**Порядок разработки подключаемых модулей Symphony.Next (плагинов)**

Оглавление

[Требования к компьютеру разработчика. 1](#_Toc447142552)

[Условные обозначения 1](#_Toc447142553)

[Архитектура плагина 1](#_Toc447142554)

[Порядок создания плагинов 2](#_Toc447142555)

[Создание проекта плагина 3](#_Toc447142556)

[Создание фрейма плагина 4](#_Toc447142557)

[Создание экспортируемых функций 5](#_Toc447142558)

[Создание акций 6](#_Toc447142559)

[Создание фрейма настроек плагина 7](#_Toc447142560)

[Создание класса реализующего интерфейс IPlugInPackage 7](#_Toc447142561)

[Функция PackageInterface 8](#_Toc447142562)

[Запуск плагина под отладчиком 8](#_Toc447142563)

## Требования к компьютеру разработчика.

Требования к установленному программному обеспечению и настройкам среды разработки описаны в документе «**Подготовка к разработке.docx**»

## Условные обозначения

$(READY) – каталог с результирующими файлами, где будут собираться создаваемые программные пакеты

$(BIN) – системный каталог, содержащий главное приложение и системные пакеты. Согласно структуры каталогов, системный каталог является подкаталогом $(READY): $(READY)\bin

$(PLUGIN) – каталог плагинов, содержащий все пакеты плагинов приложения. Согласно структуры каталогов, каталог плагинов является подкаталогом $(READY): $(READY)\plugin

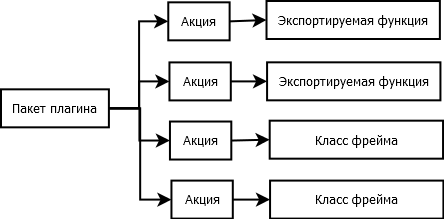
$(BIN64) и $(PLUGIN64) – 64-х битные версии каталогов $(BIN) и $(PLUGIN)

## Архитектура плагина

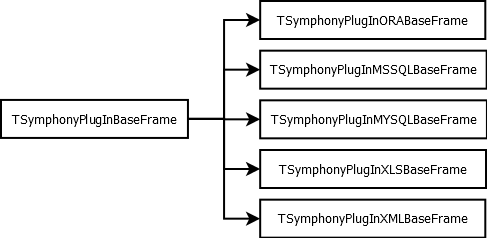
Плагины представляют собой пакеты (библиотеки) созданные Embarcadero Delphi, которые должны отвечать ряду требований. Плагин разрабатывается под определенный тип базы данных. Не допускается создание плагина для различных типов баз данных.

Плагин должен экспортировать функцию PackageInterface, которая должна вернуть интерфейс IPlugInPackage. Данный интерфейс определяет порядок взаимодействия основного приложения и плагина. Описание интерфейса находится в модуле SymphonyPlugIn.PackageInterface, который является частью проекта smplBaseFrame.dproj, входящего в библиотеку PlugIn\_Core (git-репозиторий: /repo/Unicom/Symphony.Next/PlugIn\_Core.git) . С помощью данного интерфейса программа узнает тип используемой базы данных, а также получает список акций (команд), поддерживаемых плагином.

Акция плагина позволяет указать имя экспортируемой плагином функции или имя класса фрейма, который при выполнении акции будет создан и передан основному приложению. Пакет плагина может экспортировать множество функций и классов фреймов. Все функции и классы фреймов должны разрабатываться с учетом одного и того же типа базы данных.



Любой фрейм плагина должен наследоваться от одного из базовых фреймов. Выбор базового фрейма определяется используемым типом баз данных. Все базовые фреймы наследуются от корневого базового фрейма, который задает основное поведение фрейма в рамках центрального приложения:



## Порядок создания плагинов

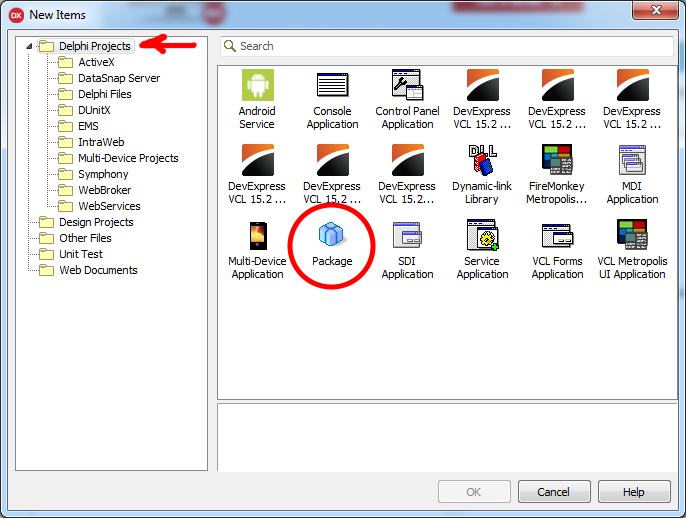
При создании плагинов необходимо сделать ряд действий:

1. Создать проект плагина
2. Выбрать базовый фрейм, исходя из используемого типа базы данных
3. Создать необходимые фреймы и функции, запускаемые основным приложением
4. Экспортировать функции и зарегистрировать классы фреймов
5. Создать акции под каждый фрейм и экспортируемую функцию
6. Создать фрейм управляющий настройками плагина ( при необходимости)
7. Создать класс реализующий интерфейс IPlugInPackage
8. Создать и экспортировать функцию PackageInterface, возвращающую интерфейс IPlugInPackage

### Создание проекта плагина

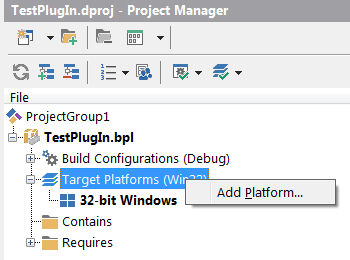
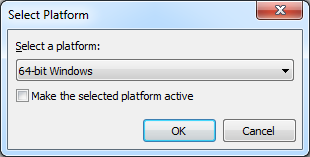
Создание плагина начинается с выбора или создания каталога файловой системы, в которой будет располагаться проект плагина.

После того как определен каталог плагина, необходимо запустить Delphi и создать в нем новый проект пакета (Package):

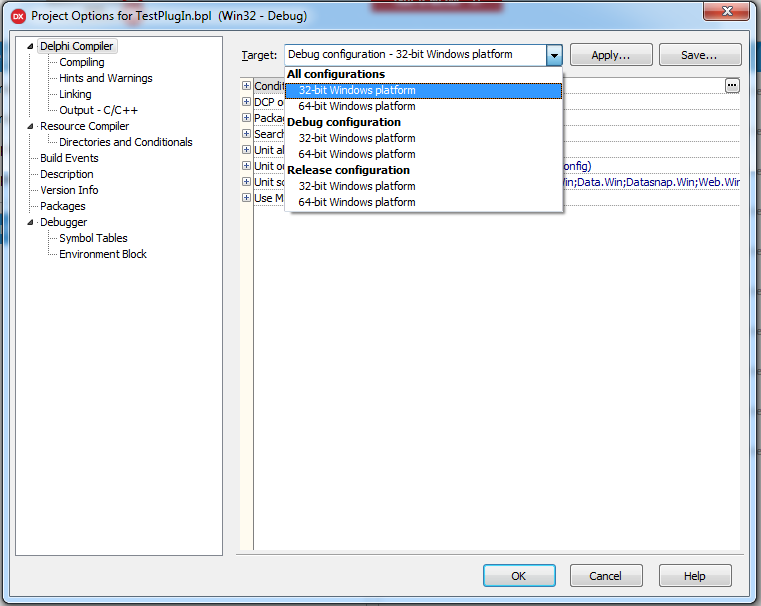


После того как проект создан, его необходимо сначала сохранить в выбранном каталоге, а затем произвести настройки проекта.

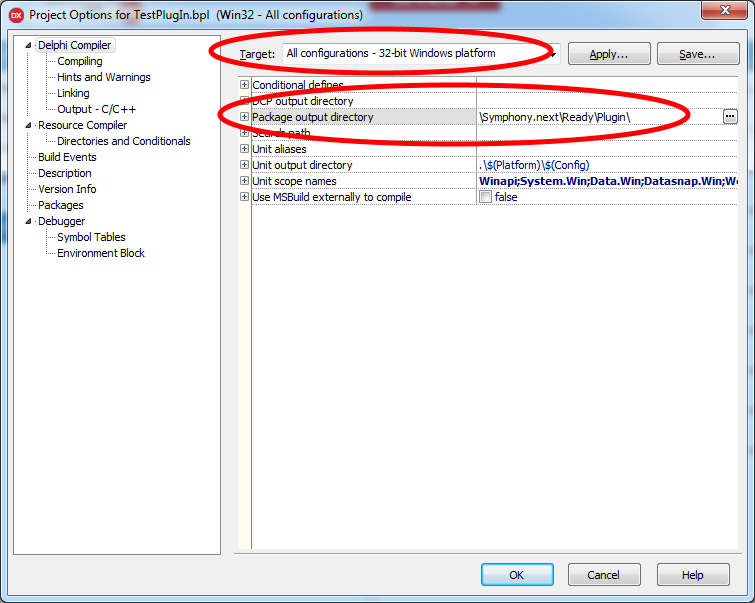
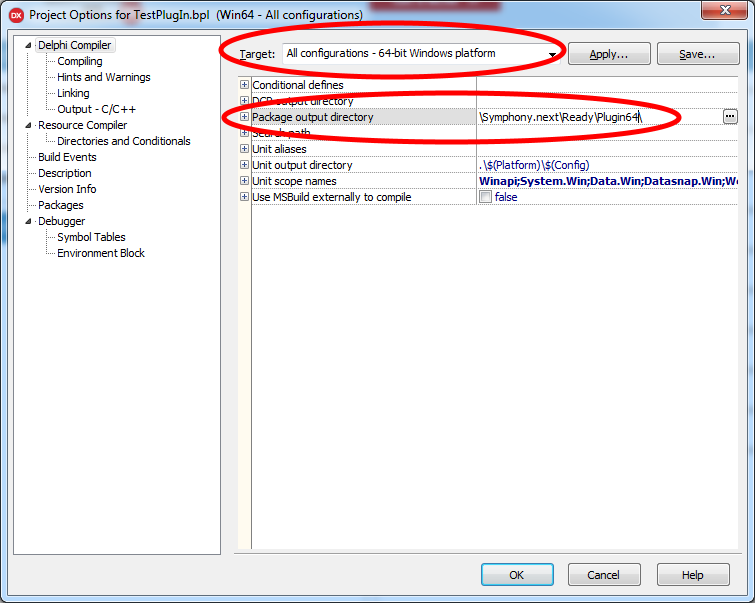
Настройка проекта начинается с определения целевых платформ. По умолчанию, проект создается под одну платформу: Win32. Если планируется выпуск 64-х разрядного плагина, то необходимо добавить в список целевых платформ платформу Win64:

После того, как определены возможные платформы пакета, необходимо открыть диалог «Project Options»:



Для каждой платформы необходимо в разделе «Delphi Compiler» установить параметр «Package output directory» равным пути к каталогу плагинов $(PLUGIN) или $(PLUGIN64) Данный параметр определяет каталог, в который среда Delphi будет помещать готовый пакет.

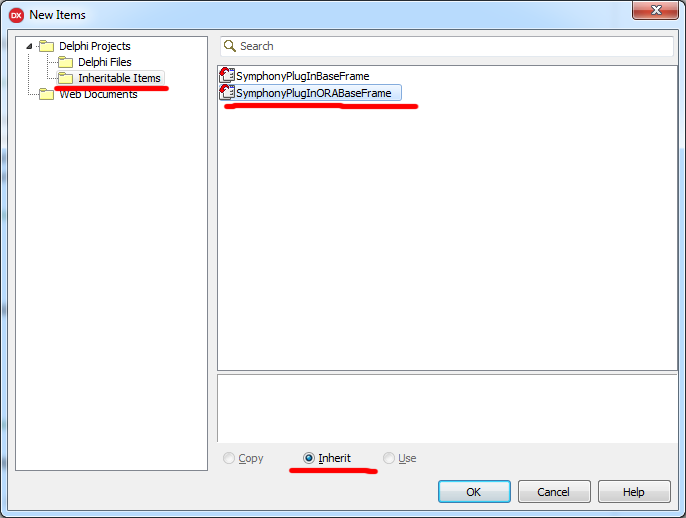
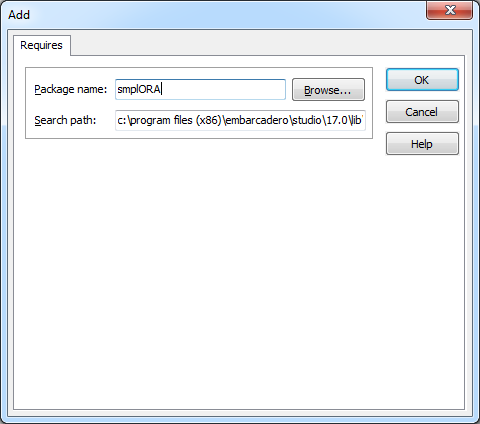
 

В разделе «Description» необходимо установить переключатель параметра «Usage options» в значение «Runtime only», а в поле «Description» можно ввести краткое описание пакета.

### Создание фрейма плагина

Прежде чем создавать плагин, необходимо определить базовый фрейм плагина. Базовый фрейм определяется типом используемой базы данных. Для примера создадим фрейм для базы данных Oracle (тип ORA).

Для типа базы данных ORA в библиотеке PlugIn\_Core (проект smplORA.dproj) создан класс фрейма TSymphonyPlugInORABaseFrame, который располагается в модуле SymphonyPlugIn.ORABaseFrame. Добавим этот модуль к проекту пакета плагина, перед этим добавив к проекту модуль SymphonyPlugIn.BaseFrame.pas. После того как модуль добавлен к проекту, необходимо создать собственный фрейм:

1. Выбираем пункт меню File -> New -> Other…
2. В открывшемся диалоге «New Items» выбираем раздел «Inheritable Items», а в правой части диалога класс SymphonyPlugInORABaseFrame:  
   
3. Сохраняем новый модуль, а модули SymphonyPlugIn.BaseFrame.pas и SymphonyPlugIn.ORABaseFrame удаляем из своего проекта.
4. В раздел «Requires» добавляем ссылки на пакеты smplBaseFrame и smplORA:  
   

В случае появления ошибок при открытии закладки Design нового фрейма, можно открыть в среде Delphi модули SymphonyPlugIn.BaseFrame.pas и SymphonyPlugIn.ORABaseFrame, но включать их в проект не надо.

После того как модуль фрейма создан, необходимо переопределить метод SetOraSession, который находится в разделе protected объявления класса фрейма. Данный метод должен назначить всем компонентам из библиотеки DevArt ODAC открытую сессию, переданную из основного приложения через параметр ASession.

Далее на фрейме располагаются любые компоненты, а в коде фрейма определяется логика работы приложения.

В секции инициализации модуля необходимо зарегистрировать класс фрейма, что бы он был виден в коде вызываемой программы:

**initialization**

System.Classes.RegisterClass(TfrmLNMKGurFrame);

**end** ;

### Создание экспортируемых функций

Экспортируемые функции создаются в любом модуле пакета. Объявление функции должно быть в разделе Interface модуля. Имя функции может быть любое. Функция должна принимать два параметра ASession: TObject и AData: ISymphonyPlugInCommand и возвращать значение типа Boolean. Объявление функции обязательно заканчивается ключевым словом export. Например:

**function** MyFunc(ASession: TObject; AData: ISymphonyPlugInCommand): boolean; **export**;

Модуль может экспортировать несколько функций. Все они должны быть объявлены с ключевым словом export.

После объявления всех экспортируемых функций необходимо после ключевого слова exports через запятую перечислить все экспортируемые модулем функции:

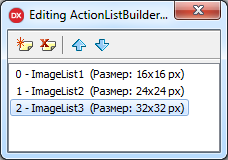
**Exports** MyFunc, MySecondFunc ;

### Создание акций

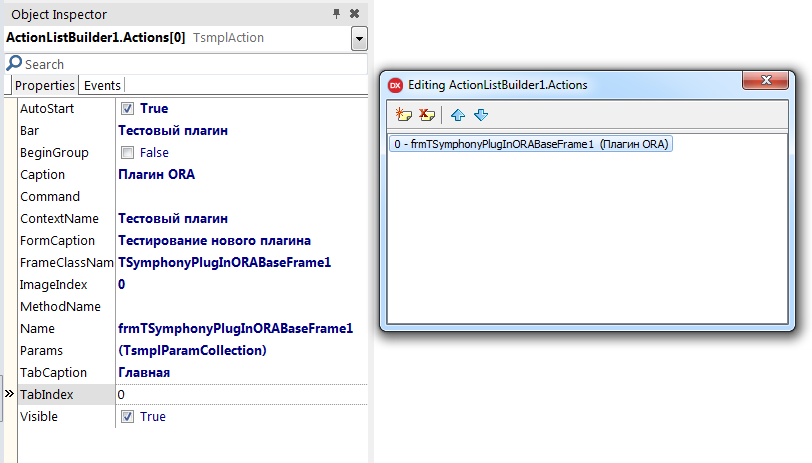
Акции, это объекты, которые создаются плагином и передаются в вызывающую программу в виде интерфейсов. Для облегчения создания акций создан компонент TActionListBuilder.

Для создания акций необходимо добавить модуль данных к пакету и на этом модуле данных расположить компонент TActionListBuilder и компоненты TImageList, с иконками различных размеров. Компоненты TImageList заполняются одинаковым количеством изображений, так что бы изображения с одинаковым индексом во всех списках использовались для одной акции.

После подготовки компонентов TImageList, их необходимо подключить к компоненту TActionListBuilder. Для этого вызываем редактор свойства Images компоненты TActionListBuilder и добавляем все необходимые TImageList в коллекцию:



Далее необходимо вызвать редактор свойства Actions. В эту коллекцию добавляются акции, а в инспекторе объектов заполняются свойства акции:



* AutoStart – акция запускается сразу после загрузки плагина
* Bar – заголовок панели инструментов, на котором будет располагаться кнопка вызова данной акции (далее: кнопка акции)
* BeginGroup – является ли кнопка акции началом новой группы на панели инструментов
* Caption – Заголовок кнопки акции
* Command – текстовая строка, передаваемая процедуре запуска акции в виде параметра
* ContextName – имя контекста, с помощью которого контролируется видимость панелей инструментов
* FormCaption – заголовок дочернего окна, на котором будет размещен фрейм акции
* FrameClassName – имя класса фрейма, который будет создан, при запуске акции. Не допускается заполнять это свойство совместно с свойством MethodName
* ImageIndex – индекс изображения из компонентов TImageList. Изображение будет использовано при создании кнопки акции
* MethodName – имя экспортируемой функции. Не допускается заполнять это свойство совместно с свойством FrameClassName.
* Params – коллекция параметров, которая передаётся в исполняемый метод акции совместно со значением свойства Command
* TabCaption и TabIndex – определяют закладку, на которой будет располагаться кнопка акции
* Visible – видимость акции. Если свойство установлено в False, то кнопка акции не создается.

Для получения списка акций необходимо вызвать метод GetActionList компонента TActionListBuilder.

### Создание фрейма настроек плагина

Фрейм настроек плагина создается, в случае если, для работы плагина необходимо сохранять некоторые параметры пользователя.

Фрейм настроек должен наследовать класс фрейма TfrmSymphonyPlugInTuner, который определен в модуле SymphonyPlugIn.TunerFrame проекта smplBaseFrame библиотеки PlugIn\_Core. Методика создания своего фрейма настройки аналогична созданию фрейма акции.

Фрейм настройки должен содержать элементы управления, с помощью которого пользователь будет настраивать плагин. Для загрузки и выгрузки данных из элементов управления, программист должен реализовать публичные абстрактные методы Load и Save, а так же абстрактные методы возвращающие заголовок параметров плагина и его краткое описание.

### Создание класса реализующего интерфейс IPlugInPackage

В любом модуле пакета необходимо определить класс, реализующий интерфейс IPlugInPackage (модуль: SymphonyPlugIn.PackageInterface):

TLNMKPackageImpl = **class**(TInterfacedObject, IPlugInPackage)

**public**

**function** GetActionList: ISymphonyPlugInActionList ;

**function** TunerFrameClassName: **String** ;

**function** PackageDBType: **String** ;

**end**;

Функция GetActionList должна вернуть список акций пакета, который формируется компонентом TActionListBuilder (одноименный метод компонента). Интерфейс ISymphonyPlugInActionList объявлен в модуле SymphonyPlugIn.ActionInterface.

Функция TunerFrameClass должна вернуть имя класса фрейма для настройки плагина. Если плагин не требует настроек, то класс фрейма не реализуется, а данная функция возвращает пустую строку.

Функция PackageDBType должна вернуть строку с типом используемой СУБД. В нашем случае, функция возвращает строку «ORA».

### Функция PackageInterface

Функция PackageInterface вызывается автоматически при загрузке плагина. Имя функции строго задано протоколом загрузки плагинов. Функция должна иметь следующее объявление:

**function** PackageInterface: IPlugInPackage ; **export** ;

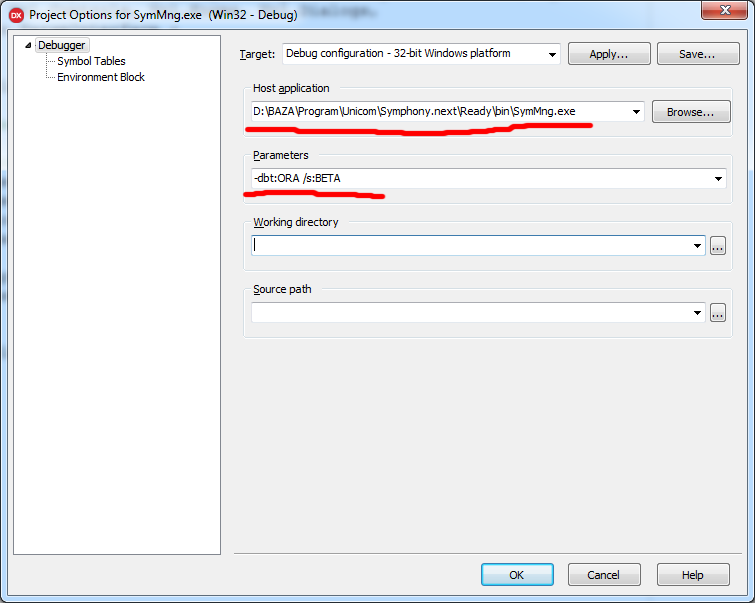
**exports** PackageInterface ;

Функция PackageInterface должна создать объект реализующий интерфейс (см. Создание класса реализующего интерфейс IPlugInPackage) и вернуть его экземпляр в виде интерфейса.

## Запуск плагина под отладчиком

Для запуска плагина под отладчиком необходимо задать host - приложение а так же параметры его запуска. Для этого необходимо открыть или активизировать проект плагина. В меню Run выбрать пункт Parametrs…

В открывшемся диалоге необходимо в поле Host application указать приложение SymMng, а в поле Parametrs указать парметры запуска приложения SymMng:



Далее в коде плагина расставляются точки останова и запускается отладка плагина (клавиша F9).