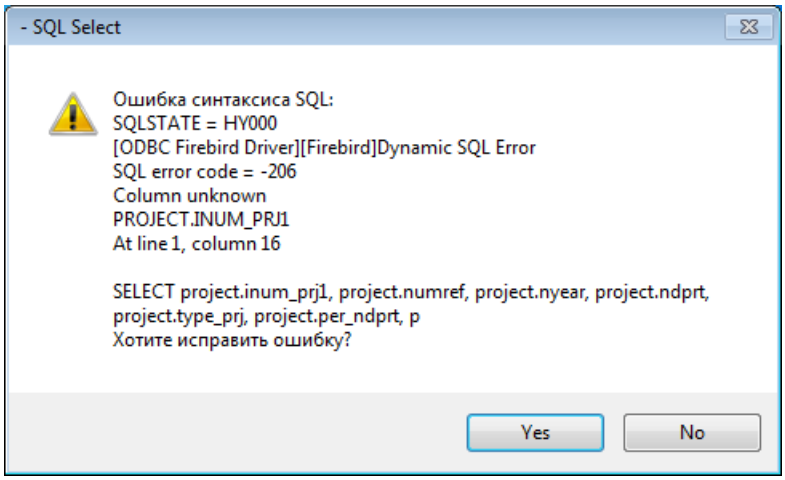


Введем выражение SQL-запроса для формы, как показано на рисунке выше. Данный запрос содержит объединение с таблицей PERSONAL для вывода наименования ответственного лица за проект. Также, отметим другие интересные особенности запроса. Обратим внимание, что в выражении WHERE используются следующие специальные выражения:

@cnref	- это ссылка на глобальную переменную, которая хранит значение текущего раздела учета
%type_prj	- локальная переменная, которая устанавливается системой автоматически при открытии объекта и хранит код объекта в конфигурации модуля (в нашем случае код равен 1102)
%period_data1	- локальная переменная, которая устанавливается системой автоматически и хранит значение даты начала периода просмотра журнала документов
%period_data2	- локальная переменная, которая устанавливается системой автоматически и хранит значение даты конца периода просмотра журнала документов

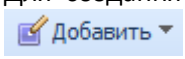
После того, как мы закончили работать над SQL запросом формы, нажмем на кнопку 

в меню редактора. Если SQL запрос содержит ошибки, то будет выведен диалог с предупреждением:

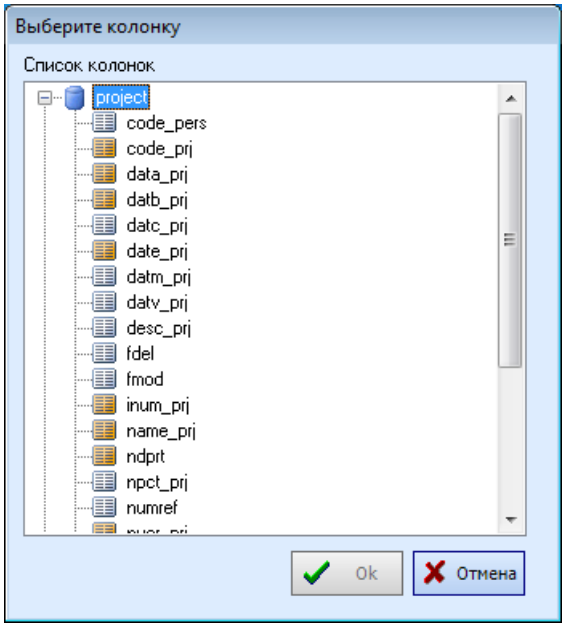


Если ошибок не найдено, то окно ввода SQL запроса исчезнет и откроется окно визуального дизайнера формы.

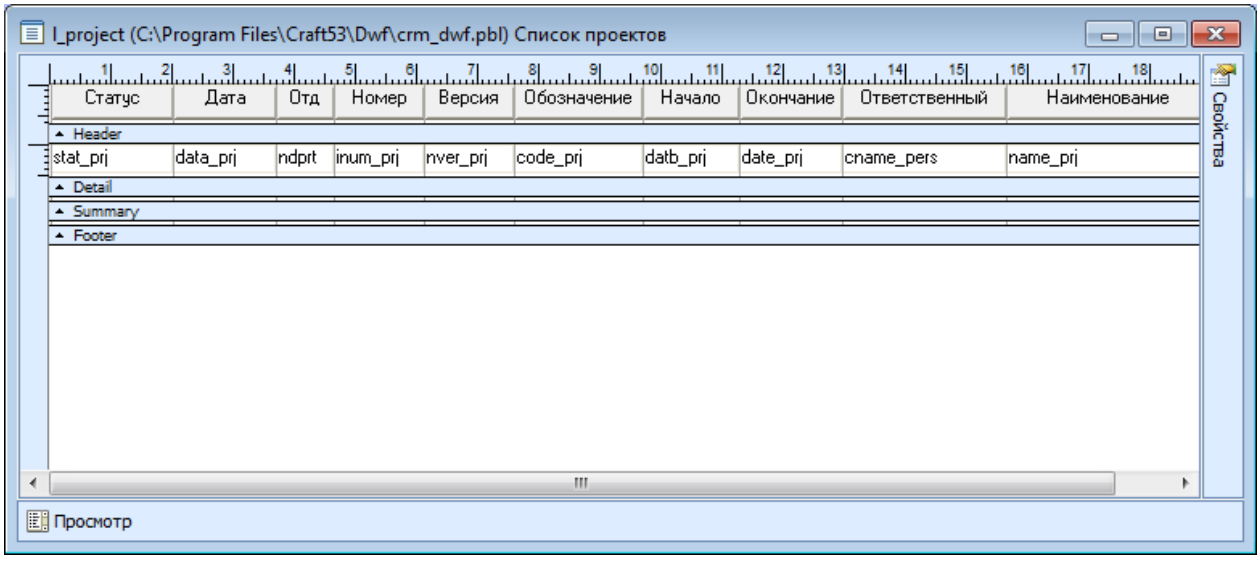
Выведем необходимые для показа поля таблицы на форму. Следует помнить, что любая форма имеет несколько так называемых «областей» (или бандов): Header, Detail, Summary и Footer. Так как при создании мы выбрали тип формы Grid, то поля данных необходимо размещать в области Detail, а поля заголовков – в области Header.

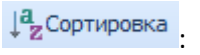
Для создания нового поля данных в форме нужно в меню дизайнера форм активировать пункт 

и выбрать тип объекта – «Поле БД» (всего же возможно использовать такие объекты: Текст, Поле БД, Вычисляемое поле, Линия, Эллипс, Прямоугольник, Кнопка, Изображение, Групповой блок, Отчет). Далее необходимо мышкой указать на место в форме, куда должно быть помещено новое поле. При этом будет выведен диалог со списком всех доступных полей таблиц в SQL запросе и вычисляемых выражений SQL запроса (желтым цветом выделены поля, которые уже присутствуют в форме):

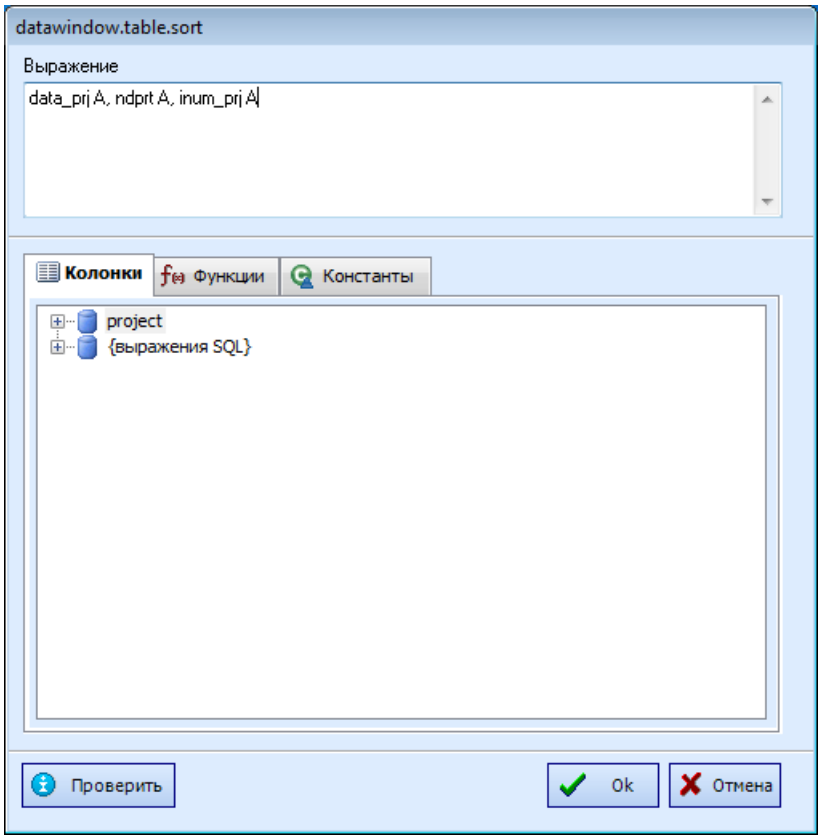


Также, для каждого поля нужно сделать заголовок в форме. Для этого создадим текстовые поля и поместим их над соответствующими полями в области Header и сделаем подписи. В результате у нас должна получиться форма со списком полей, как показано ниже:



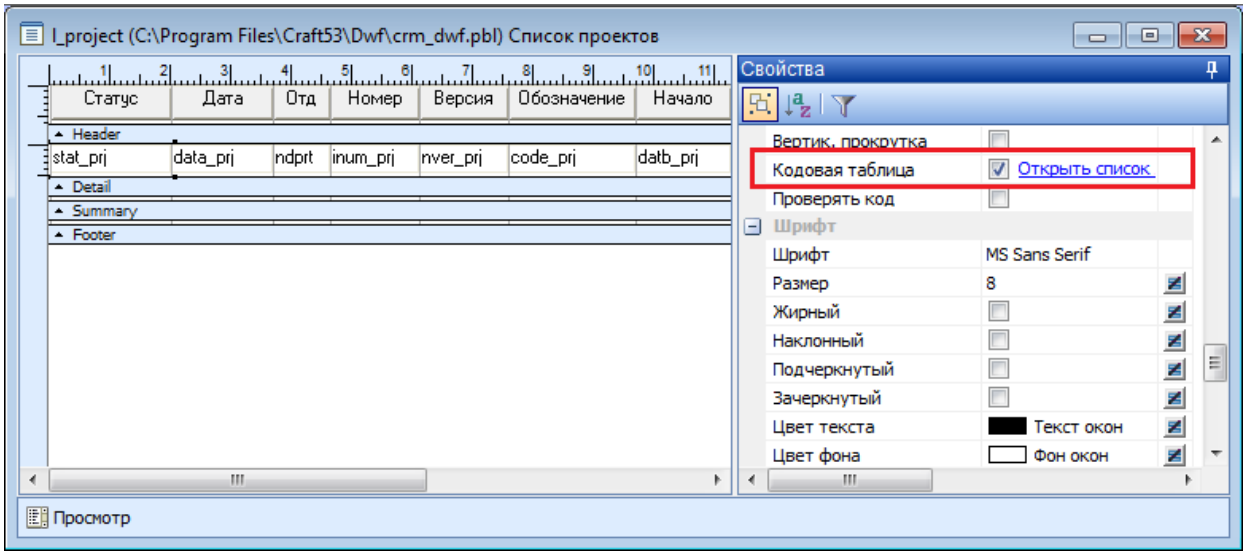
Установим порядок сортировки полей в форме. Для этого вызовем диалог установки сортировки с помощью пункта меню 

:

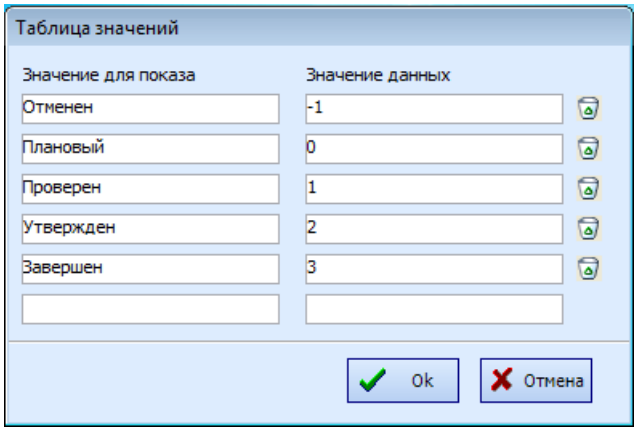


В выражении сортировки необходимо записать порядок полей в форме для сортировки через запятую и признак сортировки (A – Ascendant, D - Descendant).

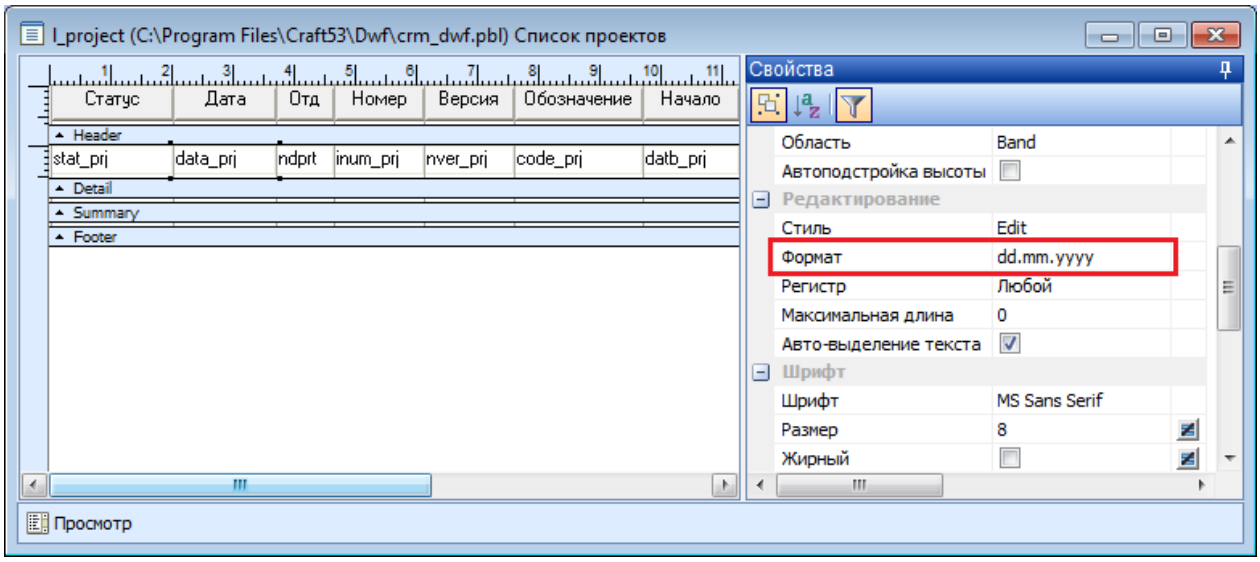
Сделаем еще несколько изменений в форме, которые послужат для более удобного отображения данных. Во-первых, установим формат отображения поля «Статус». Для этого выделим поле **stat_prj** в форме и вызовем панель свойств с помощью меню Свойства. Далее, в панели свойств для поля в разделе «Редактирование» установим отметку на свойстве «Кодовая таблица» (выделено красным прямоугольником):



Кодовая таблица устанавливает соответствие между данными в поле БД и тем значением, которое будет показано вместо него на экране. Чтобы установить набор значений в кодовой таблице, нужно кликнуть мышью на ссылке «Открыть список значений» в панели свойств. При этом откроется диалог, в котором мы установим следующие значения, как показано ниже:

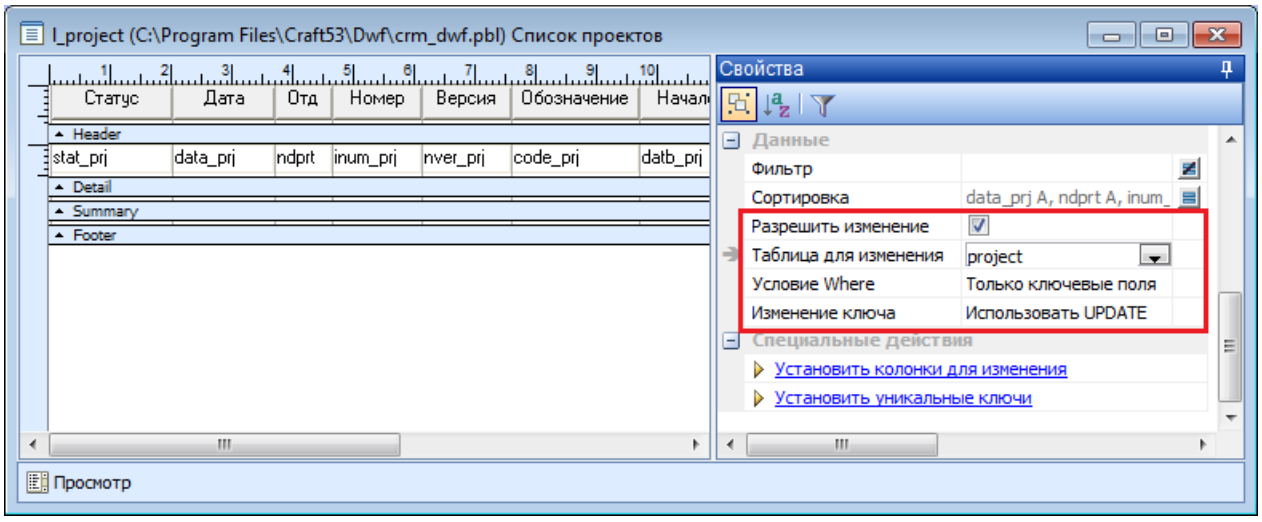


Далее установим формат отображения данных для поля «Дата». Для этого выделим поле **data_prj** в форме и в панели свойств для поля в разделе «Редактирование» установим значение в свойстве «Формат» - ”dd.mm.yyyy” (выделено красным прямоугольником):

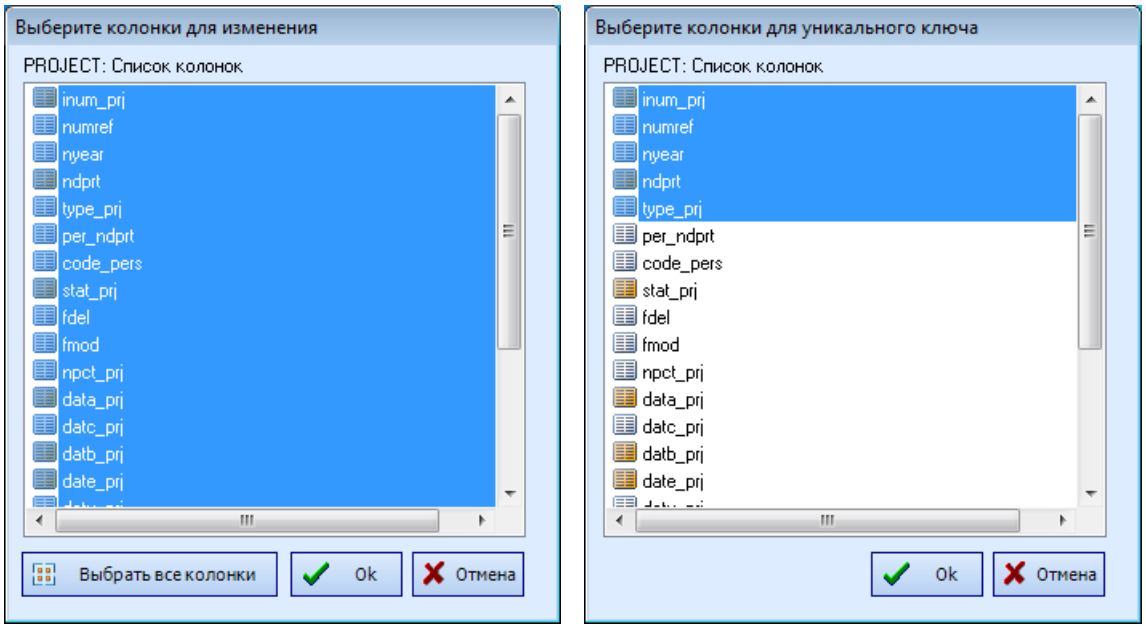


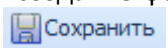
То же самое проделаем для полей «Начало» (**datb_prj**) и «Окончание» (**date_prj**).

В завершение необходимо установить основную таблицу для изменения в форме и указать критерий для обновления данных в таблице. Для этого сначала нужно выбрать объект DataWindow (то есть вся форма) – щелкнуть кнопкой мыши по пустому месту на форме и в панели свойств для DataWindow в разделе «Данные» установить значения свойств, как показано на рисунке ниже (выделено красным прямоугольником):

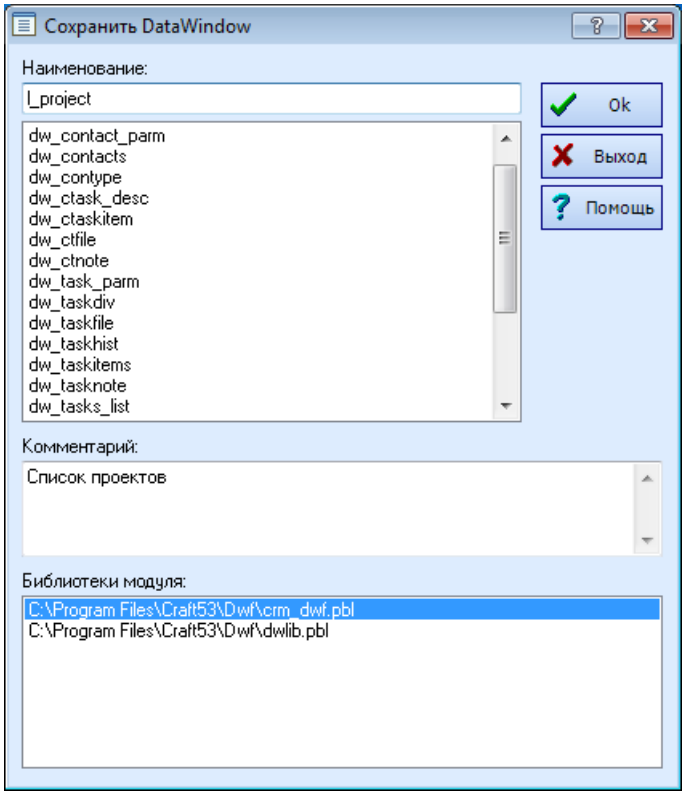


Наконец, необходимо установить колонки для изменения и уникальные ключи таблицы, которые будут использоваться системой при управлении транзакциями в данной форме, при помощи диалогов, как показано ниже (вызываются также из панели свойств для DataWindow):



На этом создание формы списка для документа «Проект» можно считать завершенным. Нажимаем кнопку 

в меню дизайнера – нам выводится окно для выбора библиотеки и указания идентификатора формы:

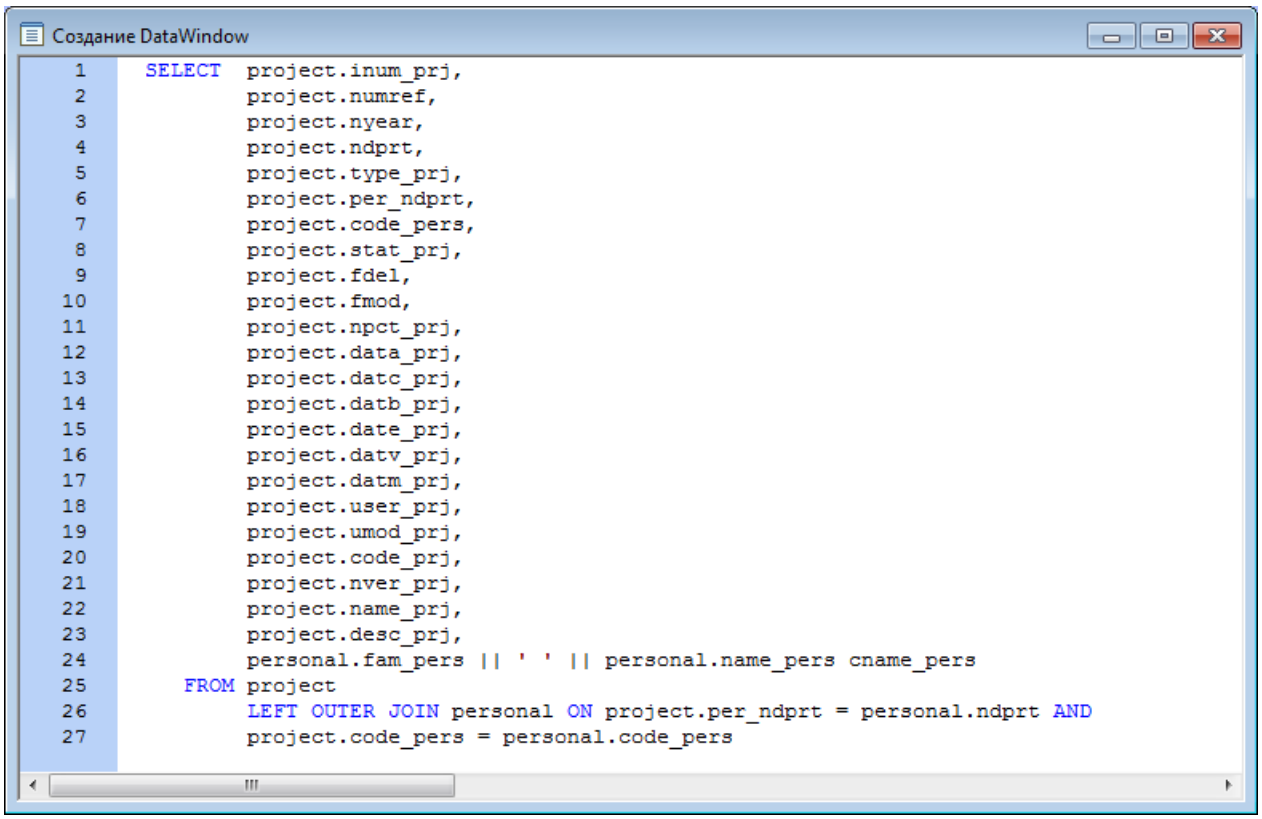


Выберем внизу диалога библиотеку ”C:\Program Files\Craft53\Dwf\crm_dwf.pbl” и укажем в поле «Наименование» идентификатор для формы – **l_project**, а в поле «Комментарий» - пояснительный текст и нажмем на кнопку **Ok**. Новая форма будет записана в библиотеку. Теперь можно вернуться к окну описания объекта «Проект» и указать в качестве формы для списка идентификатор только что созданной формы – **l_project**.

Сейчас нам предстоит создать форму для карточки документа. Таким же образом, как и для формы списка, вызываем диалог для ввода новой формы (нажав на кнопку с пиктограммой справа от

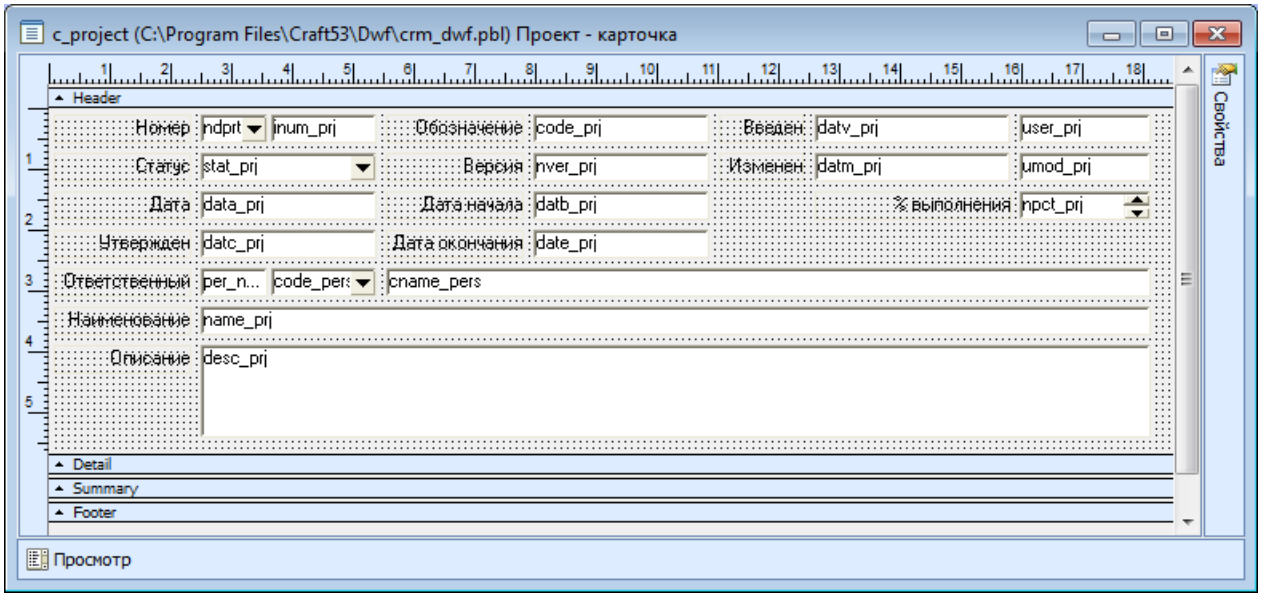
поля «Форма карточки» в окне описания объекта). Только в данном случае выбираем тип представления для формы –« Free Form».

В открывшемся окне вводим SQL-запрос:

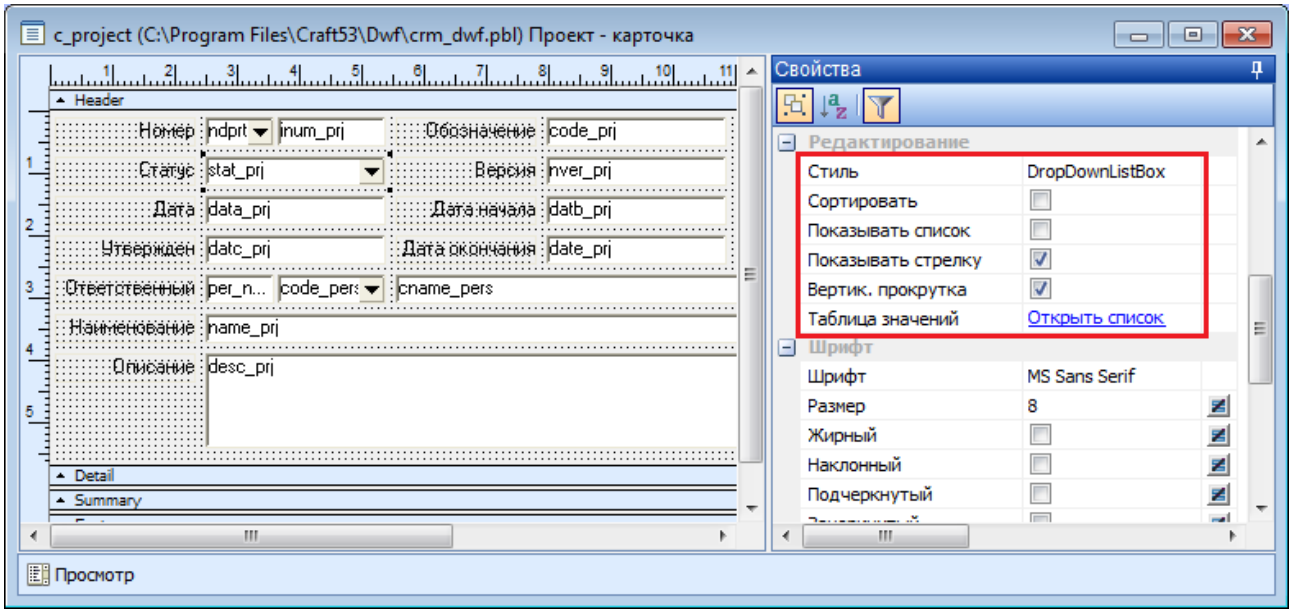


Заметим, что в данном случае мы не вводим никаких выражений WHERE в запрос. Так как форма карточки будет открываться из формы списка документов, то система сама позаботится о том, чтобы дополнить SQL-запрос выражением для отображения конкретного выбранного документа.

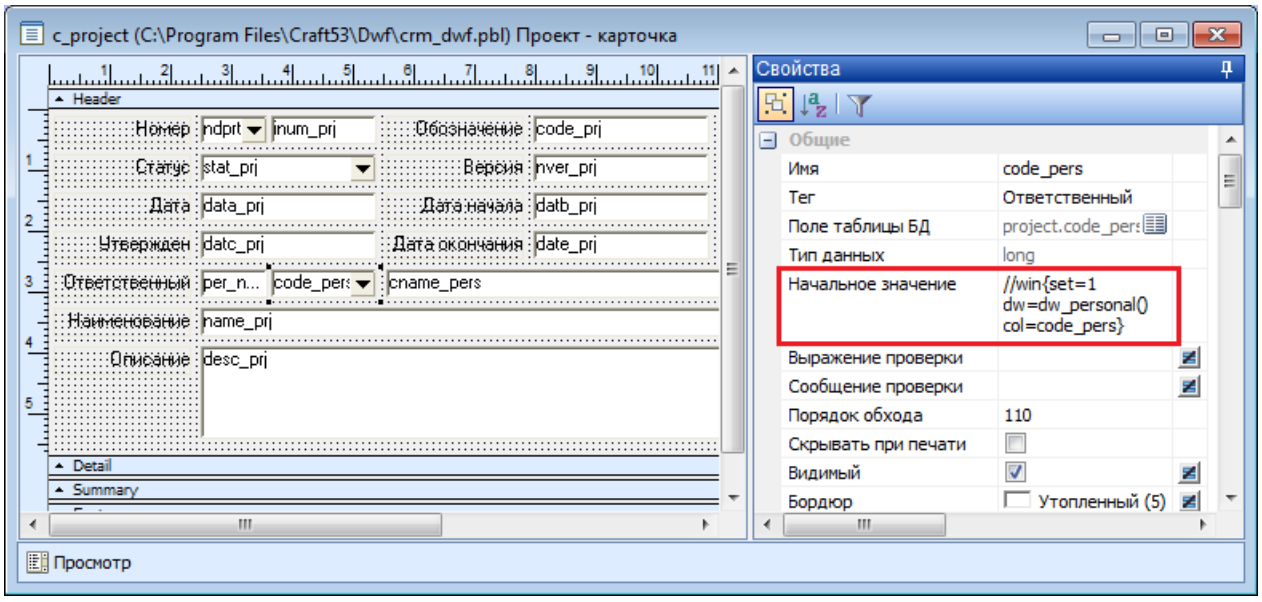
Далее переходим в визуальный дизайнер и создаем текстовые поля и поля БД так, чтобы в результате получилась такая форма:



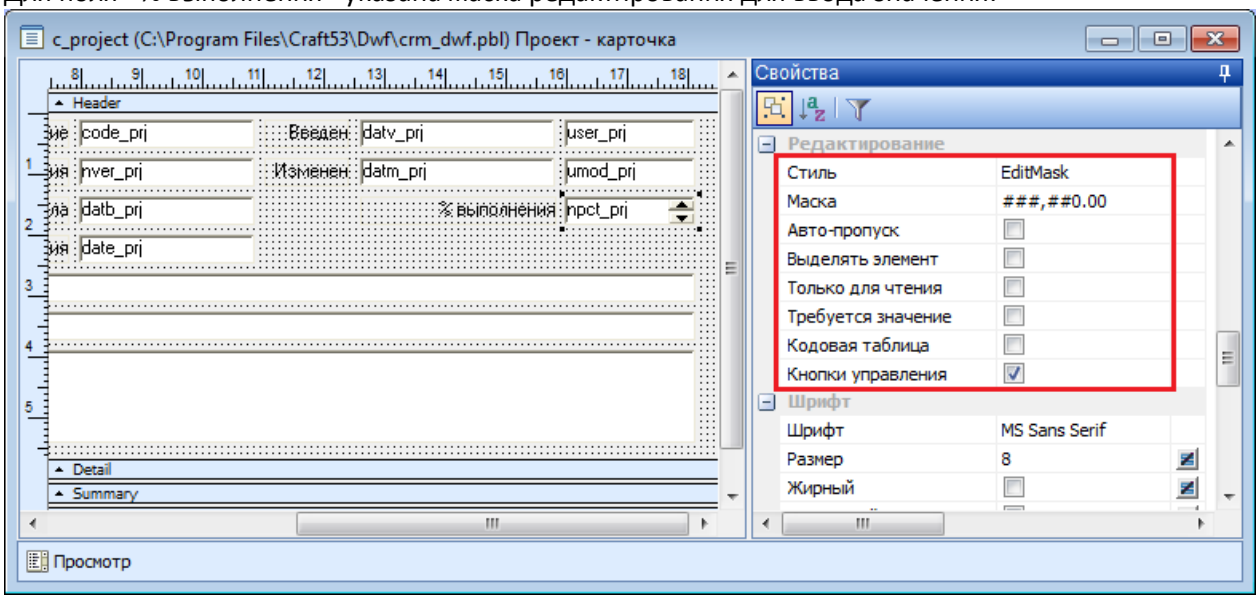
Отметим несколько интересных моментов для созданной формы. Во-первых, для поля «Статус» установлен стиль редактирования «Выпадающий список» и указаны такие же значения в кодовой таблице списка, как и в форме списка документов (выделено красным прямоугольником):



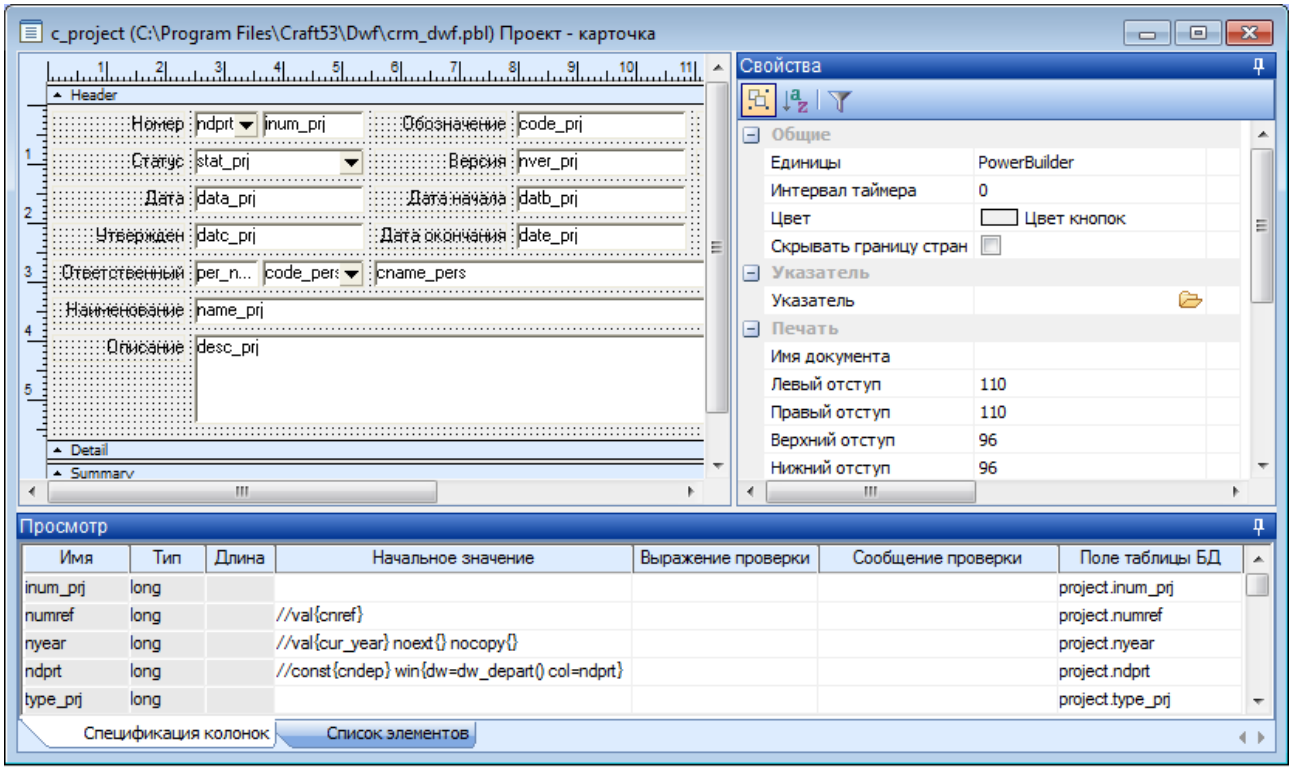
Во-вторых, для поля «Ответственный» (**code_person**) также установлен стиль редактирования «Выпадающий список», однако список значений не установлен. Вместо этого, установлен специальный тег в свойстве «Начальное значение», которое позволяет вызвать диалоговое окно с формой, указанной в теге WIN{} – **dw_personal**, для выбора значений из таблицы БД.



Для поля «% выполнения» указана маска редактирования для ввода значения:



Для завершение дизайна формы необходимо еще позаботиться о том, чтобы при вводе данных все ключевые поля таблицы в форме корректно инициализировались. Значение инициализации для полей формы можно посмотреть в специальной панели «Спецификация колонок», расположенной внизу в окне редактора. Для того чтобы открыть панель, поместим курсор мыши над заголовком панели «Просмотр» – откроется панель просмотра с закладками «Спецификация колонок» и «Список элементов» формы. Можно закрепить панель просмотра в открытом виде с помощью специальной кнопки справа вверху панели .



Введем начальные значения для ключевых полей данной формы:

- numref: //val{cnref}
- nyear: //val{cur_year} noext{} nocopy{}
- ndprt: //const{cndep} win{dw=dw_depart() col=ndprt}

Дадим краткие пояснения к коду инициализации полей. Символ комментария // необходим, чтобы система исполнения КРАФТ воспринимала все, что стоит после него, как специальный код, а не абсолютное значение. После символа // могут идти самые разнообразные теги для управления поведением полей в форме (один мы уже расмотрели выше – WIN{}). Если при этом необходимо указать абсолютное значение для инициализации поля, то его можно указать перед символом //.

Тег VAL{} используется для получения значения глобальной переменной модуля по ее идентификатору. В данном примере происходит обращение к двум таким переменным:

- cnref – значение текущего выбранного раздела учета
- nyear – значение текущего выбранного учетного периода (года)

Тег CONST{} используется для получения значения глобальной константы модуля по ее идентификатору. В данном примере происходит обращение к одной такой константе:

- cndep – значение кода текущего установленного подразделения компании

Наконец, тег WIN{}, как уже было сказано выше, используется для вызова диалогового окна выбора значения из таблицы БД. В данном случае используется выбор из списка подразделений при помощи формы **dw_depart** и значение берется из колонки с идентификатором **ndprt**.

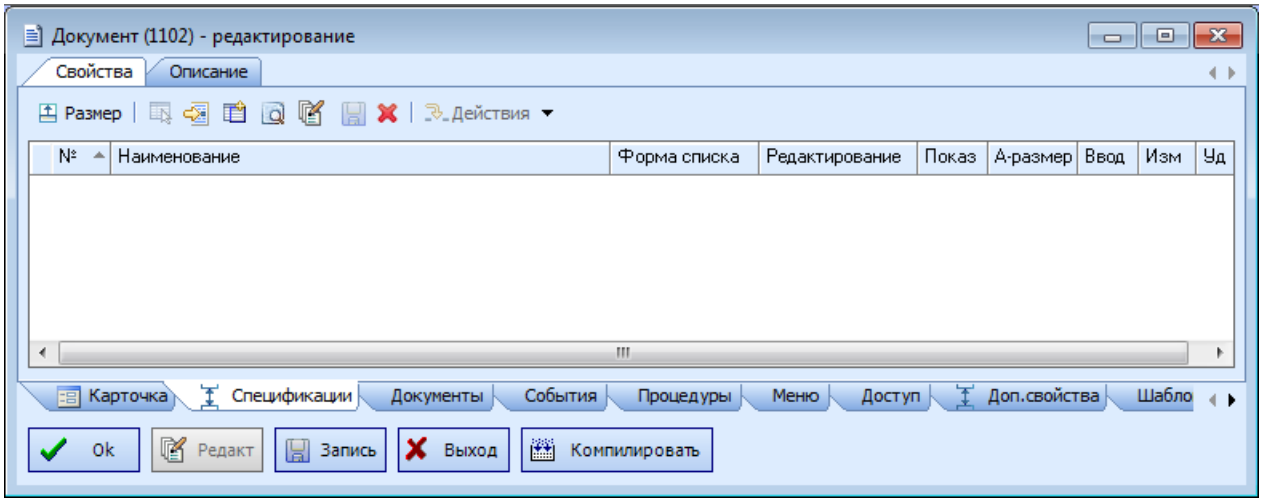
Сохраним данную в форму в библиотеке форм модуля под именем **c_project**. Теперь можно записать идентификатор новой формы (**c_project**) в поле «Форма карточки» в окне описания объекта «Проект» в конфигурации модуля.

Нам необходимо теперь позаботиться о выполнении следующих двух условий в поставленной задаче:

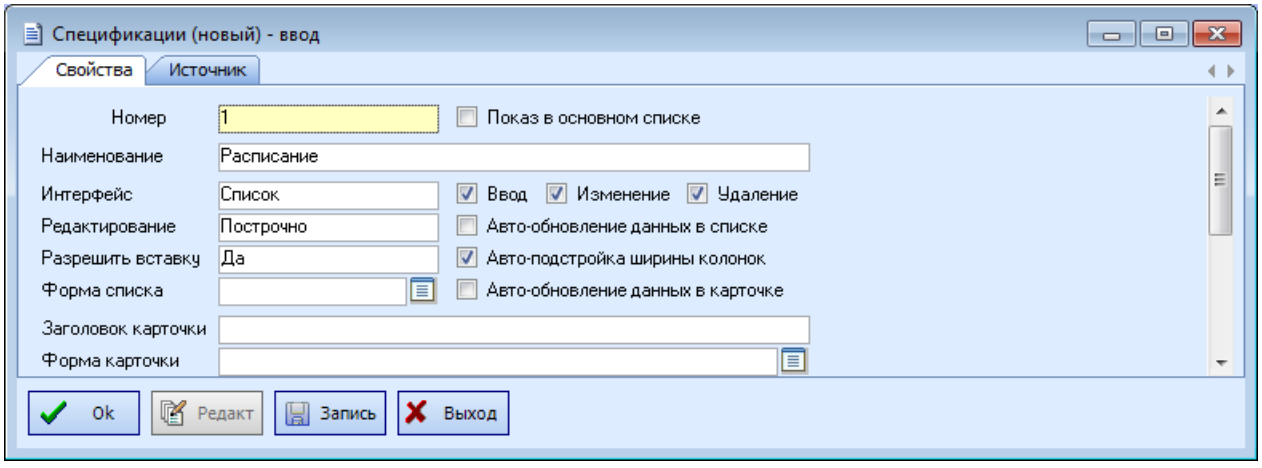
- Проект должен содержать план работ
- Проект должен использовать в качестве опорных пунктов плана уже имеющуюся в конфигурации приложения сущность «задача»

Чтобы выполнить эти требования, нужно использовать дополнительные таблицы и привязать их к сущности «Проект». В качестве плана работ будем использовать созданную нами новую таблицу TASKLIST. А в качестве опорных пунктов плана (задач) – возьмем имеющуюся уже таблицу TASKITEM.

Итак, чтобы «привязать» подчиненные таблицы к нашему документу, будем использовать такое понятие в определении объекта конфигурации, как «спецификация». Спецификация позволяет описать сколько угодно подчиненных сущностей объекта (и указать формы для их просмотра и редактирования), при этом при открытии карточки объекта в среде выполнения все его спецификации будут таже доступны для просмотра и редактирования, как подчиненные таблицы. Для определения спецификаций созданного нами объекта «Проект» перейдем на закладку «Спецификация» в окне описания объекта в конфигурации модуля:



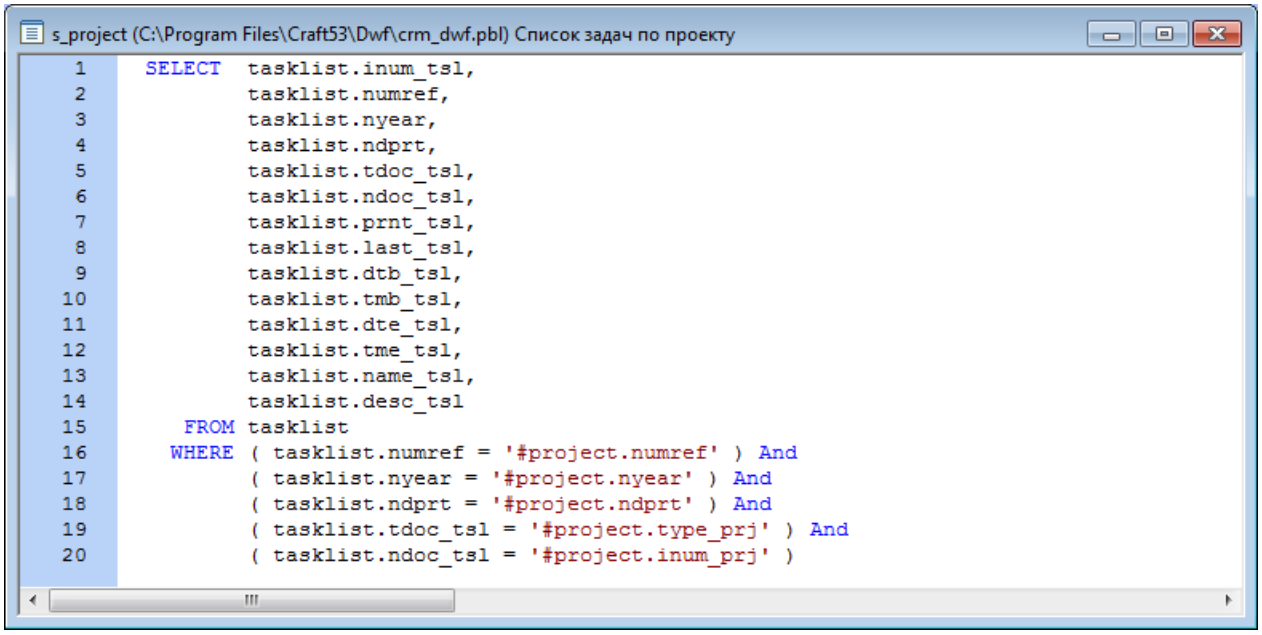
Далее при помощи контекстного меню или нажатием на кнопку  «Ввести» добавим новый элемент в спецификацию объекта:



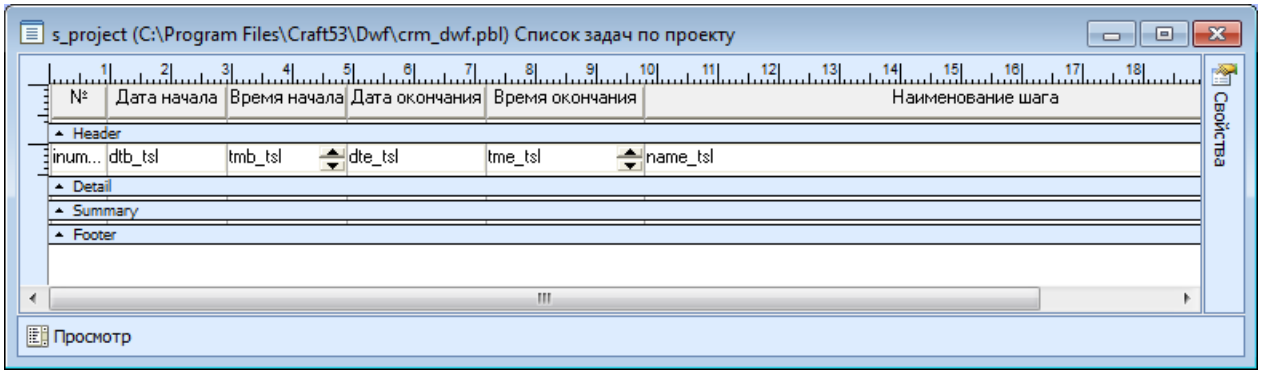
В этом окне установим основные свойства для элемента спецификации:

- Наименование: «Расписание»
- Интерфейс: Список
- Редактирование: Построчно
- Разрешить вставку: Да

Далее, создадим форму списка для этого элемента. В окне редактора форм введем SQL-запрос:



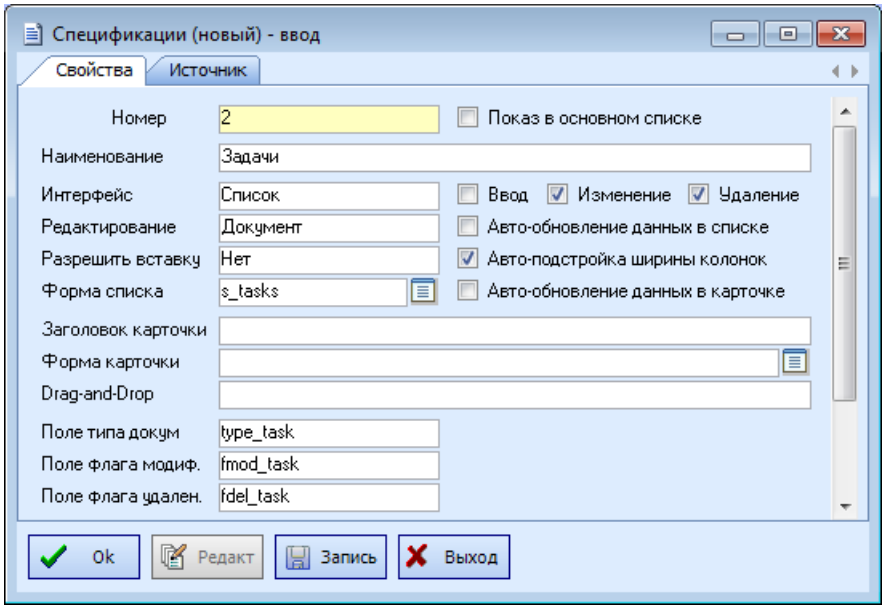
Результирующая форма (сохраним ее в библиотеке под именем **s_project**) должна выглядеть так:



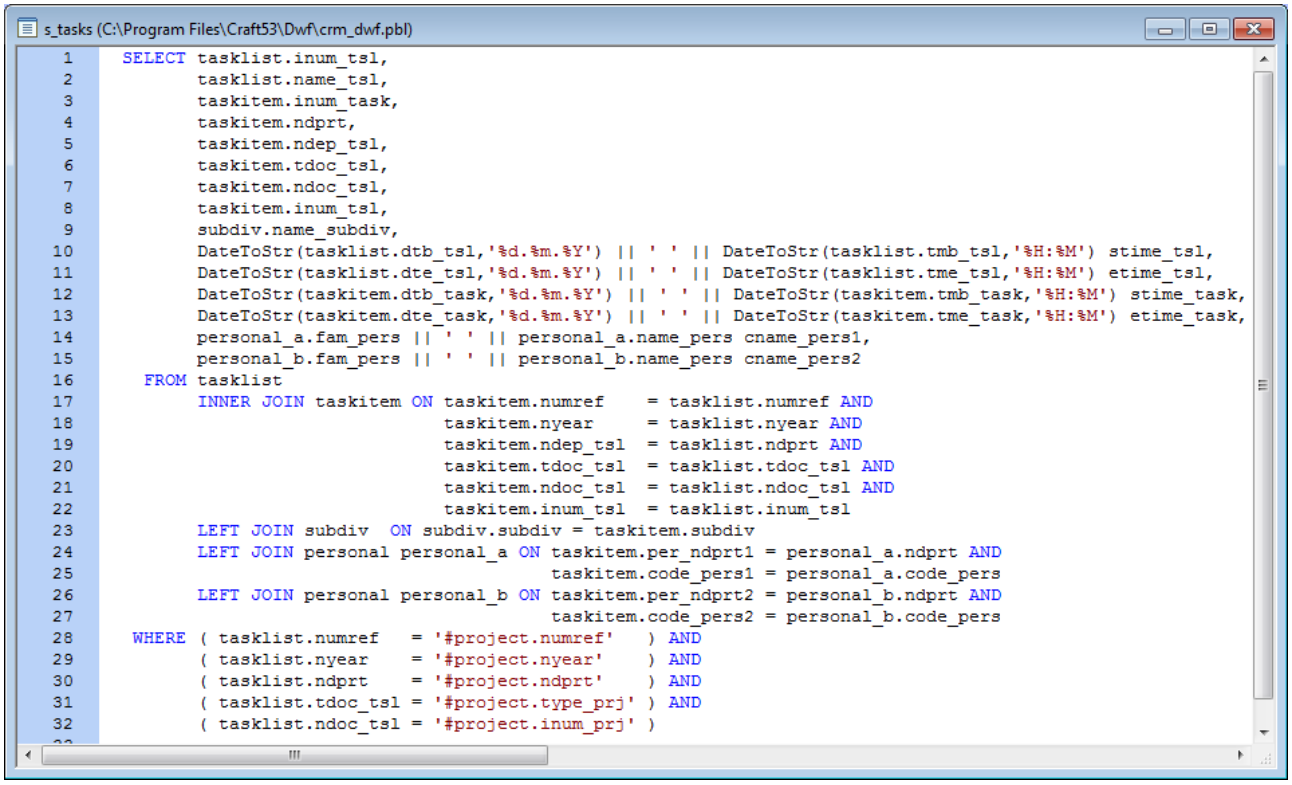
Укажем идентификатор формы (**s_project**) в описании элемента спецификации объекта конфигурации в качестве формы списка.

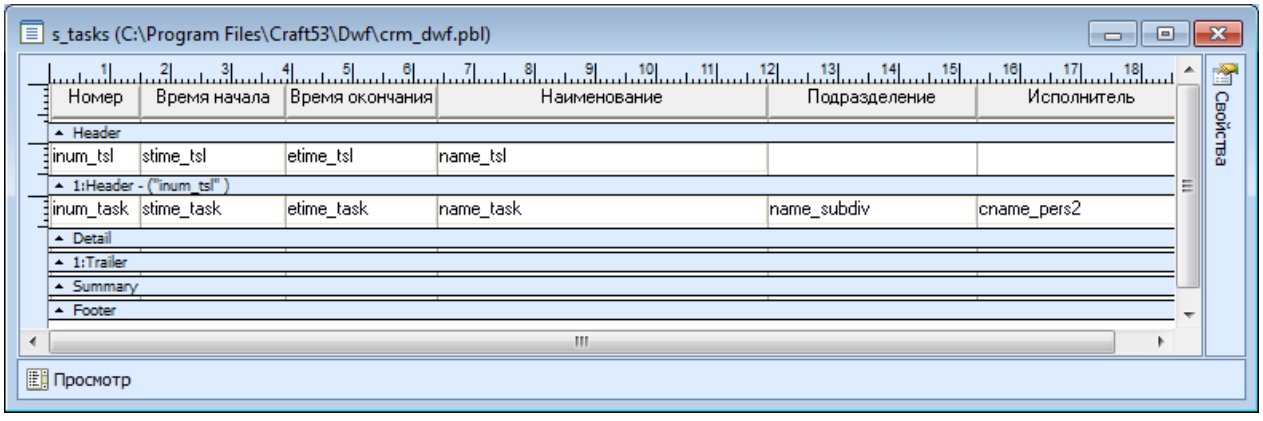
Вторым элементом спецификации должен стать непосредственно список задач по плану. Как уже говорилось, в качестве опорного пункта плана должна быть использована уже имеющаяся в конфигурации сущность «Задача», которая описана в таблице TASKITEM. Нам необходимо создать специальную форму для просмотра задач с привязкой к плану и указать ее как второй элемент спецификации в объекте «Проект». Необходимо также учесть, что «Задача» уже описана в конфигурации модуля, как объект типа «Документ», поэтому нам не нужно будет создавать форму для карточки «задачи» в спецификации и вообще – заботиться об управлении логикой работы данного объекта. Все что нам необходимо сделать – это указать в элементе спецификации стиль редактирования: «Документ». Для того, чтобы система смогла идентифицировать данные в спецификации как документ конфигурации, необходимо явно указать поле в таблице, которое содержит значение кода объекта (документа) конфигурации. Таким образом, при открытии данного объекта для просмотра или редактирования система будет знать, с каким объектом конфигурации его нужно идентифицировать и, соответственно, как его представлять и какую бизнес-логику он должен реализовывать.

Сказанное иллюстрирует картинка ниже:



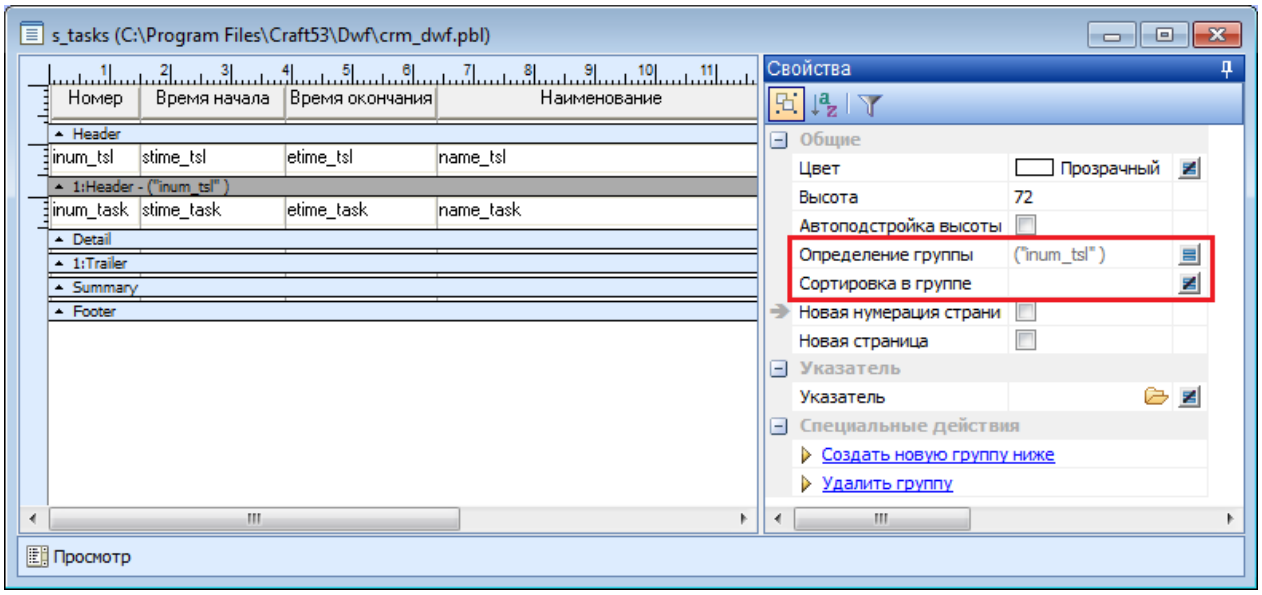
В завершение определения элемента спецификации необходимо создать форму для списка – **s_tasks**. SQL-запрос для формы и ее внешний вид соответственно показаны на картинках ниже:





Заметим, что для этой формы данные организованы в виде группированного списка так, что в заголовке группы представлены этапы плана из таблицы TASKLIST, а внутри группы показаны все

задачи из таблицы TASKITEM, относящиеся к данному этапу. На рисунке ниже показано определение свойств для группы (группировка данных по значению поля **inum_tsl**):



На этом описание основных свойств объекта конфигурации «Проект» и его визуальных форм для работы с данными можно считать законченным.

5. Определение бизнес-логики объектов в конфигурации модуля

В предыдущем разделе было описано, как создать новый объект в конфигурации модуля, каким образом сконструировать формы для ввода и просмотра данных по объекту. Этого уже обычно бывает достаточно для простой манипуляции с данными в случае, например, простых справочников. В нашем же случае созданный объект является документом и предполагает несколько более сложную обработку данных. Для того, чтобы созданный бизнес-объект смог выполнять возложенную на него задачу, необходимо добавить к описанию объекта процедуры для реализации некоторой бизнес-логики.

Процедуры для определения бизнес-логики на платформе КРАФТ пишутся на встроенном языке программирования системы K-Script. Язык K-Script является проблемно-ориентированным языком с поддержкой объектов и SQL. Синтаксис языка совместим с языком PowerScript среды разработки PowerBuilder фирмы Sybase, поэтому те программисты, которые уже знакомы с PowerBuilder, смогут легко начать программировать на языке платформы КРАФТ. Язык K-Script содержит более 500 различных встроенных функций, встроенную поддержку OLE и XML, строгий контроль типов при компиляции. Отличительной чертой языка программирования K-Script является также пре-компиляция процедур бизнес-логики, что обеспечивает высокую надежность и скорость выполнения процедур прикладного модуля (в отличие от подобных систем с интерпретируемым языком), а также исключение всех ошибок уже на этапе компиляции. В среде разработки платформы КРАФТ имеется удобный редактор для процедур на языке К-скрипт с синтаксической подсветкой, в котором можно сразу же скомпилировать процедуру и даже вызвать ее на выполнение.

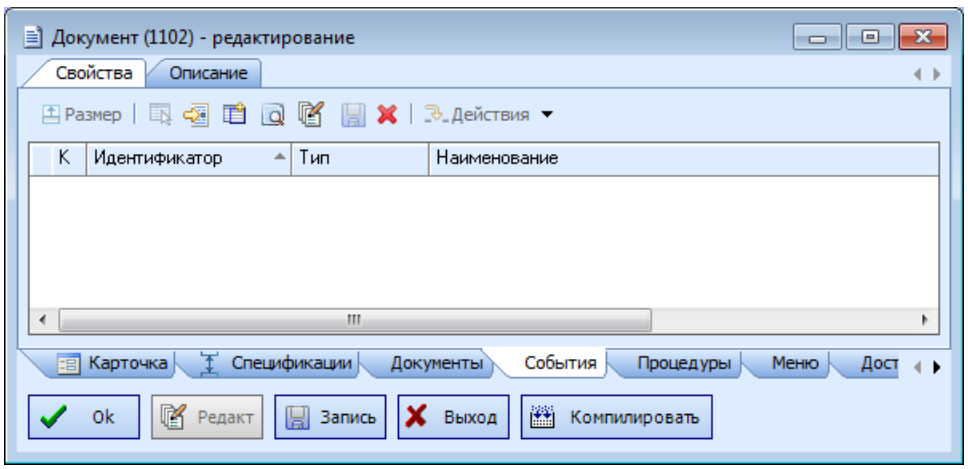
Итак, каким образом реализуется бизнес-логика объектов конфигурации в платформе КРАФТ? Это может быть сделано двумя основными способами: с помощью процедур обработки событий, ассоциированных с объектом, или с помощью процедур, ассоциированных с контекстным меню объекта. Разница между ними очевидна: в первом случае выполнение процедуры, ассоциированной с событием, инициируется самой системой в зависимости от наступления определенных условий. Во втором случае выполнение процедуры инициируется пользователем при вызове пункта контекстного меню (или при нажатии на определенную кнопку в форме или панели инструментов окна объекта).

Начнем с событий. Приведем список поддерживаемых событий объектов данных платформы КРАФТ:

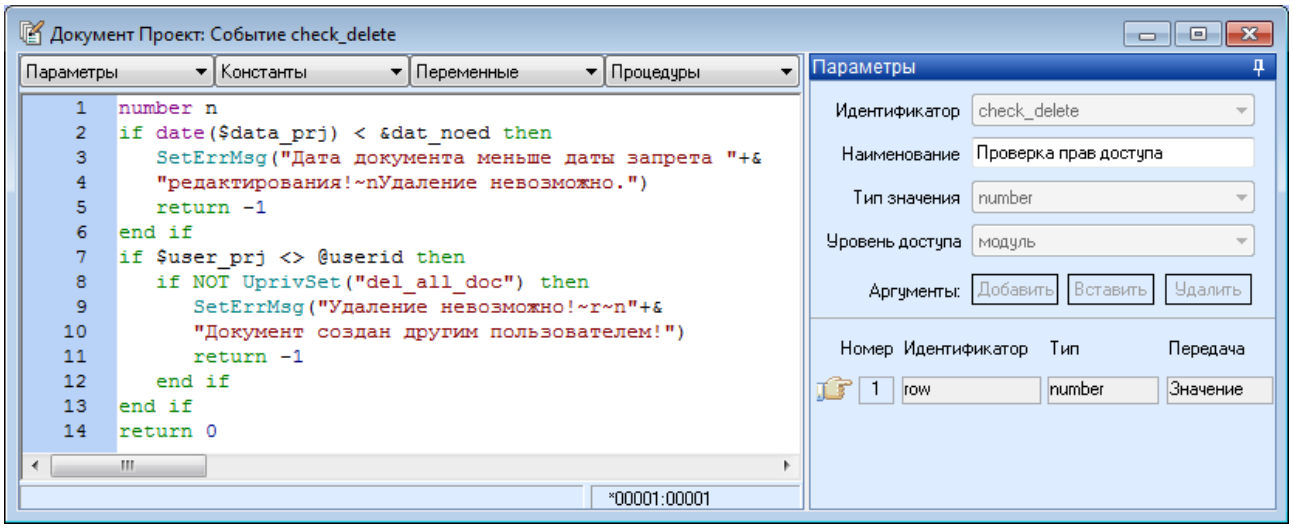
Событие	Описание
CheckDelete	При инициировании операции удаления данных
CheckEdit	При инициировании операции редактирования данных
CheckInsert	При инициировании операции ввода данных
Itemchanged	При изменении поля данных во время редактирования
OnBack	При отмене ”проведения” документа
OnBackEnd	При завершении отмены ”проведения” документа
OnChoose	Перед вызовом окна для отбора данных
OnClose	При закрытии MDI-окна данных
OnCtrlEnter	При нажатии на клавиши Ctrl+Enter
OnDblClick	При двойном щелчке левой кнопкой мыши в объекте данных
OnDdlbClick	При щелчке левой кнопкой мыши на кнопку выпадающего списка
OnDelete	После удаления строки в объекте данных
OnEdit	После перехода в режим редактирования
OnFindTree	При вызове функции поиска по дереву в объекте данных
OnInsert	После ввода данных и перехода в редактирование
OnOpen	При открытии MDI-окна данных
OnPrint	При вызове функции печати в карточке объекта данных
OnReset	При сбросе флага проведения документа
OnRun	При ”проведении” документа
OnRunEnd	При завершении ”проведения” документа
OnSet	При установке флага ”проведения” документа
OnTimer	При срабатывании установленного таймера в окне данных
OnWinSet	При установке значения из диалогового окна выбора
PostDelete	После операции удаления строки данных в БД
PostInsert	После операции записи новой строки данных в БД
PostUpdate	После операции записи измененной строки данных в БД
PreDelete	Перед операцией удаления строки данных в БД
PreInsert	Перед операцией записи новой строки данных в БД
PreUpdate	Перед операцией записи измененной строки данных в БД
RetrieveEnd	По окончании операции чтения в объекте данных
RetrieveStart	Перед началом операции чтения в объекте данных
RowChanged	После изменения текущей строки в объекте данных
RowChanging	Перед изменением текущей строки в объекте данных
UpdateEnd	После завершения общей транзакции изменения данных
UpdateStart	Перед стартом общей транзакции изменения данных

Для нашего объекта «Проект» нам понадобятся обработчики как минимум для двух событий: UpdateStart и CheckDelete.

Чтобы ввести код для обработки события объекта, необходимо переключиться на закладку «События» в окне описания объекта конфигурации.

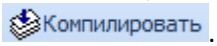


Далее при помощи контекстного меню или нажатием на кнопку  «Ввести» добавим новый элемент в список событий объекта – откроется окно редактора процедур. Справа в панели параметров процедуры выберем идентификатор события в выпадающем списке (check_delete) и в окне редактора введем код обработки события, как показано ниже:

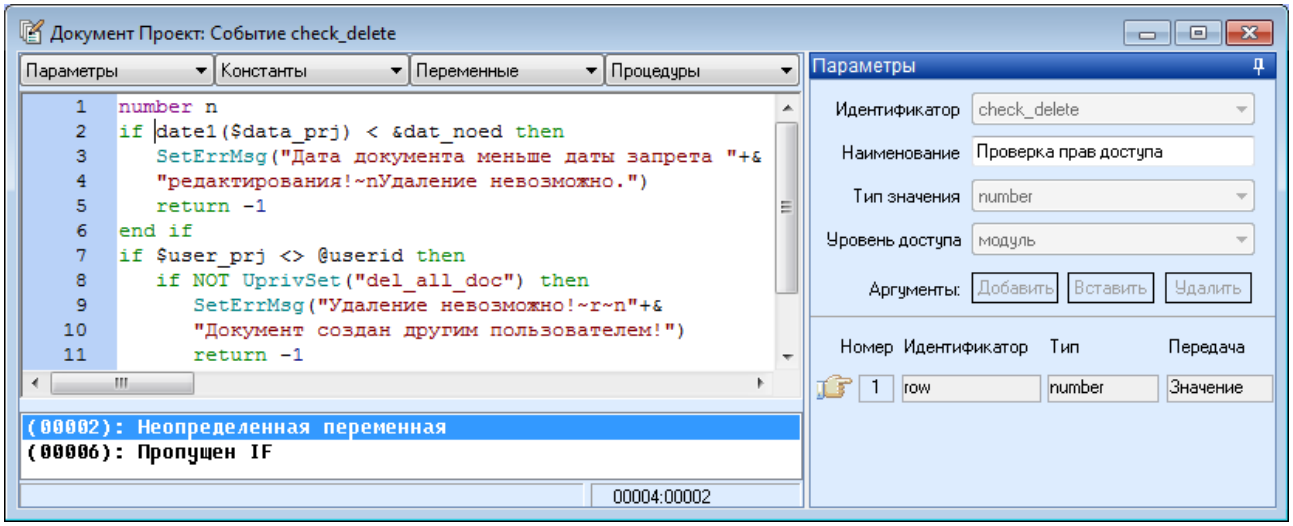


Обратим внимание, что в панели параметров процедуры идентификатор события, тип возвращаемого значения и набор аргументов – недоступны для редактирования. Это происходит из-за того, что идентификаторы событий являются внутренними зарезервированными словами для системы КРАФТ и изменять их нельзя, равно как и набор аргументов для события.

Прокомментируем код процедуры. В первом операторе IF сравнивается текущее значение поля **data_prj** (дата ввода документа) из формы ввода для объекта (для обозначения поля формы используется префикс $ перед идентификатором поля) и значение глобальной константы модуля – **dat_noed** (чтобы указать, что происходит обращение к глобальной константе по ее идентификатору, используется префикс &). Функция SetErrMsg() устанавливает текст сообщения об ошибке, который будет автоматически выведен на экран. Во втором операторе IF сравнивается текущее значение поля **user_prj** (пользователь, создавший документ) из формы ввода для объекта и значение глобальной переменной модуля – **userid** (чтобы указать, что происходит обращение к глобальной переменной по ее идентификатору, используется префикс @). Далее при помощи функции UprivSet() проверяется наличие привилегии у текущего пользователя на удаление чужих документов. В случае отсутствия привилегии устанавливается текст сообщения об ошибке и событие возвращает код ошибки.

После того, как код введен, можно его скомпилировать при помощи пункта меню редактора 

. Если при компилиции найдены ошибки, то они выводятся на экран (при перемещении по списку ошибок курсор ввода позиционируется на строке с ошибкой в тексте):



Теперь введем код обработки для второго события нашего объекта – UpdateStart:

```
number n
if $nyear <> year(date($data_prj)) then
        SetErrMsg("Текущий год не соответствует дате документа!")
        return -1
end if
choose case winobject.State
case NewModified!
        if date($data_prj) < &dat_noed then
                SetErrMsg("Дата документа меньше даты запрета редактирования!~nВвод невозможен.")
                return -1
        end if
        if NOT UprivSet("indprt_doc") then
                if $ndprt > &cndep then
                        SetErrMsg("Ввод невозможен!~r~n"+&
                        "Отделение документа не соответствует текущему отделению!")
                        return -1
                end if
        end if
case DataModified!
        if date(!data_prj) < &dat_noed then
                SetErrMsg("Дата документа меньше даты запрета редактирования!~nИзменение невозможно.")
                return -1
        end if
        if $user_prj <> @userid then
                if NOT UprivSet("edt_all_doc") then
                        SetErrMsg("Изменение невозможно!~r~n"+&
                        "Документ создан другим пользователем!")
                        return -1
                end if
        end if
end choose
return 0
```

Краткие комментарии по коду обработки события. Сначала проверяется соответствие значения поля **nyear** (год) из формы ввода значению, вычисленному из поля **date_prj** в форме. Мы помним, что поле **nyear** инициализируется в форме из глобальной переменной при помощи специального тега: VAL{cur_year}. Для вычисления года из поля **data_prj** значение сначала преобразуется формат даты при помощи встроенной функции date(), а затем из даты выделяется год с помощью встроенной функции year().

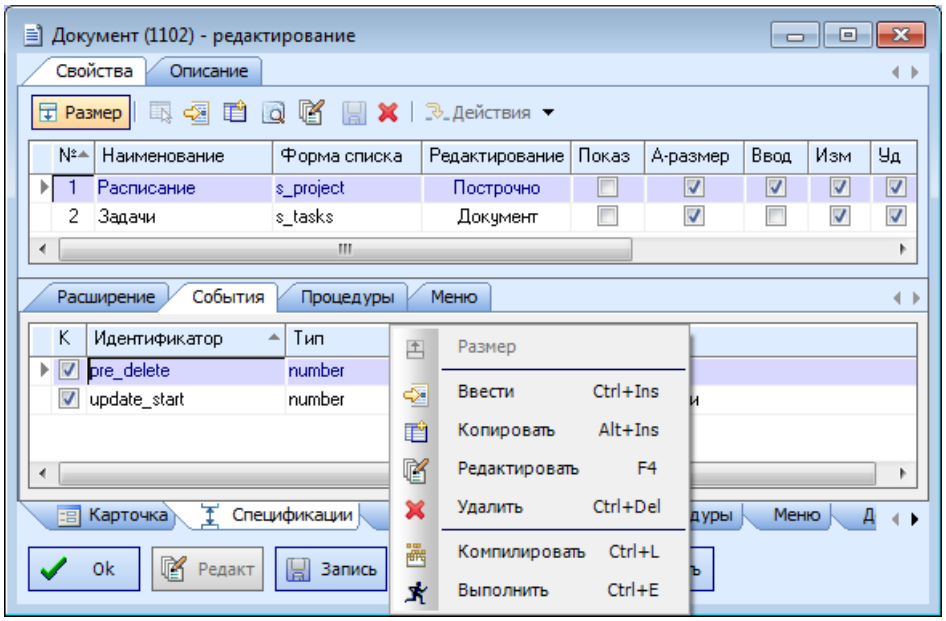
Далее в коде проверяется значение свойства State объекта Winobject. Winobject является зарезервированным словом языка K-Script и обозначает ссылку на текущий объект конфигурации (его главное окно). Свойство State может принимать ряд зарезервированных значений, которые используются для проверки в операторе CASE:

- NewModified! – окно в режиме ввода нового объекта, данные изменились
- DataModified! – окно в режиме редактирования объекта, данные изменились
- NotModified! – окно в режиме редактирования или просмотра объекта, данные не изменились

Далее в коде делается ряд проверок, очевидных по смыслу, и которые не нуждаются в комментариях. После компиляции кода данного события редактор процедур можно закрыть.

Для объекта «Проект» мы не будем вводить процедуры, ассоциированные с меню для главной формы объекта.

Теперь перейдем к бизнес-логике на уровне спецификации объекта. Для начала определим код обработки событий для спецификации «Расписание». Чтобы ввести новый обработчик события на уровне спецификации объекта, необходимо сначала перейти на закладку «Спецификация» в окне описания объекта конфигурации. Далее, нужно выделить требуемую спецификацию (для которой предполагается добавление обработчика событий) и нажать на кнопку «Размер» в панели кнопок спецификации – в окне описания объекта откроется еще одна панель внизу с закладками «Расширение», «События», «Процедуры», «Меню». Для ввода нового обработчика событий для спецификации «Расписание» нужно перейти на закладку «События» в этой подчиненной панели.



Затем при помощи контекстного меню панели вызвать функцию ввода нового элемента – откроется окно редактора процедур. Приведем текст обработчиков событий для спецификации «Расписание».

Событие PreDelete (проверка при удалении):

```
number ncount
// Проверка наличия задач по шагу расписания
select count(inum_task) into :ncount from taskitem
where numref = !numref and nyear = !nyear and
ndep_tsl = !ndprt and tdoc_tsl = !tdoc_tsl and
ndoc_tsl = !ndoc_tsl and inum_tsl = !inum_tsl;
if ncount > 0 then
        SetErrMsg("Невозможно удалить шаг расписания!"+&
                        "~r~nПо нему уже есть назначенные задачи.")
        return -1
end if
return 0
```

Событие PreDelete вызывается непосредственно перед транзакцией удаления данных в форме. Если обработчик события возвращает значение меньше нуля, то транзакция прерывается и данные не удаляются.

В коде обработчика данного события проверяется, существуют ли задачи, привязанные к удаляемому этапу плана. Для этого используется SQL-запрос в БД к таблице TASKITEM. В параметрах запроса используются значения из полей формы. Обратим внимание, что идентификаторы полей формы в коде имеют префикс «!», что означает (в оличие от префикса $) использование оригинальных считанных из БД значений данных полей, даже если впоследствии они подверглись изменению в форме. Значение, возвращаемое SQL-запросом, присваивается переменной **ncount** с помощью выражения INTO в SQL-запросе.

Приведем также код обработчика события UpdateStart. Цель написания данного обработчика состоит в том, чтобы проверить корректность вводимых дат для шагов расписания в плане.

Во-первых, дата окончания шага не может быть меньше даты начала шага. Во-вторых, период дат шага не должен пересекаться с другими уже введенными шагами по плану. Для проверки этого условия выполняется SQL-запрос к таблице TASKLIST. Наконец, в случае если даты или время периода шага изменились (при редактировании), проверяется наличие уже введенных задач по данному шагу. Для проверки этого условия также выполняется SQL-запрос, но уже к таблице TASKITEM.

Заметим, что функция DateToDouble(), которая используется в обоих SQL-запросах в обработчике данного события – это встроенная функция сервера БД Firebird и необходима для сравнения дат.

Событие UpdateStart (проверка при вводе и изменении):

```
number ncount
datetime dtb,dte
// Период текущего шага
dtb = DateTime($dtb_tsl,$tmb_tsl)
dte = DateTime($dte_tsl,$tme_tsl)
if DaysAfter($dtb_tsl, $dte_tsl) < 0 then
        SetErrMsg("Дата окончания периода не может быть меньше даты начала периода!")
        return -1
end if
// Проверка пересечения периода с другими шагами
select count(inum_tsl) into :ncount from tasklist
where numref = $numref and nyear = $nyear and
ndprt = $ndprt and tdoc_tsl = $tdoc_tsl and
ndoc_tsl = $ndoc_tsl and inum_tsl <> $inum_tsl and
(DateToDouble(dtb_tsl + tmb_tsl) < DateToDouble(:dtb) and
 DateToDouble(dte_tsl + tme_tsl) > DateToDouble(:dtb) or
 DateToDouble(dtb_tsl + tmb_tsl) < DateToDouble(:dte) and
 DateToDouble(dte_tsl + tme_tsl) > DateToDouble(:dte));
if ncount > 0 then
        SetErrMsg("Период шага пересекается с другими шагами расписания по документу!")
        return -1
end if
// Если изменился периода шага расписания
if $dtb_tsl <> !dtb_tsl or $tmb_tsl <> !tmb_tsl or &
        $dte_tsl <> !dte_tsl or $tme_tsl <> !tme_tsl then
        // Проверка наличия задач по шагу расписания
        // с периодом большим шага расписания
        select count(inum_task) into :ncount from taskitem
        where numref = $numref and nyear = $nyear and
        ndep_tsl = $ndprt and tdoc_tsl = $tdoc_tsl and
        ndoc_tsl = $ndoc_tsl and inum_tsl = $inum_tsl and
        (DateToDouble(dtb_task + tmb_task) < DateToDouble(:dtb) or
         DateToDouble(dte_task + tme_task) > DateToDouble(:dte));
        if ncount > 0 then
                SetErrMsg("Невозможно изменить период шага расписания!"+&
                        "~r~nПо данному периоду уже есть назначенные задачи.")
                return -1
        end if
end if
return 0
```

Наконец, чтобы завершить определение бизнес-логики для нашего объекта «Проект», введем процедуру для спецификации «Задачи». Данная процедура будет вызываться из контекстного меню, ассоциированного с формой списка данной спецификации (о чем будет рассказано далее).

Кратко поясним, зачем понадобилась специальная процедура для ввода задачи. Мы помним из предыдущего раздела (Создание объектов в конфигурации модуля), что «задача» - это уже существующий объект конфигурации типа «документ». Это в частности означает, что все поведение объекта «задача» уже описано в конфигурации и можно не беспокоиться об организации ввода и редактировании данных – система сделает это сама. Однако, в объекте «Проект» мы хотим предоставить пользователю дополнительный сервис по автоматизации ввода задач в план (чтобы не нужно было даже открывать окно для ввода задачи и вводить туда какие то данные), тем более что все необходимые параметры для ввода задачи задачи у нас уже есть в шаге расписания. Вторая преследуемая цель – максимально исключить возможные ошибки ввода данных пользователем при привязке задач к плану. Поэтому, мы выключили флажок «Ввод» в определении спецификации и установили свойство «Разрешить вставку» в значение «Нет». Таким образом, ввод новых задач в расписание будет доступен только через специальную процедуру (но не модификация или удаление).

Для ввода процедуры для спецификации «Задачи» выделим ее в списке в панели спецификаций в окне описания объекта конфигурации и переключимся на закладку «Процедуры» в подчиненной панели внизу. Затем при помощи контекстного меню панели вызовем функцию ввода нового элемента – откроется окно редактора процедур. Приведем ниже текст процедуры **create_task** (у нее нет параметров):

```
datetime ctime
string title,errtext,sqlw,sinum,sname,sdtb,sdte,stmb,stme
number dnref,dyear,dndep,dtype,dnkey,nlist,ttask,ntask,nrow
if winobject.State = NewModified! then return -1
if @uperscode = 0 then
        errtext = "Текущий пользователь не ассоциирован с сотрудником!"+&
                                "~r~nНевозможно назначить автора задачи."
        goto error
end if
ctime   = DateTime(today(),now())
title   = "Ввод задачи"
ttask   = 1101
// Ключ документа
dnref = #numref
dyear = #nyear
dndep = #ndprt
dtype = #type_prj
dnkey = #inum_prj
// Текущий номер шага
nrow    = dwo.GetRow()
if nrow > 0 then sinum = string(dwo.GetItemNumber(nrow,"inum_tsl"))
// Выбор шага расписания для ввода задачи
if OpenLstDlg(List!,sinum,"inum_tsl","Выбор шага расписания","dw_tasklist_spr",&
        "numref="+string(dnref)+" and nyear="+string(dyear)+" and ndprt="+string(dndep)+&
        " and tdoc_tsl="+string(dtype)+" and ndoc_tsl="+string(dnkey)) < 0 then return -1
if sinum = "" then return -1
// Чтение параметров шага
nlist = number(sinum)
sname   = GetDragVal(1, "tasklist.name_tsl")
sdtb    = GetDragVal(1, "tasklist.dtb_tsl")
sdte    = GetDragVal(1, "tasklist.dte_tsl")
stmb    = GetDragVal(1, "tasklist.tmb_tsl")
stme    = GetDragVal(1, "tasklist.tme_tsl")
// Ввод новой задачи
select coalesce(max(inum_task),0) + 1 into :ntask
from taskitem where type_task = 1101 and ndprt = :dndep;
insert into taskitem (ndprt,type_task,inum_task,code_tvar,code_tdiv,subdiv,
cli_ndprt,numb_client,per_ndprt1,code_pers1,per_ndprt2,code_pers2,npri_task,stat_task,
fcst_task,trep_task,nday_task,nwks_task,nmon_task,rmon_task,rtue_task,rwed_task,
rthu_task,rfri_task,rsat_task,rsan_task,remn_task,remv_task,fdis_task,tsc_task,
tsm_task,dtb_task,tmb_task,dte_task,tme_task,npct_task,auth_task,umod_task,name_task,
desc_task,numref,nyear,ndep_tsl,tdoc_tsl,ndoc_tsl,inum_tsl,fmod_task,fdel_task)
values (:dndep,:ttask,:ntask,1,0,0,0,0,@upersndep,@uperscode,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,:ctime,:ctime,:sdtb,:stmb,:sdte,:stme,0,@userid,'',
:sname,'',:dnref,:dyear,:dndep,:dtype,:dnkey,:nlist,0,0);
if SQLCA.SQLCode <> 0 then
        errtext = "Ошибка записи задачи в БД:~r~n" + SQLCA.SQLErrText
        rollback using SQLCA;
        goto error
else
        commit using SQLCA;
end if
// Запись отметки о создании задачи в истории изменений
insert into taskhist (ndprt, type_task, inum_task, inum_hist, per_ndprt, code_pers,
nact_hist, stat_hist, dact_hist, user_hist, cont_hist)
values (:dndep, :ttask, :ntask, 1, @upersndep, @uperscode,
1, 0, :ctime, @userid, 'Создана задача');
if SQLCA.SQLCode <> 0 then
        rollback using SQLCA;
else
        commit using SQLCA;
end if
// Чтение данных
winobject.WndTrigger("retrieve_spc")
// Поиск задачи
nrow = dwo.Find("inum_task="+string(ntask),1,dwo.RowCount())
// Переход на строку задачи
if nrow > 0 then dwo.SetRow(nrow)
return 0

error:
MessageBox(title,errtext)
return -1
```

Так как текст процедуры достаточно сложный, потребуются некоторые пояснения.

Итак, в самом начале проверяется состояние окна основного объекта. Если окно находится в режиме ввода, то процедура завершается с кодом ошибки. Это сделано для того, чтобы было нельзя вводить данные в подчиненную таблицу до тех пор, пока не записаны данные в таблицу PROJECT.

Далее проверяется, что сотрудник был автоматически определен при входе в систему по его логину (так как код сотрудника в справочнике персонала предприятия понадобится при вводе задачи).

Далее в локальные переменные скрипта считываются значения ключевых полей из основной таблицы объекта – PROJECT. Обратим внимание, что в этом случае используется префикс #, так как данная процедура ассоциирована не с основной формой объекта, а ее спецификацией (подчиненной формой). Для доступа к значениям полей основной формы из подчиненной используется префикс #.

Определяется текущий номер шага в форме. Для этого используются функции объекта данных dwo. Dwo является зарезервированным словом языка K-Script и обозначает ссылку на объект текущей формы (DataWindow). Объект dwo (DataWindow) является очень мощным компонентом для управления данными и имеет обширный набор функций и свойств. В данном случае используются функция GetRow() для получения номера текущей строки и функция GetItemNumber() для получения значения поля данных из указанной строки.

Затем вызывается диалог для выбора шага расписания при помощи функции OpenLstDlg(). Параметрами данной функции являются, по порядку:

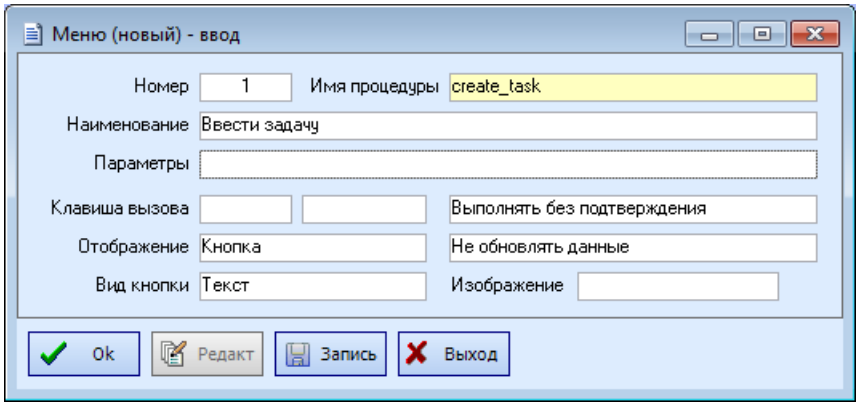
- Интерфейс (List! или TreeView!)
- Имя переменной, в которой будет сохранено выбранное значение
- Наименование поля базы данных колонки DataWindow, из которой будет браться значение
- Текст заголовка
- Идентификатор формы данных (имя DataWindow)
- Дополнительное условие в выражении WHERE для SQL-запроса в форме

Если пользователь в диалоге выбора шага нажал кнопку «Отмена», то процедура завершается. Если «Ок», то в переменной **sinum** оказывается значение выбранного шага. Также, можно получить и другие значения из таблицы выбора в диалоге, при помощи специальной функции GetDragVal(). Таким способом считываются все остальные параметры выбранного шага расписания.

На следующем этапе записываются данные о новой задаче в таблицу TASKITEM, используя SQL оператор INSERT и полученные ранее параметры. Управление транзакцией осуществляется через объект транзакций по умолчанию – SQLCA (хотя на самом деле в коде процедур конфигурации можно создать и использовать сколько угодно объектов типа Transaction). В случае возникновения ошибки при выполнении SQL-оператора транзакция откатывается с помощью команды ROLLBACK. Завершающим этапом ввода задачи является запись информации в подчиненную таблицу истории изменения по задаче – TASKHIST.

В самом конце процедуры обновляется окно объекта (точнее - спецификация) с помощью вызова внутренней функции окна «retrieve_spc» функцией объекта окна WndTrigger(). Затем выполняется поиск только что введенной задачи по ее номеру с помощью функции Find объекта dwo. Если задача найдена, то происходит позиционирование на строку с найденной задачей в форме списка спецификации.

Итак, нам осталось только «привязать» (ассоциировать) только что введенную процедуру спецификации к контекстному меню объекта. Для этого необходимо переключиться на закладку «Меню» в подчиненной панели спецификации внизу. Затем при помощи контекстного меню панели вызвать функцию ввода нового элемента – откроется окно для ввода нового пункта меню. Описанное здесь меню будет затем доступно в панели инструментов спецификации объекта в качестве выпадающего меню кнопки «Действия» или в качестве самостоятельной кнопки в панели инструментов, в зависимости от определения.



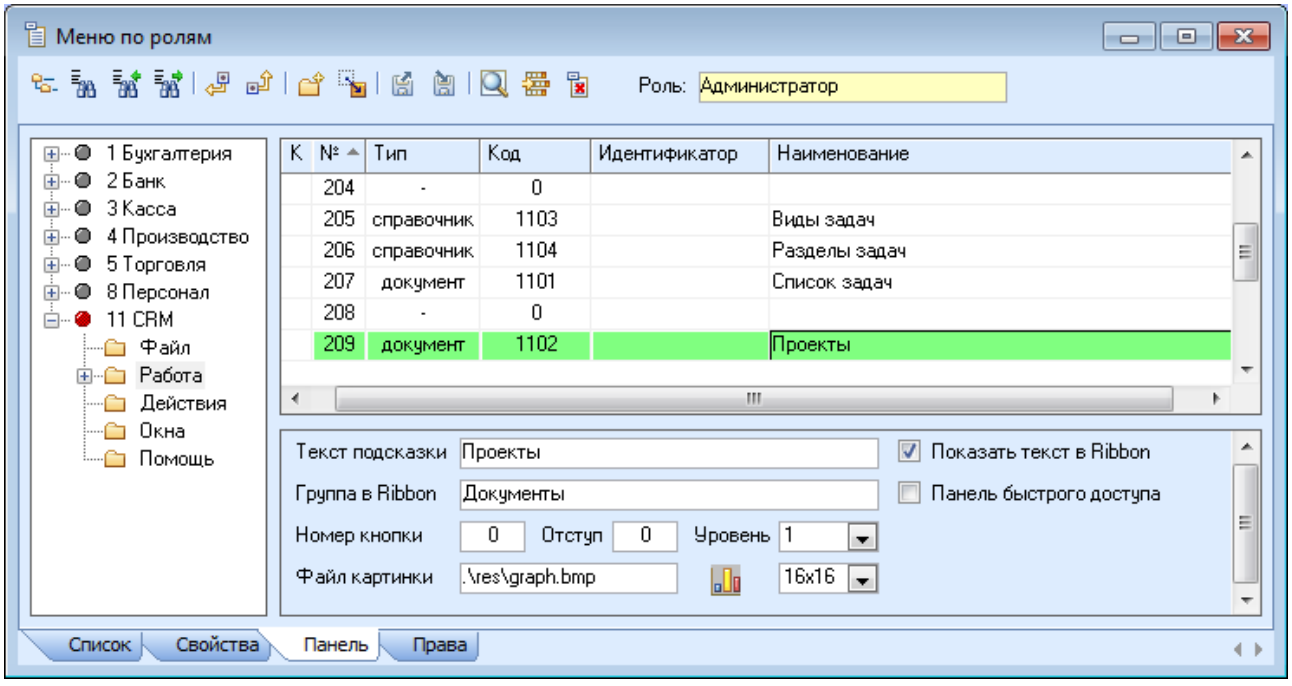
В данном окне выберем имя процедуры в выпадающем списке, назначим ей порядкоый номер в меню, а также выберем вид отображеия – «Кнопка». Таким образом, наша процедура будет вызываться по нажатию отдельной кнопки панели инструментов спецификации.

На этом описание бизнес-логики объекта «Проект» можно считать законченным.

6. Регистрация бизнес-объекта в меню приложения

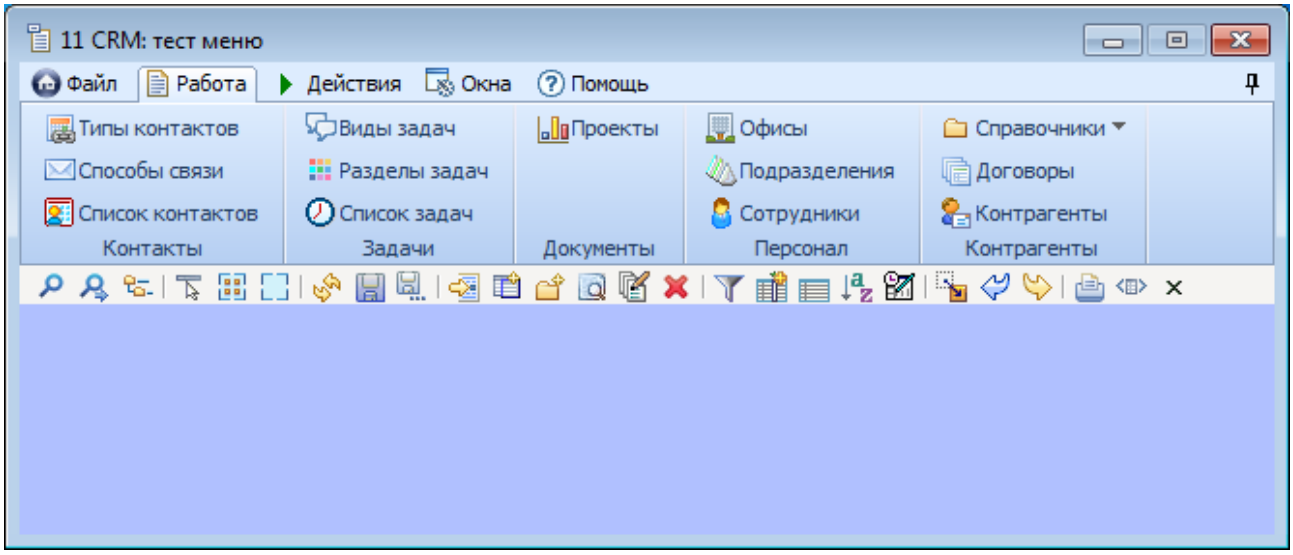
Из всех рассмотренных этапов – это самый простой и приятный, так как является завершающим ☺

Для регистрации нового объекта в меню приложения нужно вызвать окно управления меню приложений с помощью пункта основного меню дизайнера «Меню по ролям». По умолчанию все меню определяются для роли «Администратор» (меню для других ролей можно скопировать из меню Администратора с помощью сервисной функции). В навигаторе объектов выбираем модуль CRM, а в нем – раздел меню «Работа». Из контекстного меню окна или при помощи клавиш Ctr+Ins вызываем функцию добавления нового элемента меню и заполняем так, как показано на картинке ниже:



Для того, чтобы указать для пункта меню созданныый нами объект «Проект», нужно в поле Тип выбрать «документ», а в поле код – указать номер нашего объекта в конфигурации (либо выбрать из выпадающего списка). На закладке «Панель» определяются свойства элемента для показа его в меню приложения. Отметим флажок «Показать текст в Ribbon» и укажем файл картинки для отображения в меню. Записываем изменения. Все!

Теперь можно сразу же протестировать внешний вид полученного меню. Для этого нажимаем на кнопку «Просмотр» в панели инструментов окна управления меню. На экран будет выведено окно с меню, которое сгенерировано по нашему описанию (точно также будет выглядеть меню рабочего приложения):



Ура! Мы видим пункт «Проекты» с соответствующей картинкой в меню приложения!

Итак, нас можно поздравить – мы успешно завершили ввод нового объекта конфигурации в модуль CRM.

7. Тестирование готового приложения

А теперь запустим приложение и убедимся, что новый объект «Проект» работает так, как мы задумали. Выберем пункт «Проекты» в меню приложения и из выведенного окна списка вызовем функцию добавления нового документа. Вот так будет выглядеть созданный нами объект: