Библиотека "Robin"для создания интерфейсов к БД

[Основные настройки и запуск 2](#_Toc47738063)

[Настройки форм, методов и команд 2](#_Toc47738064)

[Создание меню 2](#_Toc47738065)

[Конфигурационный файл приложения 2](#_Toc47738066)

[Запуск приложения 3](#_Toc47738067)

[Хранение характеристик форм 3](#_Toc47738068)

[Работа с формой для списков (FormList) и объектами списка (DataList) 4](#_Toc47738069)

[Создание формы 4](#_Toc47738070)

[Источник данных списка 4](#_Toc47738071)

[Запуск запроса и обновление формы 4](#_Toc47738072)

[Формирование столбцов 4](#_Toc47738073)

[Управление значениями параметров запроса 5](#_Toc47738074)

[Событие получения данных 5](#_Toc47738075)

[Выбор параметра запроса из списка 5](#_Toc47738076)

[Логика поведения элементов формы и списков 5](#_Toc47738077)

[Получение данных записи или отмеченных записей списка 6](#_Toc47738078)

[Связи списков на форме 6](#_Toc47738079)

[Добавление команды 6](#_Toc47738080)

[Обработка выбора из списка в список 7](#_Toc47738081)

[Редактирование данных - форма FormEdit 8](#_Toc47738082)

[Создание формы редактирования 8](#_Toc47738083)

[Запуск формы редактирования и передача параметров 8](#_Toc47738084)

[Получение данных на форму 9](#_Toc47738085)

[Выбор из формы списка - элемент SelectBox 9](#_Toc47738086)

[Проверка данных перед записью 10](#_Toc47738087)

[Запись данных 10](#_Toc47738088)

[Списки связанных записей 10](#_Toc47738089)

[Логика поведения элементов формы 11](#_Toc47738090)

[Полезные классы и методы 11](#_Toc47738091)

## Основные настройки и запуск

### Настройки форм, методов и команд

Воспользоваться приложением Master.

Настройки хранятся в dm.tCommand

Поле "Код" настройки уникально в рамках приложения, будет использовано в необходимых местах - настройках пунктов меню, свойствах списков и форм, как параметр методов для вызова форм, методов и sql-команд.

Особенность настройки команд sql:

- переменные параметров декларировать не нужно (будет ошибка) - они декларируются в момент формирования запускаемого sql-кода

- при получении null с клиента переменная параметра декларируется как nvarchar

### Создание меню

Путем добавления записей в dm.tMenu:

delete dm.tMenu where appcode = 'код\_приложения'

insert dm.tMenu (code, appcode, parent, ord, caption, img, execType, command)

values

('код\_пункта', ' код\_приложения', 'код\_родительского\_пункта', порядок, 'Наименование', 'имя\_картинки\_из\_ресурсов', 0 = команда, 1 = подменю, 'код\_настройки\_формы\_или\_метода')

### Конфигурационный файл приложения

Файл имя\_приложения.xml может содержать такие параметры (тэги Prop):

<?xml version="1.0" encoding="windows-1251"?>

<AppConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Prop Name="ConnectionString" Value="Data Source=сервер;Initial Catalog=база;Integrated Security=True" />

<Prop Name="AppName" Value="наименование приложения" /> <!-- для презентации - напр.заголовок главного окна -->

<Prop Name="AppCode" Value="код приложения" /> <!-- для указания в таблицах настройки (tForm, tCommand, tMenu, tFormOptions) -->

<Prop Name="AppMode" Value="что-то" /> <!-- для целей отладки и др.логики -->

<Prop Name="ImgAsm" Value="имя dll или exe" /> <!-- сборка с иконками в ресурсах (по умолчанию - само приложение) -->

<Prop Name="UserOptionsInDB" /> <!-- или <Prop Name="UserOptionsInFile" - где хранить настройки форм - в табл. tFormOptions для пользователя или в файле FormsИмяПриложения.xml/> -->

<Prop Name="RestoreFormPos" /> <!-- восстанавливать позицию форм -->

<Prop Name="CommandTimeout" Value="30" /> <!-- макс время ожидания (сек) начала выполнения команды, если не задан - по умолчанию для mssql - 30 сек -->

</AppConfig>

Можно задавать и другие Prop, которые потом можно получить через:

AppConfig.Prop("имя\_из\_aтрибута\_Name")

Можно использовать файл настроек с другим именем, тогда его имя нужно указать как параметр при запуске приложения.

### Запуск приложения

* Метод Main должен выглядеть так:

static void Main(string[] args) {

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Context.Run(new FMain(), args.Count() > 0 ? args[0] : "");

}

* Делаем главную форму (например FMain)

В ней минимально достаточно разместить ToolStrip (назвать например tools)

Затем сделать обработчик Shown и в нем - вызов:

Ctx = Context.Self;

Text = $"{AppConfig.Prop("AppName") ?? Text} ({Ctx.Conn.DataSource}.{Ctx.Conn.Database})";

Ctx.FillMenu(tools);

Тогда в заголовке пропишется "имя приложения сервер.база"

И сформируются пункты меню из dm.tMenu

Впрочем, кнопки в меню можно добавить и в приложении и у них в обработчиках прописать вызов нужных форм, например:

Context.Self.ExecForm(Context.Self.GetForm("имя\_формы\_из\_настроек"), this, FormModes.Single);

### Хранение характеристик форм

В базе, в таблице tFormOptions, для каждого пользователя сохраняются характеристики формы, которые были на момент ее последнего закрытия: размеры, положение на экране, положение и ширина столбцов, положение сплиттеров, значения элементов-параметров. Сохраненные характеристики форм можно сбросить для пользователя через программу Master (пока только сразу для всех форм)

Если в конфигурационном файле задан параметр "UserOptionsInFile", то настройки хранятся в файле FormsИмяПриложения.xml

## Работа с формой для списков (FormList) и объектами списка (DataList)

### Создание формы

1. Наследовать форму от FormList, задать свойства:

*LoadDataWithForm* = true чтобы запрос отработал сразу после запуска формы

(иначе будет ждать, пока нажмут кнопку или F5)

1. Положить на форму компонент списка (DataList) и задать его свойства:

*AutoColumns* = true если нужно автоматом генерить столбцы по всем полям

*KeyNames* = имена полей, составляющих уникальный ключ записи

*ParamPanel* = панель с элементами для параметров запроса

*QueryCmdCode* - код настройки sql (задается в программе Master)

или *QuerySql* - непосредственно текст команды sql

*ShowCheckBoxes* = true для возможности отмечать записи галочками

1. Если необходимо задать параметры запроса для списка - на панель, указанную в *ParamPanel*, можно добавить элементы управления. Их свойство, содержащее данные, попадет в переменную параметра sql-запроса с тем же именем, что и имя элемента (без символов подчеркивания в начале)

### Источник данных списка

Могут быть варианты:

* первый результат запроса, указанного в *QueryCmdCode* или *QuerySql -* это будет DataTable
* в обработчике события GetData объекта DataList указать перечислимый объект для свойства Result параметра ProcessDataEventArgs. Это не обязательно DataTable, например:

private void dataList1\_GetData(object sender, ProcessDataEventArgs e) {

e.Result = Directory.GetFiles("...")

.Select(x => new { name = Path.GetFileName(x), type = Path.GetExtension(x), path = x })

.ToList();

}

См.также "Событие получения данных"

### Запуск запроса и обновление формы

Обновить через объект списка: dataList1.LoadData();

Обновить через объект формы, например так можно обновить другую открытую форму:

var f = (Ctx.LivingForms["имя\_главной\_формы.имя\_формы"] as FormList);

f.LoadGrid(f.имя\_главного\_объекта\_списка);

### Формирование столбцов

Добавляя столбец, обязательно указываем DataPropertyName

Выравнивание, форматы и др. можно указать в DefaultCellStyle

Можно программно добавить столбцы в событии Load формы:

grid.Columns.Add(NewCol(...))

DataGridViewTextBoxColumn NewCol(string name, string head, string field, int width,

string format = null) {

var col = new DataGridViewTextBoxColumn() {

Name = name, HeaderText = head, DataPropertyName = field, Width = width };

if (!string.IsNullOrEmpty(format))

col.DefaultCellStyle = new DataGridViewCellStyle() { Format = format };

return col;

}

### Управление значениями параметров запроса

Автоматически для каждого списка собирается коллекция полей-параметров, из которых будут взяты значения параметров запроса. Можно обработать события для внесения изменений в эту коллекцию и/или в содержимое этих полей-параметров:

* CreateUiParamsList - при инициализации формы, сразу после создания и автоматического заполнения коллекции.
* SetUiParamsDefaultValues - срабатывает сразу после CreateUiParamsList (до загрузки сохраненных значений параметров) или в процессе работы, после очистки полей-параметров списка (по Ctrl+F5). Обработчик события может быть использован, например, для установки значений параметров по умолчанию.
* SetUiParamsValuesAfterLoad - при открытии формы, после загрузки сохраненных значений полей-параметров, может быть использовано например для принудительной установки параметров.

Загрузка/Сохранение полей-параметров происходит при открытии/закрытии формы.

Можно скорректировать список и значения параметров, полученных при открытии и подготовленных для сохранения перед закрытием формы в обработчиках событий SetLoadedUiParams и PrepareUiParamsForSave.

Перед выполнением запроса можно скорректировать подготовленный словарь параметров запроса в обработчике события QueryParamsCheck или следующего за ним события GetData, которое срабатывает до стандартного выполнения запроса (см."Событие получения данных")

Словарь параметров передается в поле Pars параметров этих событий.

### Событие получения данных

У объекта списка может быть задан обработчик события GetData, в котором можно скорректировать словарь параметров запроса и присвоить результат запроса или любую коллекцию в поле Result параметра события и отменить дальнейшую стандартную обработку (которая использует QueryCmdCode или QuerySql и возвращает DataTable).

Тогда объект из Result станет источником данных для списка.

### Выбор параметра запроса из списка

Для выбора из формы выбора используется элемент управления SelectBox - см. в описании FormEdit.

Для выбора из выпадающего списка - обычный ComboBox. Для его заполнения данными из БД удобно использовать обработчики событий формы - RefreshList (для FormList) и Load. Можно применять метод GetDataToCombo()

### Логика поведения элементов формы и списков

Для обработки поведения элементов формы используем событие ControlTrigger, которое срабатывает при определенных событиях определенных элементов, на которые происходит подписка в SubscribeControlTrigger() - это события изменения состояния (Checked, Selected, ...), входа (Enter), выхода (Leave)

Для обработки поведения списка используем событие WhatsUp - срабатывает по завершении OnMouseUp, OnKeyUp, LoadData, SetFilter. Используется например для установки доступности кнопок в зависимости данных от текущей строки.

### Получение данных записи или отмеченных записей списка

Используются функции DataList:

* GetRowObject - для указанного индекса или текущей строки вернет object = DataRow или DataBoundItem, если это не DataRow
* GetKey - для получения ключа записи указанной или текущей строки. Ключ - это object = словарь строка-объект
* GetCheckedRowsKeys - для получения ключей отмеченных записей
* GetCheckedRowsIdx - вернет индексы отмеченных строк.
* GetKeyRow - вернет индекс строки по указанному ключу

### Связи списков на форме

Для списка с дочерними записями должны быть указаны следующие свойства:

ParentGrid = список с главными записями

ExtParamsMap = "имя\_параметра\_запроса=имя\_внешнего\_параметра;..." - по этому соответствию фильтруются записи в дочернем списке

ParamPanel = панель с полями-параметрами

### Добавление команды

1. В Load прописать создание кнопок и обработчиков кнопок, клавиш и кликов, которые будут вызывать метод обработки:

panel.Items[

panel.Items.Add(

new ToolStripButton() {

Name = "addcopy",

Image = Ctx.GetImage(Ctx.ImgAsmName, "add"),

ToolTipText = "Добавить (Ins)" })

].Click += (o, ek) => Commands("addcopy", grid);

grid.KeyDown += (o, ek) => {

base.OnKeyDown(ek);

if (ek.KeyCode == Keys.Insert && ek.Modifiers == Keys.None)

Commands("add", grid);

};

grid.DoubleClick += (o, ek) => {

grid.DoubleClickCell((g) => Commands("edit", grid));

};

1. Создать метод обработки, в котом запускать форму редактирования через ExecFormEdit или команду SQL через ExecCommand

Например:

void Commands(string cmd, DataList grid) {

var r = (grid.GetRowObject() as DataRow);

// удаление

if (cmd == "del") {

if (grid.CurrentRow == null) return;

var ms = "Удалить отмеченные записи?";

ExecCommand(null, "Delete", grid, null, null, ms, true);

}

if (cmd == "edit") {

if (dataList1.CurrentRow == null) return;

ExecFormEdit(null, view.EditForm, FormModes.Default, grid, "id=id");

}

}

### Обработка выбора из списка в список

1. Вызвать форму выбора через ExecFormSelect, передав параметр getMulti = true
2. Для обработки результата выбора:

* в ExecFormSelect передать обработчик Action<object>, принимающий объект результата,
* или - уже после вызова функции обработать объект результата, который будет помещен в Context.TransferObject.

В случае множественного выбора объект результата представляет собой List<object> из выбранных DataRow

## Редактирование данных - форма FormEdit

### Создание формы редактирования

1. В программе Master добавить настройку формы. Код настройки передается в метод ExecFormEdit при вызове редактора
2. В программе Master добавить настройки sql

* Запросы для выдачи данных в редактор в режиме добавления и обновления. В качестве параметров передаются:

- ключ существующей записи или null

- для дочерних записей - необходимые поля родительской записи

Эти параметры как правило берем из списка, откуда вызываем редактор.

Запросы должны вернуть одну запись. Соответствующий ей объект DataRow попадает в свойство формы SourceObject

* Команды для добавления и изменения. В качестве параметров - необходимые данные с формы, как правило это поля объекта с данными редактора (SourceObject). Команда должна вернуть ключ обработанной записи.

1. Добавить в проект новую форму. Наследовать ее от FormEdit. Элементы формы по возможности назвать так же, как поля полученного объекта (или добавить в начало имени символы "\_"). Если элементы имеют имена, отличные от имен полей или нужен какой-то особый биндинг - нужно добавить обработчики событий DataMapCreate и/или AfterBinding.
2. Настройки sql необходимо указать в свойствах формы SelectCmdCode, SelectNewCmdCode, InsertCmdCode, UpdateCmdCode
3. Если нужно указать sql напрямую - используем свойства SelectSql, SelectNewSql, InsertSql, UpdateSql
4. Необходимые действия при инициализации формы можно прописать в обработчиках событий:

* NewRecInit или ViewOnlyInit - особые действия для формы в режиме NewRecEdit или ViewOnlyEdit
* Load или Show - остальные действия, вызываются после обработчиков NewRecInit / ViewOnlyInit

1. Про особенности обработки получения данных - см. "Получение данных на форму"
2. Про особенности проверки и сохранения данных - см. "Проверка данных перед записью" и "Запись данных"
3. Прописать вызов формы в нужном месте, например в обработчике команды (кнопка, клик) - через метод ExecFormEdit. - см. далее.

### Запуск формы редактирования и передача параметров

Штатный запуск - через вызов метода ExecFormEdit

Обязательно передать

* form - объект формы редактора или formCode - код формы из настроек.
* viewMode - один из режимов FormModes - Default, NewRecEdit или ViewOnlyEdit

При необходимости:

* grid- Список, из которого запускаем. Если задан - поля источника текущей строки берутся в качестве параметров
* mapForParams - Строка соответствия имен: "формируемый1=полученный1;...", где формируемый - параметр для запроса данных на форму, полученный - параметр из списка и customParams. Если нет полученного, параметр будет сформирован со значением null. Если соответствий не задано (по умолчанию) - берем все полученные параметры
* customParams - Параметры ключ-значение. Если заданы - считаются приоритетнее, чем одноименные поля из списка
* mapForResult - Строка соответствия имен: "формируемый1=полученный1;...", где полученный - поле результата выполнения скрипта вставки/обновления, формируемый - параметр поиска текущей записи после обновления данных в списке
* formModes - Другие необходимые режимы в дополнение к viewMode, по умолчанию - Modal

По окончании работы метода, если редактируемая запись была сохранена, будет заполнено свойство контекста TransferObject объектом с ключами сохраненной записи. Если вызов был из списка - список обновится.

### Получение данных на форму

Процесс получения данных происходит после инициализации формы:

1. Событие QueryParamsCheck можно обработать для проверки и коррекции полученного словаря параметров
2. Событие GetData срабатывает сразу после QueryParamsCheck и перед стандартным обработчиком. Получает подготовленный словарь параметров. Если в обработчике получаем данные для формы, то объект, содержащий эти данные должен быть помещен в свойство Result параметра события. Чтобы не вызывался обработчик по умолчанию, нужно установить признак отмены дальнейшей обработки.

Обработчик по умолчанию использует команду из свойств "Select..." в зависимости от режима - новая запись или нет. "...CmdCode" используется, если "...Sql" не указан.

По завершении обработки события полученный объект становится источником данных формы. Если объект не получен, форма не будет создана

1. Происходит его биндинг полей источника данных и элементов формы. Соответствие определяется по именам - элементы формы по возможности должны быть названы так же, как поля полученного объекта (в начало имени можно добавить символы "\_"). Если элементы имеют имена, отличные от имен полей или нужен какой-то особый биндинг - нужно добавить обработчики событий DataMapCreate и/или AfterBinding.

### Выбор из формы списка - элемент SelectBox

Необходимо в конструкторе задать свойства:

* Disabled - Должен быть установлен, если необходимо сделать элемент недоступным. Изменяет цвет и доступность выбора.
* Nullable - Признак, допустимо ли пустое значение. Если установлен и Editable = false, то при нажатии на Delete обнуляются и ранее выбранный объект, и поля формы, заполненные в результате выбора.
* Editable - Признак - допустимо ли редактирование текста. Если установлен, то для сброса других полей, заполненных в результате выбора, нужно использовать например обработчики событий клавиатуры.
* SelectionForm - Форма для выбора (указывается код формы из настроек)
* KeyMap - Пары "поле ключа для формы выбора = поле объекта-источника"
* FilterMap - Пары "поле фильтра для формы выбора = поле объекта-источника"
* ResultMap - Пары "поле целевого объекта = поле выбранного объекта"

Работает так:

1. При нажатии кнопки раскрытия списка (или F4) срабатывает событие ExecSelectionForm. Если оно не обрабатывается или при обработке нет отказа от дальнейшей обработки - вызывается форма по коду в SelectionForm, в режиме выбора.

Для вызова используется метод ExecFormSelect, в который передается исходный объект (по умолчанию - объект данных из формы редактора) и словарь доп.параметров, который может быть задан в обработчике события.

Для уточнения того, какие поля переданного объекта использовать в качестве ключевых и в качестве полей фильтра в списке на форме выбора, используются свойства KeyMap и FilterMap.

1. После выбора записи из списка, объект из выбранной записи попадает в свойство SelectedObject и происходит заполнение полей целевого объекта (по умолчанию - объекта данных редактора). Соответствие полей выбранного объекта и полей, которые необходимо заполнить, указывается в свойстве ResultMap.

До заполнения полей срабатывает событие SetResult, в обработчике которого можно сделать проверки и коррекции данных результата и занесение их еще куда-то…

В обработчике события SetResult для отказа от занесения данных результата в целевой объект можно проставить признак отказа от дальнейшей обработки.

1. После обработки выбранного объекта и занесения данных результата в целевой объект, срабатывает событие AfterSetResult - вне зависимости от занесения данных в целевой объект.

### Проверка данных перед записью

Проверить данные перед сохранением можно двумя способами, которые можно использовать одновременно:

1. Обработать событие SaveParamsCheck. При выявлении ошибок в данных должен быть заполнен словарь ("имя поля или элемента" - "сообщение")
2. Указать sql-команду для проверки данных в свойстве CheckSql или в настройке и код настройки указать в свойстве CheckCmdCode. Если есть ошибки, SQL должен вернуть таблицу из двух строковых полей - имя и сообщение.

Событие SaveParamsCheck срабатывает до вызова sql. Результат sql попадает в тот же словарь.

Если словарь заполнен, будут выведены сообщения и запись данных будет отменена.

### Запись данных

Процесс записи (инициируется нажатием кнопки или комбинацией клавиш) :

1. Событие SetData. Срабатывает после SaveParamsCheck и после вызова команды, указанной в "Check…", и перед стандартным обработчиком (он не определен только в режиме просмотра).

В обработчике выполняем команду, которой передаем параметры (они уже подготовлены в виде словаря и передаются в параметре события). Результат выполнения должен быть помещен в поле параметра события. Как правило это набор из одной строки с ключевыми полями обработанной записи.

Чтобы не вызывался обработчик по умолчанию, нужно установить признак отказа от дальнейшей обработки.

1. Обработчик по умолчанию использует команду из свойств "Insert..." или "Update..." в зависимости от режима - новая запись или нет. "...CmdCode" используется, если "...Sql" не указан.
2. Объект первой строки результата выполнения помещается в свойство контекста TransferObject. Если результат не определен - будет ошибка и форма не будет закрыта.

### Списки связанных записей

Чтобы добавить списки записей, связанных с данными формы редактирования, необходимо поместить на форму объекты DataList и указать для них в свойстве ExtParamsMap соответствие имен полей списка и полей формы. Тогда данные в списках будут отфильтрованы нужным образом.

В случае новой записи списки отображаться не будут. Это поведение можно изменить в обработчике события NewRecInit.

### Логика поведения элементов формы

Для обработки поведения элементов формы используем событие ControlTrigger, которое срабатывает при определенных событиях определенных элементов, на которые происходит подписка в SubscribeControlTrigger() - это события изменения состояния (Checked, Selected, ...), входа (Enter), выхода (Leave)

## Полезные классы и методы

См. в файле "Библиотека\_Robin.xlsx"

Там же - дополнительные подробности по меотдам и свойствам, упомянутым в описании.