Информационно-аналитическая система «Своды»

Инструкция разработчика

Содержание

[Перечень терминов и сокращений 18](#_Toc106209139)

[1 Создание и подключение нового экспортера. Ограничение на использование экспорта 21](#_Toc106209140)

[1.1 Создание и подключение нового экспортера 21](#_Toc106209141)

[1.1.1 Создание в API класса-наследника от ЭкспортерДанныхОтчетныхФорм 22](#_Toc106209142)

[1.1.2 Переопределение обязательных виртуальных методов 22](#_Toc106209143)

[1.1.3 Задание наименования экспортера и реализация логики экспорта 24](#_Toc106209144)

[1.2 Ограничение на использование экспорта 25](#_Toc106209145)

[1.2.1 Перекрытие методов «ЭкспортерПодходитДляФормы»/ «ЭкспортерПодходитДляФормыФК» 26](#_Toc106209146)

[1.2.2 Использование уже имеющейся реализации экспорта с абстрактным классом экспортера 27](#_Toc106209147)

[2 Создание и подключение нового импортера. Ограничение на использование Импорта 28](#_Toc106209148)

[2.1 Создание и подключение нового импортера 28](#_Toc106209149)

[2.1.1 Создание в API класса-наследника от ИмпортерДанныхОтчетныхФорм 29](#_Toc106209150)

[2.1.2 Переопределение обязательных виртуальных методов 29](#_Toc106209151)

[2.1.3 Задание наименования импортера и реализация логики импорта 31](#_Toc106209152)

[2.2 Ограничение на использование импортера 32](#_Toc106209153)

[2.2.1 Перекрытие методов ИмпортерПодходитДляФормы/ИмпортерПодходитДляФормыФК 32](#_Toc106209154)

[2.2.2 \*Использование уже имеющейся реализации импорта с абстрактным классом импортера 33](#_Toc106209155)

[3 Калькулятор Условных Атрибутов 34](#_Toc106209156)

[3.1 ДополнительныйАтрибут 35](#_Toc106209157)

[3.2 Формат 36](#_Toc106209158)

[3.3 УсловныйФормат 36](#_Toc106209159)

[3.4 ФорматДляОбласти 37](#_Toc106209160)

[3.5 ИдентификаторЭлементаФормы 37](#_Toc106209161)

[3.6 РезультатРасчета 37](#_Toc106209162)

[4 Переопределение базового обработчика сводной формы, добавление записей в историю сборки сводной 39](#_Toc106209163)

[4.1 Сбор сводной в Сводах 48](#_Toc106209164)

[4.1.1 Получение параметров и формирование списка форм для сведения 49](#_Toc106209165)

[4.1.2 Сборка сводной 50](#_Toc106209166)

[4.1.3 Сведение ячеек 51](#_Toc106209167)

[4.1.4 Сведение строк 52](#_Toc106209168)

[4.1.5 Сведение таблиц форм 52](#_Toc106209169)

[4.1.6 Сведение динамических таблиц форм 53](#_Toc106209170)

[4.1.7 Сведение статических таблиц форм 54](#_Toc106209171)

[4.1.8 Методы, переопределяемые в потомках 54](#_Toc106209172)

[5 Сверка классификаторов 56](#_Toc106209173)

[6 Наложение ограничений на отображение/ редактирование вкладок (таблиц), других элементов формы. Обязательность заполнения ячеек 60](#_Toc106209174)

[6.1 Обязательность заполнения ячеек 63](#_Toc106209175)

[7 Работа с пользовательскими параметрами, зависимость пользовательских параметров 65](#_Toc106209176)

[7.1 Новые параметры 65](#_Toc106209177)

[7.1.1 Подготовка прикладного кода 65](#_Toc106209178)

[7.1.2 Fluent Builder пользовательских параметров 65](#_Toc106209179)

[7.1.2.1 Создание параметров с помощью билдера 65](#_Toc106209180)

[7.1.2.1.1 Примеры использования 68](#_Toc106209181)

[7.1.2.2 ApplyConfig 70](#_Toc106209182)

[7.1.2.3 WithColumns 72](#_Toc106209183)

[7.1.3 Пользовательский параметр выбора из справочника 74](#_Toc106209184)

[7.1.3.1 Выбор из физического справочника 74](#_Toc106209185)

[7.1.3.2 Выбор из виртуального справочника 75](#_Toc106209186)

[7.1.3.3 Выбор из справочника с ленивой загрузкой записей 77](#_Toc106209187)

[7.1.4 Пользовательский параметр выбора из таблицы 80](#_Toc106209188)

[7.1.4.1 Описание реализации 80](#_Toc106209189)

[7.1.4.2 Пример использования 80](#_Toc106209190)

[7.1.5 Пользовательский параметр выбора компонента отчётного периода 83](#_Toc106209191)

[7.1.6 Пользовательский параметр выбора отчётного периода 85](#_Toc106209192)

[7.1.7 Пользовательский параметр выбора учреждений 86](#_Toc106209193)

[7.1.7.1 CommonAPI 86](#_Toc106209194)

[7.1.7.1.1 DepartmentSelector 86](#_Toc106209195)

[7.1.7.2 CommonWebAPI 94](#_Toc106209196)

[7.1.7.2.1 DеpartmentSelector 94](#_Toc106209197)

[7.1.7.3 CommonWinAPI 103](#_Toc106209198)

[7.1.7.3.1 DepartmentSelector 103](#_Toc106209199)

[7.1.8 Пользовательский параметр выбора элемента цепочки 106](#_Toc106209200)

[7.1.9 Пользовательский параметр выбора элементов цепочки из компонентов 108](#_Toc106209201)

[7.1.10 Создание кастомного параметра 110](#_Toc106209202)

[7.1.10.1 Общая логика 110](#_Toc106209203)

[7.1.10.2 WEB 119](#_Toc106209204)

[7.1.10.3 Полный код DepartmentSelector 137](#_Toc106209205)

[7.1.11 Соответствие интерфейсов пользовательских параметров 137](#_Toc106209206)

[7.1.12 Соответствие пользовательского параметра и элемента списка его значений 138](#_Toc106209207)

[7.1.13 Зависимость параметров 138](#_Toc106209208)

[7.1.13.1 Реализация зависимостей между пользовательскими параметрами 138](#_Toc106209209)

[7.1.14 UML 141](#_Toc106209210)

[7.2 Старые параметры 141](#_Toc106209211)

[7.2.1 Формирование списка пользовательских параметров 141](#_Toc106209212)

[7.2.2 Получение значений пользовательских параметров 142](#_Toc106209213)

[7.2.3 Список добавляемых пользовательских параметров 142](#_Toc106209214)

[7.3 Зависимость параметров 145](#_Toc106209215)

[8 Работа с хранимыми блокировками 148](#_Toc106209216)

[9 Асинхронные обработки и управление прогресс баром в ПФ 150](#_Toc106209217)

[9.1 Пример реализации асинхронной операции в прикладном коде. 150](#_Toc106209218)

[9.2 Пример управления прогресс баром в макросе Печатной форме 152](#_Toc106209219)

[9.3 Пример файла svody.config с заданным количеством параллельно выполняемых обработок 153](#_Toc106209220)

[10 Описание виртуальных методов обработчика с их параметрами 155](#_Toc106209221)

[11 Работа с константами 164](#_Toc106209222)

[11.1 Форматирование констант 168](#_Toc106209223)

[11.2 Заполнение констант через макрос 169](#_Toc106209224)

[12 Создание печатной формы 170](#_Toc106209225)

[12.1 XLSX и XLSM 170](#_Toc106209226)

[12.1.1 Работа с файлами excel через IOpenXmlWorkbook 171](#_Toc106209227)

[12.1.1.1 Раскрытие 171](#_Toc106209228)

[12.1.1.2 Свойства и методы IOpenXmlWorkbook (документа) 172](#_Toc106209229)

[12.1.1.3 Свойства и методы ISheet (листа) 173](#_Toc106209230)

[12.2 DOCX 180](#_Toc106209231)

[12.2.1 Примеры кода 181](#_Toc106209232)

[12.2.1.1 Добавление текста со стилями 181](#_Toc106209233)

[12.2.1.2 Выравнивание текста 182](#_Toc106209234)

[12.2.1.3 Интервалы и отступы 183](#_Toc106209235)

[12.2.1.4 Добавление списков 184](#_Toc106209236)

[12.2.1.5 Добавление верхних и нижних колонтитулов и нумерация страниц 185](#_Toc106209237)

[12.2.1.6 Копирование текста из параграфа в параграф 186](#_Toc106209238)

[12.2.1.7 Работа с изображениями 186](#_Toc106209239)

[12.2.1.8 Работа с таблицами 187](#_Toc106209240)

[12.2.1.9 Задание разрыва страницы и параметров страницы 188](#_Toc106209241)

[12.2.1.10 Добавление защиты документа с паролем и без него 189](#_Toc106209242)

[12.2.1.11 Диаграммы 189](#_Toc106209243)

[12.3 PDF 194](#_Toc106209244)

[12.4 Параметры отчета 195](#_Toc106209245)

[12.4.1 Общая информация о выгружаемой печатной форме 196](#_Toc106209246)

[12.4.2 ПростыеПараметрыОтчета 198](#_Toc106209247)

[12.4.2.1 Использование «ПростыеПараметрыОтчета» в шаблонах ПФ 200](#_Toc106209248)

[12.4.3 СтруктурныеПараметрыОтчета. Секции 202](#_Toc106209249)

[12.4.3.1 Использование СтруктурныеПараметрыОтчета для управления содержанием секций печатных форм 204](#_Toc106209250)

[12.5 Атрибуты 207](#_Toc106209251)

[12.5.1 Добавление ПФ через Дизайнер 207](#_Toc106209252)

[12.5.2 Навешанные атрибуты на метод 209](#_Toc106209253)

[13 Работа с данными об ЭП 211](#_Toc106209254)

[14 Работа с субтаблицами (сводная, ПФ) 214](#_Toc106209255)

[14.1 ПФ для субтаблиц 214](#_Toc106209256)

[14.2 Сводная для субтаблиц 215](#_Toc106209257)

[14.3 Модальные субтаблицы 217](#_Toc106209258)

[15 Увязки 219](#_Toc106209259)

[15.1 Внутриформенные увязки 220](#_Toc106209260)

[15.2 Межформенные увязки 222](#_Toc106209261)

[15.3 Внутривкладочные увязки 224](#_Toc106209262)

[15.4 Формирование Пользовательских параметров 225](#_Toc106209263)

[16 Переопределение источника записей справочника 227](#_Toc106209264)

[17 Запись данных в ОФ 229](#_Toc106209265)

[17.1 Через экранную форму 229](#_Toc106209266)

[17.2 Во время вызова события 229](#_Toc106209267)

[17.3 Через импорт 230](#_Toc106209268)

[17.4 Через обработку 230](#_Toc106209269)

[17.5 Через сбор сводной 230](#_Toc106209270)

[17.6 Пример метода для заполнения свободных ячеек и таблиц 230](#_Toc106209271)

[18 Получение другой ОФ 233](#_Toc106209272)

[19 Обрабатываемые события при работе с формой 235](#_Toc106209273)

[19.1 Написание событий макросом 235](#_Toc106209274)

[19.2 Обработки 235](#_Toc106209275)

[19.2.1 ОбработатьСобытиеОткрытияФормы 235](#_Toc106209276)

[19.2.2 ОбработатьСобытиеДоСохраненияФормы 235](#_Toc106209277)

[19.2.3 ОбработатьСобытиеЗакрытияСубтаблицы 236](#_Toc106209278)

[19.2.4 ОбработатьСобытиеЗакрытияФормы 237](#_Toc106209279)

[19.2.5 ОбработатьСобытиеПередЗапускомМакроса 237](#_Toc106209280)

[19.2.6 ОбработатьСобытиеПередИмпортомДанных 238](#_Toc106209281)

[19.2.7 ОбработатьСобытиеПередОткрытиемТаблицы 238](#_Toc106209282)

[19.2.8 ОбработатьСобытиеПередПроверкойУвязок 239](#_Toc106209283)

[19.2.9 ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы 239](#_Toc106209284)

[19.2.10 ОбработатьСобытиеПередЭкспортомДанных 240](#_Toc106209285)

[19.2.11 ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы 240](#_Toc106209286)

[19.2.12 ОбработатьСобытиеПослеЗагрузкиАрхива 241](#_Toc106209287)

[19.2.13 ОбработатьСобытиеПослеЗапускаМакроса 241](#_Toc106209288)

[19.2.14 ОбработатьСобытиеПослеИзмененияЗначенияЯчейки 241](#_Toc106209289)

[19.2.15 ОбработатьСобытиеПослеИмпортаДанных 242](#_Toc106209290)

[19.2.16 ОбработатьСобытиеПослеПроверкиУвязок 242](#_Toc106209291)

[19.2.17 ОбработатьСобытиеПослеСохраненияСубтаблицы 243](#_Toc106209292)

[19.2.18 ОбработатьСобытиеПослеУстановкиЗначенияЯчейкиСправочника 243](#_Toc106209293)

[19.2.19 ОбработатьСобытиеПослеЭкспортаДанных 244](#_Toc106209294)

[19.2.20 ОбработатьСобытиеПриОткрытииТаблицы 244](#_Toc106209295)

[19.2.21 ОбработатьСобытиеСохраненияСубтаблицы 244](#_Toc106209296)

[19.2.22 ОбработатьСобытиеСохраненияФормы 245](#_Toc106209297)

[19.2.23 ОбработатьСобытиеФормированияВнутреннихУвязок 245](#_Toc106209298)

[19.2.24 ОбработатьСобытиеФормированияМежформенныхУвязок 246](#_Toc106209299)

[19.2.25 ОбработатьНеверноЗаполненныеЯчейкиДоЗакрытияСубтаблицы 246](#_Toc106209300)

[19.2.26 ПолучитьИсточникЗаписейДляВыбораИзСправочника 246](#_Toc106209301)

[19.2.27 СформироватьСписокПользовательскихПараметров 248](#_Toc106209302)

[19.2.28 Макрос для печатной формы 248](#_Toc106209303)

[20 Создание обработки. Атрибуты 250](#_Toc106209304)

[20.1 Добавление обработки через Дизайнер 250](#_Toc106209305)

[20.2 Добавление напрямую в файл метаструктуры 252](#_Toc106209306)

[21 Пересчеты 254](#_Toc106209307)

[22 Вывод результаты файлом/окном 256](#_Toc106209308)

[22.1 Вывод результата окном 256](#_Toc106209309)

[22.1.1 РезультатыВыполненияСверкиДанных 256](#_Toc106209310)

[22.1.1.1 СтолбецСверкиДанныхФормы 256](#_Toc106209311)

[22.1.1.2 СтрокаОтчетаСверкиДанных 257](#_Toc106209312)

[22.2 Вывод результата файлом 259](#_Toc106209313)

[23 Основные методы работы с данными формы 261](#_Toc106209314)

[23.1 ЗагрузитьМетаструктуру 261](#_Toc106209315)

[23.1.1 Параметр типа МетаструктураФормы 261](#_Toc106209316)

[23.2 Загрузить 263](#_Toc106209317)

[23.2.1 Параметр типа Stream 263](#_Toc106209318)

[23.3 ЗагрузитьИзФайла 264](#_Toc106209319)

[23.4 Load 265](#_Toc106209320)

[23.5 ЗагрузитьИзXPathDocument 265](#_Toc106209321)

[23.5.1 Параметр типа XPathDocument 265](#_Toc106209322)

[23.6 СчитатьТаблицы 266](#_Toc106209323)

[23.6.1 Параметр типа XPathNavigator 266](#_Toc106209324)

[23.7 СериализоватьВXML 266](#_Toc106209325)

[23.7.1 Возвращаемый тип XMLDocument 266](#_Toc106209326)

[23.8 GetFromDataXml 267](#_Toc106209327)

[23.9 Сохранить 267](#_Toc106209328)

[23.10 СохранитьВФайл 267](#_Toc106209329)

[23.11 Save 268](#_Toc106209330)

[23.12 SaveTo 268](#_Toc106209331)

[23.13 GetDataFileStream 268](#_Toc106209332)

[23.13.1 Возвращаемый тип MemoryStream 269](#_Toc106209333)

[23.14 MaterializeDictionaryCellValues 269](#_Toc106209334)

[23.15 ЗаполнитьЗначенияСправочников 269](#_Toc106209335)

[23.15.1 Параметр типа ТаблицаДанных 270](#_Toc106209336)

[23.16 ЗаполнитьСвободныеЯчейки 270](#_Toc106209337)

[23.17 ЗаполнитьЗначенияСправочниковСтрокДанных 271](#_Toc106209338)

[23.17.1 Параметр типа СтрокаДанных 271](#_Toc106209339)

[23.18 ЗаполнитьЗначенияСправочниковСвободныхЯчеек 271](#_Toc106209340)

[23.18.1 Параметр типа ЯчейкаДанных 272](#_Toc106209341)

[23.19 ОчиститьДанные 272](#_Toc106209342)

[23.20 СодержитТаблицу 272](#_Toc106209343)

[23.21 НайтиЯчейку 273](#_Toc106209344)

[23.21.1 Тип возвращаемого объекта ТипЯчейки 273](#_Toc106209345)

[23.22 СкопироватьДанные 274](#_Toc106209346)

[23.22.1 Параметр типа ДанныеФормы 274](#_Toc106209347)

[23.23 ПерегенерироватьСтрокиДинамическихТаблиц 275](#_Toc106209348)

[23.24 ПроверитьУникальностьСтрокДинамическихТаблиц 275](#_Toc106209349)

[23.24.1 Возвращаемый тип СтрокаОтчетаСверкиДанных 276](#_Toc106209350)

[23.25 ПересчитатьАвтоблоки 277](#_Toc106209351)

[23.26 УстановитьНеверноЗаполненыеЯчейки 277](#_Toc106209352)

[23.27 ЗаполнитьЗначенияПоУмолчанию 277](#_Toc106209353)

[23.28 SetTableChanged 278](#_Toc106209354)

[23.29 SetTableCommited 278](#_Toc106209355)

[23.30 SetAllTableComited 278](#_Toc106209356)

[23.31 GetTableChanged 279](#_Toc106209357)

[23.32 GetTableCommited 279](#_Toc106209358)

[23.33 ДополнительныеМетодыПолученияДанных.ПолучитьДанныеОтчетнойФормы 279](#_Toc106209359)

[24 Работа с вложениями 281](#_Toc106209360)

[24.1 Настройка вложений 283](#_Toc106209361)

[25 Работа с архивами данных 285](#_Toc106209362)

[25.1 Пример получения архива данных формы 286](#_Toc106209363)

[26 Работа с динамическими таблицами, добавление строк, заполнение данных по умолчанию 287](#_Toc106209364)

[26.1 Добавление строк 287](#_Toc106209365)

[26.2 Заполнение данных по умолчанию 288](#_Toc106209366)

[27 Открытие формы из формы 290](#_Toc106209367)

[27.1 Сигнатуры макросов 291](#_Toc106209368)

[28 Работа вне отчетной формы 294](#_Toc106209369)

[28.1 Кастомизация статусов 294](#_Toc106209370)

[28.2 Обработчик смены статусов. Изменения статуса отчетной формы 297](#_Toc106209371)

[28.2.1 Методы 298](#_Toc106209372)

[28.2.1.1 GetBeforeChangeStatusReportAsync 298](#_Toc106209373)

[28.2.1.2 ДоСменыСтатуса 299](#_Toc106209374)

[28.2.1.3 ПослеСменыСтатуса 300](#_Toc106209375)

[28.2.2 ОбработчикСменыСтатусаФормыПоУмолчанию 300](#_Toc106209376)

[28.3 Скрытие пунктов основного меню в зависимости от каких-то условий 301](#_Toc106209377)

[28.4 Замена карточки учреждения в прикладных API 302](#_Toc106209378)

[28.4.1 Кастомизация инстанцирования формы Учреждения с помощью прикладных API проекта 302](#_Toc106209379)

[28.4.2 Для сводов на .net core и выше 304](#_Toc106209380)

[28.4.3 Пошаговая инструкция замены карточки учреждения 304](#_Toc106209381)

[28.5 Логирование 306](#_Toc106209382)

[28.5.1 Хранение логов 306](#_Toc106209383)

[28.5.2 Конфигурационный файл Nlog 306](#_Toc106209384)

[28.5.2.1 Секция <targets> 307](#_Toc106209385)

[28.5.2.2 Шаблон формирования записи лога 307](#_Toc106209386)

[28.5.2.3 Основные маркеры подстановки 307](#_Toc106209387)

[28.5.2.4 Секция <rules> 307](#_Toc106209388)

[28.5.3 Пример 308](#_Toc106209389)

[28.6 Менеджер фоновых процессов Системы 309](#_Toc106209390)

[28.6.1 Новый интерфейс менеджера фоновых процессов 309](#_Toc106209391)

[28.6.2 Базовый класс фонового процесса и его интерфейс 311](#_Toc106209392)

[28.6.3 Класс для создания параметров запуска экземпляра фонового процесса (показан только интерфейс класса) 312](#_Toc106209393)

[28.6.4 Пример добавления триггера выполнения фонового процесса с помощью нового API 313](#_Toc106209394)

[28.6.5 Конфигурирование менеджера фоновых процессов с помощью web.config 313](#_Toc106209395)

[28.7 Пермиссии и роли пользователя 314](#_Toc106209396)

[28.7.1 Пермиссия. Определение, виды и реализация 314](#_Toc106209397)

[28.7.1.1 Существующие в Системе пермиссии 315](#_Toc106209398)

[28.7.1.2 Задание собственных пиктограмм пермиссий, отображаемых имен и списка разрешенных кодов 317](#_Toc106209399)

[28.7.1.3 Создание пермиссии 319](#_Toc106209400)

[28.7.2 Роли пользователя 321](#_Toc106209401)

[28.7.2.1 Задание роли в Системе и установка ее пользователю 323](#_Toc106209402)

[28.7.3 Пример создания новой пермиссии 326](#_Toc106209403)

[28.7.4 Наложение пермиссий 328](#_Toc106209404)

[28.8 Инициализация веб-приложения 329](#_Toc106209405)

[28.8.1 Пример 330](#_Toc106209406)

[28.9 Логин/ логаут 330](#_Toc106209407)

[28.9.1 Классический способ входа в приложение через страницу логина 331](#_Toc106209408)

[28.9.1.1 События и сервисы при логине/логауте используемые в базовой реализации ISvodyAppAuthorizationModel 334](#_Toc106209409)

[28.9.1.2 Сервис обработки ограничений на количество попыток входа 337](#_Toc106209410)

[28.9.2 Кроссплатформенная авторизация 337](#_Toc106209411)

[28.9.2.1 Open ID Connect / Keycloak 337](#_Toc106209412)

[28.9.2.2 Авторизация по заголовкам запросов 338](#_Toc106209413)

[28.9.2.3 Кросс-авторизация (на платформенных сущностях) 338](#_Toc106209414)

[28.9.2.4 Кросс-авторизация (собственный контроллер 338](#_Toc106209415)

[28.9.3 Кросс-авторизация в Системе. Открытие формы во Frame 339](#_Toc106209416)

[28.9.3.1 Открытие формы во Frame 339](#_Toc106209417)

[28.9.3.2 Пример генерации хеш 340](#_Toc106209418)

[28.9.3.3 Версия 4.1 341](#_Toc106209419)

[28.9.4 Авторизация по заголовкам запроса 342](#_Toc106209420)

[28.9.5 Настройка авторизации по протоколу Open ID Connect 343](#_Toc106209421)

[28.9.5.1 Логика авторизации 343](#_Toc106209422)

[28.9.5.2 Подготовка SSO 344](#_Toc106209423)

[28.9.5.2.1 Bars.AM 344](#_Toc106209424)

[28.9.5.2.2 Keycloak 350](#_Toc106209425)

[28.9.5.2.3 Keycloak (массовый экспорт пользователей) 353](#_Toc106209426)

[28.9.5.3 Конфигурация Системы 355](#_Toc106209427)

[28.9.5.3.1 Вид секции 355](#_Toc106209428)

[28.9.5.3.2 Параметры 356](#_Toc106209429)

[28.9.5.3.3 Пример настроенной секции 357](#_Toc106209430)

[28.9.5.3.4 Особенности настройки приложения СВОДЫ на Linux с Nginx 358](#_Toc106209431)

[28.9.6 Логаут 358](#_Toc106209432)

[28.9.6.1 Очистка неактивных сессий 360](#_Toc106209433)

[28.10 Переопределение стартовой страницы Системы 360](#_Toc106209434)

[28.10.1 Описание способов переопределения стартовой страницы 360](#_Toc106209435)

[28.10.2 Пример добавления шаблона стартовой страницы 362](#_Toc106209436)

[28.10.3 Переопределение Razor view 363](#_Toc106209437)

[28.11 Кастомизация Front-Office части Системы 364](#_Toc106209438)

[28.11.1 Создание и загрузка прикладного JS 364](#_Toc106209439)

[28.11.1.1 Пример настройки кастомизации 367](#_Toc106209440)

[28.12 Миграции в API 369](#_Toc106209441)

[28.12.1 Возможности Системы 369](#_Toc106209442)

[28.12.2 Рекомендации по реализации API 371](#_Toc106209443)

[28.13 Работа через DataStore 373](#_Toc106209444)

[28.14 Работа с экспертизой 379](#_Toc106209445)

[28.14.1 Информация об экспертизе в хранимых данных 381](#_Toc106209446)

[28.14.2 Информация об экспертах в элементе цепочки 383](#_Toc106209447)

[28.15 Виды справочников и возможность работы с ними 386](#_Toc106209448)

[28.15.1 УниверсальныйСправочник 386](#_Toc106209449)

[28.15.2 «Новые» справочники 395](#_Toc106209450)

[28.16 Инструкция по работе с масками ввода 397](#_Toc106209451)

[28.16.1 Основные правила масок ввода (регулярных выражений) 397](#_Toc106209452)

[28.16.2 Дизайнер 398](#_Toc106209453)

[28.16.3 Реализация интерфейса 401](#_Toc106209454)

[28.17 MultiDb 402](#_Toc106209455)

[28.17.1 Получение параметров подключения 402](#_Toc106209456)

[28.17.2 API платформы 404](#_Toc106209457)

[28.17.2.1 QueryContext 404](#_Toc106209458)

[28.17.2.2 DataStore 405](#_Toc106209459)

[28.17.2.2.1 GetQueryFromAllDb 405](#_Toc106209460)

[28.17.2.2.2 GetQuery 405](#_Toc106209461)

[28.17.2.3 Выборка 406](#_Toc106209462)

[28.17.2.4 SQLЗапрос 407](#_Toc106209463)

[28.18 Новые дизайнерские увязки 408](#_Toc106209464)

[28.18.1 Новый раздел в дизайнере 408](#_Toc106209465)

[28.18.2 Написание увязок 408](#_Toc106209466)

[28.18.3 Функции, операнды и операции между функциями 409](#_Toc106209467)

[28.18.3.1 Общие сведения 409](#_Toc106209468)

[28.18.3.2 Для разработки 421](#_Toc106209469)

[28.18.3.2.1 Метод «Handler» 421](#_Toc106209470)

[28.18.3.2.2 Метод «GetTranscript» 424](#_Toc106209471)

[28.18.4 Пользовательские параметры 431](#_Toc106209472)

[28.18.5 Примеры типовых увязок 432](#_Toc106209473)

[28.18.6 Операции с операндами 434](#_Toc106209474)

[28.19 Отправка сообщений, дублирование на почту, вложения 437](#_Toc106209475)

[28.19.1 Конфигурационные файлы 437](#_Toc106209476)

[28.19.1.1 messengerService.config 437](#_Toc106209477)

[28.19.1.2 connection.config 438](#_Toc106209478)

[28.19.1.3 svody.config 438](#_Toc106209479)

[28.19.2 Центр сообщений 439](#_Toc106209480)

[28.19.3 Примеры 440](#_Toc106209481)

[28.20 Расширение Фронтенд Системы версии 5.0 на проектном уровне 442](#_Toc106209482)

[28.20.1 Темы 442](#_Toc106209483)

[28.20.1.1 Создание темы 442](#_Toc106209484)

[28.20.1.2 Настройка сборки приложения с темой 443](#_Toc106209485)

[28.20.2 Custom.scss 445](#_Toc106209486)

[28.20.2.1 Рекомендации по использованию 445](#_Toc106209487)

[28.20.3 apiJs 446](#_Toc106209488)

[28.21 Регистрация SOAP-сервисов 446](#_Toc106209489)

[29 Аналитические выборки 449](#_Toc106209490)

[29.1 Использование классов в аналитических выборках 449](#_Toc106209491)

[29.2 Создание аналитической выборки 450](#_Toc106209492)

# Перечень терминов и сокращений

| Термин, сокращение | Определение |
| --- | --- |
| API | Application Programming Interface – интерфейс программирования приложений – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах |
| Back-office | Операционно-учётное подразделение, обеспечивающее работу подразделений, участвующих в управлении активами и пассивами организации, осуществляющей деятельность на финансовых рынках |
| Bars.AM | Программное обеспечение, которое является средством защиты конфиденциальной информации и реализует механизм единого входа в систему, обеспечивая централизованное управление паролями |
| DTO | Data Transfer Object – один из шаблонов проектирования, используется для передачи данных между подсистемами приложения |
| Front-office | Общее наименование группы подразделений или процессов в организациях, отвечающих за непосредственную работу с клиентами, заказчиками |
| Keycloak | Продукт с открытым кодом для реализации технологии единого входа с возможностью управления доступом |
| Web-браузер | Прикладное программное обеспечение для просмотра web-страниц, содержания web-документов, компьютерных файлов и их каталогов; управления web-приложениями |
| Web-приложение, приложение | Клиент-серверное приложение, в котором клиентом выступает web-браузер, а сервером – web-сервер |
| XML | Расширяемый язык разметки |
| БД | База данных |
| ВР | Вид расхода |
| Дизайнер | Дизайнер отчетных форм информационно-аналитической системы «Своды» |
| ИНН | Идентификационный номер налогоплательщика |
| КБК | Коды бюджетной классификации |
| Код | Идентификатор записи хранимого объекта (отчетный период, цепочка сдачи отчетности и т.д.) |
| Компонент отчетного периода | Запись, позволяющая связать цепочку сдачи отчетности с пакетом форм |
| КПП | Коробка переключения передач |
| ОКАТО | Общероссийский классификатор объектов административно-территориального деления |
| ОКВЭД | Общероссийский классификатор видов экономической деятельности |
| ОКИН | Общероссийский классификатор информации о населении |
| ОКОГУ | Общероссийский классификатор органов государственной власти и управления |
| ОКОНХ | Общесоюзный классификатор «Отрасли народного хозяйства» |
| ОКОПФ | Общероссийский классификатор организационно-правовых форм |
| ОКПО | Общероссийский классификатор предприятий и организаций |
| ОКТМО | Общероссийский классификатор территорий муниципальных образований |
| ОКУД | Общероссийский классификатор управленческой документации |
| ОКФС | Общероссийский классификатор форм собственности |
| ОП | Отчетный период |
| Оператор | Пользователь Системы, которому назначено рабочее учреждение и даны права на заполнение всех или только некоторых отчетных форм этого учреждения |
| Отчетная форма | Абстрактное обозначение набора ячеек с данными, логически объединенных с целью обеспечения единого представления и поведения с точки зрения предметной области |
| Отчетный период | Период времени, за который учреждение сдает определенные отчетные формы |
| Пакеты отчетных форм | Набор отчетных форм, которые заполняются учреждениями в определенном отчетном периоде |
| ПФ | Печатная форма |
| ПФР | Пенсионный фонд Российской Федерации |
| Система | Информационно-аналитическая система «Своды» |
| СУБД | Система управления базами данных |
| Таблица | Структурная единица отчетной формы, используемая для объединения нескольких ячеек, с целью обеспечения смысловой связи между элементами, принадлежащими одному столбцу или одной строке |
| Увязка | Контрольное соотношение, которое используется при проверке правильности заполнения отчетной формы |
| ФИАС | Федеральная информационная адресная система |
| ХО | Хранимый объект |
| Цепочка сдачи отчетности | Структура, задающая иерархию сдачи отчетности среди учреждений |
| ЦС | Целевая статья |
| ЭП | Электронная подпись. Реквизит электронного документа, полученный в результате криптографического преобразования информации с использованием закрытого ключа электронной цифровой подписи |
| ЭФ | Экранная форма |

# Создание и подключение нового экспортера. Ограничение на использование экспорта

Для того чтобы определить свой экспортер, есть два способа:

1. написать реализацию своего экспортера, наследуясь от ЭкспортерДанныхОтчетныхФорм;
2. использовать один из уже имеющихся базовых классов экспортеров в API проекта, для которого необходимо реализовать экспортер.

В первом случае реализацию экспортера необходимо будет писать в API, т.к. подгрузка всех экспортеров происходит из имеющихся сборок API. Во втором случае подразумевается наличие абстрактного экспортера, который используется в реализации экспорта – в таком случае можно просто реализовать абстрактный экспортер внутри формы.

## Создание и подключение нового экспортера

Т.к. экспортеры подгружаются из уже имеющихся сборок API, то и определять их надо в API.

Порядок действий:

* создать в API класс-наследник от ЭкспортерДанныхОтчетныхФорм;
* переопределить обязательные виртуальные методы (т.к. эти методы не абстрактные, то тут среда разработки не подскажет. Ошибка упадет только при запуске экспорта);
* задать наименование экспортера и реализовать логику экспорта;
* перекрыть методы ЭкспортерПодходитДляФормы/ЭкспортерПодходитДляФормыФК.

Ниже этот процесс рассмотрен на примере ExcelTemplateExport из ApiBI.

### Создание в API класса-наследника от ЭкспортерДанныхОтчетныхФорм

Для определения экспортера необходимо создать класс наследник от ЭкспортерДанныхОтчетныхФорм. Этот класс находится в сборке BARS.Svody.DataForm по пути: ./ИмпортЭкспортДанных/ЭкспортерЖанныхОтчетныхФорм.cs/

Пример создания класса наследника:

|  |
| --- |
| internal class ExcelTemplateExport : ЭкспортерДанныхОтчетныхФорм  {  public ExcelTemplateExport(БазоваяОтчетнаяФормаДанных форма) : base(форма) { }  } |

### Переопределение обязательных виртуальных методов

Необходимо переопределить некоторые виртуальные методы родительского класса, которые являются обязательными.

Обязательные методы:

|  |
| --- |
| protected virtual bool ПроверкаПараметров();  protected virtual bool ПредОбработка();  protected virtual bool ВыполнитьЭкспорт();  protected virtual bool ВыполнитьЭкспорт(object экспортерФормы); |

Т.к. это виртуальные, а не абстрактные методы, то среда разработки не подскажет, что эти методы обязательны для переопределения. А значит, и ошибка упадет только при вызове этих методов. Реализация методов в ЭкспортерДанныхОтчетныхФорм.

Обязательные для переопределения методы:

|  |
| --- |
| protected virtual bool ПроверкаПараметров()  {  throw new Exception(ResourcesStrings.МетодПроверкиПараметровДолженБытьПереопределенВПотомках);  }    protected virtual bool ПредОбработка()  {  throw new Exception(ResourcesStrings.МетодПодготовкиЭкспортаДолженБытьПереопределенВПотомках);  }    protected virtual bool ВыполнитьЭкспорт()  {  throw new Exception(ResourcesStrings.МетодВыполненияЭкспортаДолженБытьПереопределенВПотомках);  }    protected virtual bool ВыполнитьЭкспорт(object экспортерФормы)  {  throw new Exception(ResourcesStrings.МетодВыполненияЭкспортаДолженБытьПереопределенВПотомках);  } |

Пример из переопределения обязательных методов в классе ExcelTemplateExport

Для того чтобы задать наименование экспорту, необходимо переопределить метод «Наименование».

Пример переопределения обязательных методов:

|  |
| --- |
| protected override bool ПроверкаПараметров()  {  return ((Форма != null) && (Форма.Данные != null));  }    protected override bool ПредОбработка()  {  return true;  }    protected override bool ВыполнитьЭкспорт()  {  var exporter = GetExporterFromForm();  var reportPath = exporter.CreateTemplate(Форма.Данные, Каталог);    ВыгруженныеФайлы.Add(reportPath);    return true;  }    protected override bool ВыполнитьЭкспорт(object экспортерФормы)  {  return ВыполнитьЭкспорт();  } |

### Задание наименования экспортера и реализация логики экспорта

Пример из ExcelTemplateExport – переопределения метода «Наименование»:

|  |
| --- |
| public override string Наименование()  {  return "Экспорт шаблона для заполнения данных";  } |

Для реализации логики обычно переопределяют «ВыполнитьЭкспорт», чтобы сохранить логику в методе «Экспортировать».

Базовая логика «Экспортировать»:

|  |
| --- |
| public virtual bool Экспортировать()  {  var beforeExportResult = BeforeExport(out var shoudContinue);  if (!shoudContinue)  {  return beforeExportResult;  }    if (this.Экспортер != null)  {  if (!this.ВыполнитьЭкспорт(this.Экспортер))  {  return false;  }  }  else  {  if (!this.ВыполнитьЭкспорт())  {  return false;  }  }  AfterExport();  return true;  } |

Но метод «Экспортировать» также можно переопределить. Ведь именно с него и начнется выполнение экспорта в методе ExportForm класса FormDataExport, если не найдены будут экспортеры формы.

Пример реализации из ExcelTemplateExport:

|  |
| --- |
| protected override bool ВыполнитьЭкспорт()  {  var exporter = GetExporterFromForm();  var reportPath = exporter.CreateTemplate(Форма.Данные, Каталог);    ВыгруженныеФайлы.Add(reportPath);    return true;  }    protected override bool ВыполнитьЭкспорт(object экспортерФормы)  {  return ВыполнитьЭкспорт();  }    private BaseExcelTemplateExporter GetExporterFromForm()  {  var formMeta = МенеджерПриложения.МенеджерМетаописаний.Получить(Форма.ИдентификаторДанныхФормы.ИдентификаторМетаописанияФормы);  var formAsembly = Assembly.GetAssembly(formMeta.ОбработчикФормы.GetType());  var baseExporterType = typeof(BaseExcelTemplateExporter);  var form = (ОтчетнаяФормаДанных)Форма;  var subtype = formAsembly.GetTypes().FirstOrDefault(t => t.IsSubclassOf(baseExporterType));    return subtype == null  ? null  : (BaseExcelTemplateExporter)Activator.CreateInstance(subtype);  } |

## Ограничение на использование экспорта

Для ограничения использования экспорта используется метод статический ЭкспортерПодходитДляФормы/ЭкспортерПодходитДляФормыФК.

### Перекрытие методов «ЭкспортерПодходитДляФормы»/ «ЭкспортерПодходитДляФормыФК»

Методы «ЭкспортерПодходитДляФормы» и «ЭкспортерПодходитДляФормыФК» отличаются только наименованием и типом параметра на вход.

В базовой реализации эти методы возвращают значение «false», и если их не перекрыть, то экспортер просто не будет отображаться для формы. А из-за того, что методы статические их необходимо именно перекрыть:

|  |
| --- |
| public static bool ЭкспортерПодходитДляФормы(string идентификаторФормы)    public static bool ЭкспортерПодходитДляФормыФК(БазоваяОтчетнаяФормаДанных форма) |

Пример перекрытия метода ЭкспортерПодходитДляФормыФК в класе ExcelTemplateExport:

|  |
| --- |
| public new static bool ЭкспортерПодходитДляФормыФК(БазоваяОтчетнаяФормаДанных форма)  {  var formMeta = МенеджерПриложения.МенеджерМетаописаний.Получить(форма.ИдентификаторДанныхФормы.ИдентификаторМетаописанияФормы);  var formAssembly = Assembly.GetAssembly(formMeta.ОбработчикФормы.GetType());  if (formAssembly == null)  {  return false;  }    var exporterType = typeof(BaseExcelTemplateExporter);    return formAssembly.GetTypes().Any(t => t.IsSubclassOf(exporterType));  } |

Пример перекрытия метода «ЭкспортерПодходитДляФормы» в классе «ЭкспортерДанныхОтчетныхФормВСКИФ»:

|  |
| --- |
| public new static bool ЭкспортерПодходитДляФормы (string идентификаторФормы)  {  string[] formList = [...]  return formList.Contains(идентификаторФормы);  } |

**Примечание** – Для того чтобы показать, что перекрытие осмысленное, лучше использовать ключевое слово «new».

### Использование уже имеющейся реализации экспорта с абстрактным классом экспортера

В примере «ЭкспортерПодходитДляФормыФК» есть проверка на наличие реализации абстрактного класса экспортера «BaseExcelTemplateExporter» внутри формы. Это сделано для того, чтобы не выносить специфичную для какой-либо формы логику в API, а оставить реализацию внутри формы.

В таком случае для использования экспорта достаточно будет реализовать внутри формы класс экспортера.

# Создание и подключение нового импортера. Ограничение на использование Импорта

Импортер определяется так же, как и экспортер. Для того чтобы определить свой импортер, есть два способа:

1. написать реализацию своего импортера, наследуясь от ИмпортерДанныхОтчетныхФорм;
2. использовать один из уже имеющихся базовых классов импортеров в API проекта, для которого необходимо реализовать импортер.

В первом случае реализацию импортера необходимо будет писать в API, т.к. подгрузка всех импортеров происходит из имеющихся сборок API. Во втором же подразумевается наличие абстрактного импортера, который используется в реализации импорта – в таком случае можно просто реализовать абстрактный импортер внутри формы.

## Создание и подключение нового импортера

Т.к. импортеры подгружаются из уже имеющихся сборок API, то и определять их надо в API. Порядок действий:

* создать в API класс-наследник от ИмпортерДанныхОтчетныхФорм;
* переопределить обязательные виртуальные методы (т.к. эти методы не абстрактные, то тут среда разработки не подскажет. Ошибка упадет только при запуске экспорта);
* перекрыть методы «ИмпортерПодходитДляФормы»/ «ИмпортерПодходитДляФормыФК»;
* задать наименование импортера и реализовать логику экспорта.

Ниже этот процесс рассмотрен на примере «ИмпортДанныхОтчетныхФормИзФК» из ApiBI.

### Создание в API класса-наследника от ИмпортерДанныхОтчетныхФорм

Для определения импортера необходимо создать класс-наследник от ИмпортерДанныхОтчетныхФорм. Этот класс находится в сборке BARS.Svody.DataForm по пути: \ИмпортЭкспортДанных\ИмпортДанных\ИмпортДанныхОтчетныхФорм.cs.

|  |
| --- |
| public class ИмпортДанныхОтчетныхФормИзФК : ИмпортерДанныхОтчетныхФорм  {  public ИмпортДанныхОтчетныхФормИзФК(БазоваяОтчетнаяФормаДанных форма) : base(форма) {}  } |

### Переопределение обязательных виртуальных методов

Необходимо переопределить некоторые виртуальные методы родительского класса, которые являются обязательными:

|  |
| --- |
| protected virtual bool ПроверкаПараметров();  protected virtual bool ПредОбработка();  protected virtual bool ВыполнитьИмпорт();  protected virtual bool ВыполнитьИмпорт(object импортерФормы); |

Т.к. это виртуальные, а не абстрактные методы, то среда разработки не подскажет, что эти методы обязательны для переопределения. А значит, и ошибка упадет только при вызове этих методов. Реализация методов в ИмпортерДанныхОтчетныхФорм:

|  |
| --- |
| protected virtual bool ПроверкаПараметров()  {  throw new Exception(ResourcesStrings.МетодПроверкиПараметровДолженБытьПереопределенВПотомках);  }    protected virtual bool ПредОбработка()  {  throw new Exception(ResourcesStrings.МетодПодготовкиИмпортаДолженБытьПереопределенВПотомках);  }    protected virtual bool ВыполнитьИмпорт()  {  throw new Exception(ResourcesStrings.МетодВыполненияИмпортаДолженБытьПереопределенВПотомках);  }    protected virtual bool ВыполнитьИмпорт(object импортерФормы)  {  throw new Exception(ResourcesStrings.МетодВыполненияИмпортаДолженБытьПереопределенВПотомках);  } |

Пример из переопределения обязательных методов в классе «ИмпортДанныхОтчетныхФормИзФК»:

|  |
| --- |
| protected override bool ПроверкаПараметров()  {  if (списокФайлов == null)  {  списокФайлов = ВыборФайлаДляВеба();  }    if(списокФайлов == null)  {  return ((Форма != null) && (Форма.Данные != null) && File.Exists(ИмяФайла));  }    return true;  }    protected override bool ПредОбработка()  {  if( АвтоИмпорт )  {  return true;  }    return false;  }    protected override bool ВыполнитьИмпорт()  {  БазовыйИмпортерФК импортер = ПолучитьИмпортер();  if( импортер == null )  {  return false;  }    return импортер.Импорт();  }    protected override bool ВыполнитьИмпорт(object импортер)  {  базовыйИмпортерФК = (БазовыйИмпортерФК)импортер;  return ВыполнитьИмпорт();  } |

### Задание наименования импортера и реализация логики импорта

Для того чтобы задать наименование импортера, необходимо переопределить метод «Наименование».

Пример из ИмпортДанныхОтчетныхФормИзФК:

|  |
| --- |
| public override string Наименование()  {  return "Импорт данных из формата ФК";  } |

Для реализации логики обычно переопределяют «Импортировать»/ «ИмпортироватьAsync» или «ВыполнитьИмпорт»:

|  |
| --- |
| public virtual Task<bool> ИмпортироватьAsync(object импортерФормы)  public virtual bool Импортировать(object импортерФормы)  public virtual bool Импортировать(string путь)  public virtual bool ИмпортироватьИзФайла(string имяФайлаДляИмпорта)  public virtual bool ИмпортироватьИзКаталога(string каталогДляИмпорта)  protected virtual bool ВыполнитьИмпорт()  protected virtual bool ВыполнитьИмпорт(object импортерФормы) |

Но метод «Импортировать» также можно переопределить. Ведь именно с него и начнется выполнение импорта в классе «FormDataImport», если не найдены будут импортеры формы.

Пример реализации из «ВыполнитьИмпорт» был приведен выше.

## Ограничение на использование импортера

Для ограничения использования импортера используется метод статический «ИмпортерПодходитДляФормы»/ «ИмпортерПодходитДляФормыФК».

### Перекрытие методов ИмпортерПодходитДляФормы/ИмпортерПодходитДляФормыФК

Отличаются методы только наименованием и типом параметра на вход.

В базовой реализации эти методы возвращают значение «false», и если их не перекрыть, то импортер просто не будет отображаться для формы. А из-за того, что методы статические, их необходимо именно перекрыть:

|  |
| --- |
| public static bool ИмпортерПодходитДляФормы(string идентификаторФормы)  public static bool ИмпортерПодходитДляФормыФК(БазоваяОтчетнаяФормаДанных форма) |

Пример перекрытия метода «ИмпортерПодходитДляФормыФК» в классе «ИмпортДанныхОтчетныхФормИзФК»:

|  |
| --- |
| public new static bool ИмпортерПодходитДляФормыФК(БазоваяОтчетнаяФормаДанных форма)  {  IМетаописаниеФормы метаописаниеФормы = МенеджерПриложения.МенеджерМетаописаний.Получить(форма.ИдентификаторДанныхФормы.ИдентификаторМетаописанияФормы);    if (метаописаниеФормы == null)  {  return false;  }    Assembly внутренниеТипыОбработчикаФорм = Assembly.GetAssembly(метаописаниеФормы.ОбработчикФормы.GetType());  Type типБазовыйИмпортерФК = typeof(БазовыйИмпортерФК);    foreach (Type внутреннийТип in внутренниеТипыОбработчикаФорм.GetTypes())  {  if (внутреннийТип.IsSubclassOf(типБазовыйИмпортерФК))  {  return true;  }  }  return false;  } |

Пример перекрытия метода ИмпортерПодходитДляФормы:

|  |
| --- |
| public new static bool ИмпортерПодходитДляФормы (string идентификаторФормы)  {  string[] formList = [...]  return formList.Contains(идентификаторФормы);  } |

Для того чтобы показать, что перекрытие осмысленное, лучше использовать ключевое слово «new».

### \*Использование уже имеющейся реализации импорта с абстрактным классом импортера

В примере «ИмпортерПодходитДляФормыФК» есть проверка на наличие реализации абстрактного класса импортера «БазовыйИмпортерФК» внутри формы. Это сделано для того, чтобы не выносить специфичную для какой-либо формы логику в API, а оставить реализацию внутри формы.

В таком случае для использования импорта достаточно будет реализовать внутри формы класс импортера.

# Калькулятор Условных Атрибутов

В Дизайнере существует возможность создавать форматы отображения ЭФ.

Если функционала форматов в Дизайнере недостаточно, можно реализовать логику в форме. Для этого необходимо переопределить событие «ПолучитьКалькуляторУсловныхАтрибутов». В событии возвращается класс-наследник от «КалькуляторУсловныхАтрибутов».

|  |
| --- |
| public override IКалькуляторУсловныхАтрибутов ПолучитьКалькуляторУсловныхАтрибутов()  {  return new Калькулятор(this);  }    public class Калькулятор : IКалькуляторУсловныхАтрибутов  {  private БазовыйОбработчикФормы обработчикФормы;  private ДанныеФормы данныеФормы;    public Калькулятор(БазовыйОбработчикФормы обработчикФормы)  {  this.обработчикФормы = обработчикФормы;  данныеФормы = обработчикФормы.Форма;  }    public РезультатРасчета Вычислить()  {  var результат = new РезультатРасчета();    var условныйФормат = new УсловныйФормат  {  Приоритет = 15000  };    var формат = new Формат  {  Наименование = "Перекрашиваем"  };    var допАтрибут = new ДополнительныйАтрибут  {  Наименование = "background-color",  Значение = "#dad3f3"  };  формат.СписокАтрибутов.Add(допАтрибут);    var списокФорматов = new List<ФорматДляОбласти>  {  new ФорматДляОбласти(условныйФормат, формат, new ОбластьПрименимостиФорма())  };    var идентификаторЯчейки = new ИдентификаторЭлементаФормы  {  КодТаблицы = "КодТаблицы",  КодСтроки = "КодСтроки",  КодСтолбца = "КодСтолбца"  };  результат.Добавить(идентификаторЯчейки, списокФорматовБлокировки);    return результат;  }    public void Очистить()  {  this.обработчикФормы = null;  }  } |

Метод «Вычислить» вызывается при открытии формы, при добавлении и удалении строки, открытии и сохранении субтаблицы, при песчете автоблоков (только если данные формы перед нажатием были изменены).

При реализации калькулятора используются следующие классы (см. п. 3.1 – 3.6).

## ДополнительныйАтрибут

Свойства:

* Наименование. Тип string. Параметр ячейки, который нужно изменить. Таблицу с допустимыми наименованиями можно посмотреть в инструкции по работе с Дизайнером отчетных форм (п. «Правила условного атрибутирования»);
* Значение. Тип object. В зависимости от наименования укажите подходящее значение. Например, цвет заливки ячейки. Примеры значений так же в таблице с допустимыми наименованиями;
* Формула. Тип string. Остается всегда пустой, на данном этапе она не доработана.

Методы:

* ReadXml(XmlReader reader). Присваивает значения из тегов «Наименование», «Формула», «Значение» в соответствующие свойства дополнительного атрибута.
* WriteXml(XmlWriter writer). Записывает значения в теги «Наименование», «Формула», «Значение» из соответствующих свойств дополнительного атрибута;
* ПолучитьЗначение(БазовыйОбработчикПеременных обработчикПеременных, IОбработчикВыражений обработчикВыражений). Если заполнено свойство формула, то рассчитывает значение по этой формуле. Возвращает object.

## Формат

Свойства:

* Наименование. Тип string. Уникальное наименование формата ячейки;
* СписокАтрибутов. Тип List<ДополнительныйАтрибут>. Указываем все атрибуты, которые необходимо применить к ячейке.

Методы:

* ReadXml(XmlReader reader). Присваивает значения из тегов «Наименование», «СписокАтрибутов» в соответствующие свойства дополнительного атрибута;
* WriteXml(XmlWriter writer). Записывает значения в теги «Наименование», «СписокАтрибутов» из соответствующих свойств дополнительного атрибута.

## УсловныйФормат

Свойства:

* Приоритет. Тип int. Приоритет действия текущего формата над другими форматами, на случай если они будут пересекаться. Работает так же и над платформенными форматами;
* ПредикатПрименимости. Тип string. Укажите, в каких случаях необходимо изменить формат ячейки. Допустимые предикаты можно посмотреть в инструкции по работе с Дизайнером отчетных форм (п. «Правила условного атрибутирования»);
* НаименованиеФормата. Тип string. Уникальное наименование при использовании более одного условного формата.

Методы:

* ReadXml(XmlReader reader). Присваивает значения из тегов «Приоритет», «ПредикатПрименимости», «НаименованиеФормата» в соответствующие свойства дополнительного атрибута;
* WriteXml(XmlWriter writer). Записывает значения в теги «Приоритет», «ПредикатПрименимости», «НаименованиеФормата» из соответствующих свойств дополнительного атрибута.

## ФорматДляОбласти

Свойства:

* УсловныйФормат. Тип УсловныйФормат. Описание класса выше;
* Формат. Тип Формат. Описание класса выше;
* ОбластьПрименимости. Тип ОбластьПрименимости. Укажите объект класса применимости. Допустимые классы: ОбластьПрименимостиФорма, ОбластьПрименимостиТаблица, ОбластьПрименимостиЯчейка.

## ИдентификаторЭлементаФормы

Свойства:

* КодТаблицы. Тип string. Укажите код таблицы;
* КодСтолбца. Тип string. Укажите код столбца;
* КодСтроки. Тип string. Укажите код строки;
* КодСвободнойЯчейки. Тип string. Укажите код свободной ячейки.

Методы:

* РазобратьСтроку(string строковоеПредставлениеПеременной, out string кодТаблицы, out string кодСтолбца, out string кодСтроки). Разбирает строковое значение на код таблицы, столбца и строк через символ ':'. Возвращает true/false, в зависимости от успеха операции;
* ПолучитьХешКод(). Получает хэш объекта. Возвращает int;
* Сравнить(object элемент). Приводит объект к типу ИдентификаторЭлементаФормы и сравнивает их.

## РезультатРасчета

Свойства:

* СловарьСписковФорматовПоИдентификаторам. Тип Dictionary<ИдентификаторЭлементаФормы, List<ФорматДляОбласти>>. Доступен только для чтения.

Методы:

* Добавить(ИдентификаторЭлементаФормыидентификатор, List<ФорматДляОбласти> списокФорматов). Добавляет словарь форматов, которые необходимо применить к указанному элементу формы;
* Добавить(РезультатРасчета результатРасчета). Добавляет СловарьСписковФорматовПоИдентификаторам из указанного результата расчета в текущий.

# Переопределение базового обработчика сводной формы, добавление записей в историю сборки сводной

Для переопределения базового обработчика сводной необходимо переопределить метод «ПолучитьОбработчикСводнойФормы»в классе «БазовыйОбработчикФормы» и вернуть экземпляр класса, который наследуется от класса «БазовыйОбработчикСводнойФормы».

Переопределение базового обработчика сводной формы:

|  |
| --- |
| public override БазовыйОбработчикСводнойФормы ПолучитьОбработчикСводнойФормы(ОтчетнаяФормаДанных своднаяФорма)  {  return new SummaryFormHandler(своднаяФорма);  }    public class SummaryFormHandler : БазовыйОбработчикСводнойФормы  {  public SummaryFormHandler(ОтчетнаяФормаДанных своднаяФорма) : base(своднаяФорма)  {  }  } |

Пример готового переопределенного обработчика сводной:

|  |
| --- |
| public override БазовыйОбработчикСводнойФормы ПолучитьОбработчикСводнойФормы(ОтчетнаяФормаДанных СводнаяФорма)  {  return new SummaryFormHandler(СводнаяФорма);  }    public class SummaryFormHandler : БазовыйОбработчикСводнойФормы  {  private List<ЗаписьИсторииСборкиИтоговыхОтчетов> \_historyRecordList;    public SummaryFormHandler(ОтчетнаяФормаДанных своднаяФорма) : base(своднаяФорма)  {  \_historyRecordList = new List<ЗаписьИсторииСборкиИтоговыхОтчетов>();  }    protected override bool СворачиватьСтрокиДинамическихТаблиц  {  get { return false; }  }    protected override void ОбработатьНачалоОбработкиТаблицы(АргументыОбработчикаСводнойФормы args)  {  var sourceTable = args.ТаблицаИсточника;  var receiverTable = args.ТаблицаПриемника;    if (receiverTable.КодТаблицы != "Субъекты\_РФ")  {  base.ОбработатьНачалоОбработкиТаблицы(args);  return;  }    foreach (var row in sourceTable.МатрицаЗначений.Строки.Values)  {  var cellDataSource = row["субъект"].ЗначениеСтрокой;    if (string.IsNullOrEmpty(cellDataSource))  {  continue;  }    var receiverRowData = receiverTable.МатрицаЗначений.Строки.Values.FirstOrDefault(r => r["субъект"].ЗначениеСтрокой == cellDataSource);    if (receiverRowData != null)  {  row.Ячейки.Ячейки.Values  .Where(x => x.Значение is TableReference)  .ToList()  .ForEach(x => MoveModalTableData(x, receiverRowData.КодСтроки, args.ДанныеИсточника, args.ДанныеПриемника, receiverRowData, false));  }  else  {  var rowCode = receiverTable.МатрицаЗначений.Строки.Count.ToString();  var receiverRow = new СтрокаДанных() { КодСтроки = rowCode };  receiverRow.Ячейки = row.Ячейки;  receiverTable.МатрицаЗначений.Добавить(receiverRow);  row.Ячейки.Ячейки.Values  .Where(x => x.Значение is TableReference)  .ToList()  .ForEach(x => MoveModalTableData(x, rowCode, args.ДанныеИсточника, args.ДанныеПриемника, receiverRow, true));  }  }    args.ОтменитьДальнейшуюОбработку = true;  }    private void MoveModalTableData(ЯчейкаДанных cell, string rowCode, ДанныеФормы sourceForm, ДанныеФормы receiverForm, СтрокаДанных receiverRow, bool newString)  {  var history = new List<HistoryCell>();  var sourceTableCode = cell.Значение.ЗначениеСтрокой;  var columnCodeWithSubTable = cell.КодЯчейки;  var templateTableCode = "1\_АЭ";  var sourceTable = sourceForm[sourceTableCode];    if (sourceTable != null)  {  ТаблицаДанных receiver;    if (newString)  {  receiver = СводнаяФорма.CreateModalTable("Субъекты\_РФ", rowCode, cell.КодЯчейки, templateTableCode);  }  else  {  var modalTableCode = receiverRow[columnCodeWithSubTable].ЗначениеСтрокой;  receiver = receiverForm[modalTableCode];  }    if (receiver != null)  {  foreach (var row in sourceTable.МатрицаЗначений.Строки.Values)  {  foreach (var col in row.ТаблицаДанных.СтолбцыМетаструктуры.Values)  {  var columnCode = col.Код;  var historyCell = history.FirstOrDefault(h => h.RowCode == row.КодСтроки && h.ColumnCode == columnCode);  var value = sourceTable[row.КодСтроки, col.Код].ЗначениеЧисло;  receiver[row.КодСтроки, col.Код].ЗначениеЧисло += value;    if (historyCell != null)  {  historyCell.Value += value;  }  else  {  history.Add(  new HistoryCell  {  ColumnCode = columnCode, RowCode = row.КодСтроки,  Value = value  });  }  }  }    AddNewRecordHistory(receiver, history);  }  }  }    protected override void ОбработатьОкончаниеСборки(АргументыОбработчикаСводнойФормы args)  {  ИсторияСборки.ДобавитьЗаписи(\_historyRecordList);  \_historyRecordList.Clear();  }    private void AddNewRecordHistory(ТаблицаДанных receiverTable, List<HistoryCell> history)  {  foreach (var receiverRow in receiverTable.МатрицаЗначений.Строки.Values)  {  var historyRows = history.Where(h => h.RowCode == receiverRow.КодСтроки).ToList();    var recordHistory = new ЗаписьИсторииСборкиИтоговыхОтчетов();    recordHistory.КодТаблицыИсточника = "";  recordHistory.КодТаблицыПриемника = receiverTable.КодТаблицы;  recordHistory.КодСтрокиИсточника = receiverRow.КодСтроки;  recordHistory.КодСтрокиПриемника = receiverRow.КодСтроки;  recordHistory.НаименованиеУчрежденияИсточника = ИсторияСборки.НаименованиеУчрежденияИсточника;  recordHistory.НаименованиеУчрежденияПриемника = ИсторияСборки.НаименованиеУчрежденияПриемника;  recordHistory.ИдентификаторДанныхИсточника = ИсторияСборки.ИдентификаторДанныхИсточника;  recordHistory.ИдентификаторДанныхПриемника = ИсторияСборки.ИдентификаторДанныхПриемника;  recordHistory.НаименованиеФормыИсточника = ИсторияСборки.НаименованиеФормыИсточника;  recordHistory.НаименованиеФормыПриемника = ИсторияСборки.НаименованиеФормыПриемника;    foreach (var historyRow in historyRows)  {  var columns = $"{historyRow.ColumnCode}{ИсторияСборкиИтоговыхОтчетов.РазделительПолейНовый}{historyRow.ColumnCode}";    if (!recordHistory.Значения.ContainsKey(columns))  {  recordHistory[columns] = historyRow.Value;  }  }    \_historyRecordList.Add(recordHistory);  }  }    public class HistoryCell  {  public string RowCode { get; set; }    public string ColumnCode { get; set; }    public decimal Value { get; set; }  }  } |

В п. 4.1 «Сбор сводной в Сводах» объясняется, что такое сводная, как получать пользовательские параметры, формировать список исходных форм, рассказывается про сведение ячеек, строк, статических и динамических таблиц, а также приведен список методов, которые переопределяются в потомках.

Таблица 1 – Не переопределяемые методы класса БазовыйОбработчикСводнойФормы

| Название | Принимаемые параметры | Возвращаемый тип | Действие |
| --- | --- | --- | --- |
| СохранитьИсториюСборки | (БазоваяОтчетнаяФормаДанных сохраняемаяФорма, ИсторияСборкиИтоговыхОтчетов история) | void | Сохраняет историю сборки в БД, далее очищает список записей истории из параметра. Если идентификатор данных приемника пустой, записи истории очистятся, а запись в БД не произойдет |
| ПолучитьСписокИсходныхФорм | (АргументыОбработчикаСводнойФормы аргументы) | List<ИдентификаторДанныхФормы> | Возвращает список исходных форм, в зависимости от аргумента |
| СвестиИсходныеФормы | (List<ИдентификаторДанныхФормы> списокИсходныхФорм, ЗначенияПоляУчрежденияОтображаемогоВЦепочкеСдачиОтчетности сellItemNameSourceProperty) | void | Сводит данные исходных форм в сводную форму |
| ПолучитьСписокОшибок |  | List<СтрокаОтчетаСверкиДанных> | Возвращает список ошибок сверки классификаторов |
| СборкаСводнойФормыДоступна |  | bool | Проверяет возможность сборки сводной формы: true в случае, если сборка сводной формы доступна для текущего рабочего места |

Для добавления записей в историю сборки используется свойство «ИсторияСборки»типа «ИсторияСборкиИтоговыхОтчетов» из класса «БазовыйОбработчикСводнойФормы».

Чтобы установить форму приемник для записей истории, необходимо заполнить три свойства:

|  |
| --- |
| ИсторияСборки.ИдентификаторДанныхПриемника = ХранилищеДанныхФорм.ПолучитьИдентификаторХранимыхДанных(СводнаяФорма.Данные.Идентификатор);  ИсторияСборки.НаименованиеУчрежденияПриемника = СводнаяФорма.НаименованиеУчреждения;  ИсторияСборки.НаименованиеФормыПриемника = "Сводная форма"; |

При использовании свойства «ИсторияСборки»эти свойства заполняются сами при срабатывании метода «СвестиИсходныеФормы»в классе «БазовыйОбработчикСводнойФормы».

Методы класса «ИсторияСборкиИтоговыхОтчетов» для создания объектов записи истории:

|  |
| --- |
| public IЗаписьИсторииСборкиИтоговыхОтчетов ДобавитьЗапись(  string кодСтрокиПриемника,  string кодСтрокиИсточника,  string кодСтолбцаПриемника,  string кодСтолбцаИсточника,  object значение);    foreach (var historyRecord in historyRecordList)  {  ИсторияСборки.ДобавитьЗапись(historyRecord.receiverRowCode, historyRecord.sourceRowCode, historyRecord.receiverColumnCode, historyRecord.sourceColumnCode, historyRecord.value);  } |

В случае дублирующихся строк истории их значения суммируются.

* «кодСтрокиПриемника» – код строки таблицы формы-приемника, в которую собирается итоговый отчет;
* «кодСтрокиИсточника» – код строки таблицы формы-источника, данные которой попадают в итоговый отчет;
* «кодСтолбцаПриемника» – код столбца таблицы формы-приемника, в который собирается итоговый отчет;
* «кодСтолбцаИсточника» – код столбца таблицы формы-источника, данные которого попадают в итоговый отчет;
* «значение» – значение, которое попадает в итоговый отчет.

Возвращается созданный объект записи истории, который был добавлен в список записей текущей истории.

|  |
| --- |
| public IЗаписьИсторииСборкиИтоговыхОтчетов ДобавитьЗапись(  string кодСтроки,  string кодСтолбца,  object значение);    if (столбецМетаструктуры.Код != null && значениеЯчейки.Значение != null)  {  if (МенеджерБД.МенеджерИнициализирован)  {  ИсторияСборки.ДобавитьЗапись(строкаМетаструктуры, столбецМетаструктуры.Код, значениеЯчейки.Значение);  }  } |

Перегрузка вызывает метод «ДобавитьЗапись»*,* который описан выше, в качестве аргументов «кодСтрокиПриемника», «кодСтрокиИсточника» и «кодСтолбцаПриемника», «кодСтолбцаИсточника» передает одинаковые значения.

При наличии дублирующихся строк истории их значения суммируются.

Метод «ДобавитьЗапись» используется в случае одинаковой метаструктуры формы-приемника и формы-источника.

* «кодСтроки» – код строки таблиц формы-приемника и формы-источника;
* «кодСтолбца» – код столбца таблиц формы-источника и формы-источника;
* «значение» – значение, которое попадает в итоговый отчет.

Возвращается созданный объект записи истории, который был добавлен в список записей текущей истории.

|  |
| --- |
| public IЗаписьИсторииСборкиИтоговыхОтчетов AddOrCreateOlapHistoryRecord(  IList<string> chainPath,  string кодСтроки,  string кодСтолбца,  object значение);    protected override void ОбработатьОкончаниеСборки(АргументыОбработчикаСводнойФормы args)  {  ИсторияСборки.AddOrCreateOlapHistoryRecord(chainPath, rowCode, columnCode, value);  } |

В случае дублирующихся строк истории их значения суммируются.

* «chainPath» – иерархия наименований элементов цепочки до формы, по которой создается запись истории;
* «кодСтроки» – код строки таблиц формы-приемника и формы-источника;
* «кодСтолбца» – код столбца таблиц формы-источника и формы-источника;
* «значение» – значение, которое попадает в итоговый отчет.

Возвращается добавленный объект записи истории, который содержит данные об иерархии цепочки сдачи.

Методы класса «ИсторияСборкиИтоговыхОтчетов» для добавления объектов записи истории:

|  |
| --- |
| public IЗаписьИсторииСборкиИтоговыхОтчетов ДобавитьЗапись(IЗаписьИсторииСборкиИтоговыхОтчетов запись); |

* «запись» – объект записи истории сборки;

Добавляет переданный объект записи истории в список записей текущей истории и возвращает его.

|  |
| --- |
| public void ДобавитьЗаписи(IList<IЗаписьИсторииСборкиИтоговыхОтчетов> записи); |

* «записи» – список объектов записей истории сборки.

Добавляет переданный список объектов записей истории в список записей текущей истории:

|  |
| --- |
| public void ДобавитьЗаписи(IEnumerable<IЗаписьИсторииСборкиИтоговыхОтчетов> записи); |

* «записи» – Список объектов записей истории сборки.

Добавляет переданный список объектов записей истории в список записей текущей истории:

|  |
| --- |
| public void ДобавитьЗаписи(List<ЗаписьИсторииСборкиИтоговыхОтчетов> записи); |

* «записи» – список объектов записей истории сборки.

Добавляет переданный список объектов записей истории в список записей текущей истории.

Объекты записей истории можно создавать и заполнять вручную, затем добавлять через имеющиеся методы. Пример:

|  |
| --- |
| private List<ЗаписьИсторииСборкиИтоговыхОтчетов> \_historyRecordList;    protected override void ОбработатьОкончаниеСборки(АргументыОбработчикаСводнойФормы args)  {  ИсторияСборки.ДобавитьЗаписи(\_historyRecordList);  \_historyRecordList.Clear();  }    private void GenerateHistoryRecord(ДанныеФормы sourceForm,  ДанныеФормы receiverForm,  string sourceTableCode,  string sourceRowCode,  string columnCode,  decimal value,  string receiverTableCode,  string receiverRowCode,  string receiverColumnCode)  {  var recordHistory = new ЗаписьИсторииСборкиИтоговыхОтчетов();    recordHistory.КодТаблицыИсточника = sourceTableCode;  recordHistory.КодТаблицыПриемника = receiverTableCode;  recordHistory.КодСтрокиИсточника = sourceRowCode;  recordHistory.КодСтрокиПриемника = receiverRowCode;  recordHistory.НаименованиеУчрежденияИсточника = sourceForm.Идентификатор.Учреждение.Наименование;  recordHistory.НаименованиеУчрежденияПриемника = receiverForm.Идентификатор.Учреждение.Наименование;  recordHistory.ИдентификаторДанныхИсточника = sourceForm.Идентификатор.DataId;  recordHistory.ИдентификаторДанныхПриемника = receiverForm.Идентификатор.DataId;  recordHistory.НаименованиеФормыИсточника = sourceForm.Идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания;  recordHistory.НаименованиеФормыПриемника = receiverForm.Идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания;  var columns = $"{receiverColumnCode}{ИсторияСборкиИтоговыхОтчетов.РазделительПолейНовый}{receiverColumnCode}";  if (!recordHistory.Значения.ContainsKey(columns))  {  recordHistory[columns] = value;  }    \_history.Add(recordHistory);  } |

Таблица 2 – Остальные методы класса «ИсторияСборкиИтоговыхОтчетов»

| Название | Принимаемые параметры | Возвращаемый тип | Действие |
| --- | --- | --- | --- |
| УстановитьИдентификаторДанныхДляВсехЗаписей | (Guid идентификатор, bool идентификаторПриемника) | void | Устанавливает значение свойства *ИдентификаторДанныхПриемника*или *ИдентификаторДанныхИсточника*для текущего объекта истории и для всех записей списка *ЗаписиИстории* в зависимости от второго параметра |
| УстановитьИдентификаторДанныхПриемникаДляВсехЗаписей | (Guid идентификатор) | void | Устанавливает значение свойства *ИдентификаторДанныхПриемника*для текущего объекта истории и для всех записей списка |
| УстановитьИдентификаторДанныхИсточникаДляВсехЗаписей | (Guid идентификатор) | void | Устанавливает значение свойства *ИдентификаторДанныхИсточника*для текущего объекта истории и для всех записей списка |
| Загрузить |  | void | Загружает записи истории в свойство *ЗаписиИстории* из файла истории сборки *ФайлИсторииСборкиИтоговыхОтчетов*, файл ищется в БД по параметрам *ИдентификаторДанныхПриемника, ArchiveDataRecordId, КодТаблицыПриемника* |
| СохранитьВБД |  | void | Формирует и сохраняет в БД объекты *ФайлИсторииСборкиИтоговыхОтчетов* на основе записей текущей истории *ЗаписиИстории* |
| CreateAndSaveArchiveRecords | (Guid formDataId, Guid archiveDataRecordId) | void | Создает архивные записи истории результатов сборки сводной путем клонирования последних актуальных |
| УдалитьИсторию |  | void | Удаляет из БД историю по всем таблицам формы-приемника |
| УдалитьИсторию | (string кодТаблицыПриемника) | void | Удаляет из БД историю по таблице формы-приемника, код которой передан в параметрах |
| ОчиститьИсторию |  | void | Очищает список записей истории *ЗаписиИстории* у текущего объекта истории *ИсторияСборкиИтоговыхОтчетов* |

## Сбор сводной в Сводах

Сбор сводной формирует сводный отчет по нижестоящим учреждениям на своде. Свод формируется на том учреждении, на котором запущена сборка сводной. Базовая (на уровне платформы) сборка сводной сводит свободные ячейки и таблицы нижестоящих учреждений на свод. Есть возможность изменить сборку сводной на прикладном уровне. Для доступа к функции сборки сводной лицензионный ключ должен обладать следующими характеристиками: тип рабочего места – бэк офис, комплектация – базовая.

### Получение параметров и формирование списка форм для сведения

После нажатия кнопки «Собрать сводную форму» в меню отчетной формы происходит проверка лицензионного ключа на соответствие требованиям и формирование параметров сборки сводной. Для формирования параметров сборки сводной используется метод : «ПолучитьСписокПользовательскихПараметровЗапускаСборки».

|  |
| --- |
| protected virtual СписокПользовательскихПараметров ПолучитьСписокПользовательскихПараметровЗапускаСборки(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  } |

В базовой сборке данные параметры пустые, на прикладном уровне существует возможность переопределить этот метод для получения необходимых параметров сборки сводной. Например, тип сборки сводной.

После получения параметров сборки сводной формируется список исходных форм для сведения. Эти формы будут сведены с формой свода. Для формирования списка исходных форм используется метод «СформироватьСписокИсходныхФорм».

|  |
| --- |
| protected virtual List<ИдентификаторДанныхФормы> СформироватьСписокИсходныхФорм(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  } |

Как видно из кода, данный метод в базовой сводной не возвращает ни одной формы. В данном случае формируется базовый список исходных форм. Для этого используется метод ОтчетнойФормыДанных «ПолучитьИсходныеФормы». Данный метод получает все дочерние учреждения для учреждения, на котором собирается свод. Для данных учреждений получите формы, у которых присутствуют хранимые данные, т.е. которые были сохранены хотя бы один раз. Далее пользователю предлагается выбрать формы, на основе которых будет произведена сборка сводной (из числа исходных форм). Выбранные формы записываются в список Исходных Форм. Далее происходит запрос пользовательских параметров. Для запроса пользовательских параметров используется метод «ПолучитьСписокПользовательскихПараметров».

|  |
| --- |
| protected virtual СписокПользовательскихПараметров ПолучитьСписокПользовательскихПараметров(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  } |

По умолчанию данные параметры пустые, но есть возможность переопределить метод получения пользовательских параметров в потомках. Далее происходит сборка сводной.

### Сборка сводной

Очистите сводную форму кроме свободных ячеек формы. Произойдет инициализация менеджера истории сборки сводной. Очистите прошлую историю сборки сводной. Обработайте начало сборки с помощью метода «ОбработатьНачалоСборки», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьНачалоСборки(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

Начните сводить исходные формы со сводной формой. При обработке первой формы все ячейки сводной буду переинициализированы. Все ячейки получат значение по умолчанию. Загрузите исходную форму, сведите сводную и форму источник. Сведение форм: заполните данные источника в истории сборки сводной (идентификатор хранимых данных источника, наименование учреждения, код формы). Обработайте начало обработки источника вызовом метода «ОбработатьНачалоОбработкиИсточника», по умолчанию пустой, переопределяется в потомках, срабатывает перед обработкой каждой исходной формы

|  |
| --- |
| protected virtual void ОбработатьНачалоОбработкиИсточника(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

Сведите свободные ячейки формы (см. выше сведение ячеек). Далее сведите таблицы формы (см. сведение таблиц).

После сведения формы вызывается метод «ОбработатьОкончаниеОбработкиИсточника», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьОкончаниеОбработкиИсточника(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

После сведения всех форм пройдите по таблицам Сводной Формы, вызовите метод «ЗаполнитьЗначенияСправочников». Данный метод загружает значения справочников, используемых в ячейках указанной таблицы данной формы. Сверните строки динамических таблиц аналогично тому, как сворачивались строки в источниках. После обработки таблиц сводной вызывается метод «ОбработатьОкончаниеСборки», по умолчанию содержит метод «ПодсчитатьСреднееЗначениеДляФормы».Данный метод устанавливает значения ячеек, для которых типДействия с полем установлен «Среднее», затем добавляет все записи истории в историю сборки сводной и сохраняет ее.

### Сведение ячеек

Проверьте, что в форме-источнике существует аналогичная ячейка. Имеется значение ячейки сводной и источника. Если ячейка сводной не является ключевой, то сведите две ячейки. Вызовите метод «ОбработатьЯчейку», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьЯчейку(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

Используя статический метод ТаблицыДанных «ОбработатьЗначениеЯчеек»,сведите две ячейки. Если ячейки имеют числовой тип, то результат сведения будет зависеть от настроек ячейки сводной (устанавливается в Дизайнере), если строковый тип – осущетсвляется проверка, что ячейка источника не ключевая и заполнена. Если условие выполняется, то в сводную садится склеенное значение двух ячеек через пробел + разделитель (устанавливается в Дизайнере). Добавьте запись в историю (ячейка, код ячейки, значение).

### Сведение строк

Вызовите метод «ОбработатьСтроку», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьСтроку(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

Получите код строки или хеш код строки (формируется из ключевых столбцов), если таблица динамическая. Пройдите по столбцам строки сводной. Вызывается 3 метода (с разными параметрами) «ОбрабатыватьСтолбец», по умолчанию возвращают значение «True», можно переопределить в потомке.

|  |
| --- |
| protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец)  {  return true;  } |

|  |
| --- |
| protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец, ОтчетнаяФормаДанных отчетнаяФорма)  {  return true;  } |

|  |
| --- |
| protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец, ОтчетнаяФормаДанных отчетнаяФорма, ТаблицаМетаструктуры текущаяТаблица)  {  return true;  } |

Если один из методов вернет значение «false», столбец не будет обработан. Получите ячейку приемника и ячейку источника, обработайте ячейки (см. п. 4.1.3).

### Сведение таблиц форм

Поочередно начните сводить таблицы форм. Берется таблица из сводной формы и проверяется, что данная таблица существует в форме источника, иначе выдается исключение. Запишите код таблицы в историю сборки сводной. Вызовите метод «ОбработатьНачалоОбработкиТаблицы», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьНачалоОбработкиТаблицы(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

Обработайте сводные ячейки таблицы (см. выше). Если свободная ячейка перекрывает ячейку таблицы, то дублируйте запись в истории так же, только вместо кода ячейки добавьте код столбца. Последующая обработка зависит от того, какой является эта таблица: динамической или статической. После обработки таблицы вызывается метод «ОбработатьОкончаниеОбработкиТаблицы», по умолчанию пустой, переопределяется в потомках.

|  |
| --- |
| protected virtual void ОбработатьОкончаниеОбработкиТаблицы(АргументыОбработчикаСводнойФормы Аргументы)  {  } |

### Сведение динамических таблиц форм

Проверьте свойство «СворачиватьСтрокиДинамическихТаблиц», по умолчанию установленое значение «True», может быть переопределено в потомках. Если количество строк > 0, и значение свойства = true, сверните строки таблицы, используя метод таблицы данных «СвернутьСтроки».

Сворачивание: cворачивает строки данных таблицы по ключевым столбцам. Строки, имеющие одинаковые значения в ключевых ячейках, считаются одинаковыми. Одинаковые строки сворачиваются в одну. Значения строковых ячеек дублирующих строк объединяются в одну строку через разделитель (устанавливается в Дизайнере). Поведение числовых ячеек при сворачивании строк определяется параметром «ДействиеСПолем» числового столбца в метаописании формы.

Пройдите по строкам таблицы источника: попытка получить строку сводной из кеша (если строка с таким ключом уже была добавлена). Ключ 1 код таблицы, ключ 2 хеш код ключевых столбцов.

|  |
| --- |
| private Dictionary<string, Dictionary<string, СтрокаДанных>> строкиДинамическихТаблицСводной; |

Если по ключу 2 в кеше нет строк, то формируйте новую строку. Обработайте полученную строку (см. п. 4.1.4).

### Сведение статических таблиц форм

Пройдите по строкам сводной, проверьте, что строка с таким кодом существует в форме источника. Обработайте строку методом «ОбработатьСтроку».

### Методы, переопределяемые в потомках

|  |
| --- |
| protected virtual void ОбработатьНачалоСборки(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual void ОбработатьОкончаниеСборки(АргументыОбработчикаСводнойФормы Аргументы)  {  ПодсчитатьСреднееЗначениеДляФормы();  }    protected virtual void ОбработатьНачалоОбработкиИсточника(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual void ОбработатьОкончаниеОбработкиИсточника(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual void ОбработатьНачалоОбработкиТаблицы(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual void ОбработатьОкончаниеОбработкиТаблицы(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец)  {  return true;  }    protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец, ОтчетнаяФормаДанных отчетнаяФорма)  {  return true;  }    protected virtual bool ОбрабатыватьСтолбец(СтолбецМетаструктуры Столбец, ОтчетнаяФормаДанных отчетнаяФорма, ТаблицаМетаструктуры текущаяТаблица)  {  return true;  }    protected virtual void ОбработатьСтроку(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual void ОбработатьЯчейку(АргументыОбработчикаСводнойФормы Аргументы)  {  }    protected virtual List<ИдентификаторДанныхФормы> СформироватьСписокИсходныхФорм(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  }    protected virtual СписокПользовательскихПараметров ПолучитьСписокПользовательскихПараметров(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  }    protected virtual СписокПользовательскихПараметров ПолучитьСписокПользовательскихПараметровЗапускаСборки(АргументыОбработчикаСводнойФормы Аргументы)  {  return null;  } |

# Сверка классификаторов

Чтобы добавить к платформенным проверкам макросные проверки, переопределите в форме свойство «СверкаКлассификаторов». Проверки будут срабатывать при сохранении формы и таблицы:

|  |
| --- |
| protected СверкаКлассификаторов \_сверкаКлассификаторов;  public override IСверкаКлассификаторов СверкаКлассификаторов  {  get  {  if (\_сверкаКлассификаторов == null)  {  \_сверкаКлассификаторов = new СверкаКлассификаторов(Форма.МенеджерСправочников)  {  ОбработатьСобытиеПроверкиТаблицы = ПроверитьТаблицу,  ОбработатьСобытиеПроверкиСтроки = ПроверитьСтроку,  ОбработатьСобытиеПроверкиЯчейки = ПроверитьЯчейку  };  }  return \_сверкаКлассификаторов;  }  }    private List<СтрокаОтчетаСверкиДанных> ПроверитьТаблицу(АргументыОбработатьСобытиеПроверкиТаблицы arguments)  {  var checkRows = new List<СтрокаОтчетаСверкиДанных>();  var tableCode = arguments.ТаблицаДанных.КодТаблицы;  var tableName = !string.IsNullOrEmpty(arguments.ТаблицаДанных.ТаблицаМетаструктуры.Наименование)  ? arguments.ТаблицаДанных.ТаблицаМетаструктуры.Наименование  : tableCode;    foreach (var row in arguments.ТаблицаДанных.МатрицаЗначений.Строки.Values.Where(row => Check(row)))  {  checkRows.Add(new СтрокаОтчетаСверкиДанных()  {  Форма = Форма.МетаструктураФормы.Код,  Вкладка = tableName,  Субтаблица = tableCode,  Столбец = "КодСтолбца",  Строка = row.КодСтроки,  Условие = "Условие ошибки",  ТипОшибки = "Ошибка заполнения данных",  Значение = row["КодСтолбца"].ЗначениеЧисло,  СохранениеРазрешено = true  });  }    return checkRows;  }    private List<СтрокаОтчетаСверкиДанных> ПроверитьСтроку(АргументыОбработатьСобытиеПроверкиСтроки arguments)  {  var строкиОтчета = new List<СтрокаОтчетаСверкиДанных>();  //...  return строкиОтчета;  }    private СтрокаОтчетаСверкиДанных ПроверитьЯчейку(АргументыОбработатьСобытиеПроверкиЯчейки arguments)  {  var строкаОтчета = new СтрокаОтчетаСверкиДанных();  //...  return строкаОтчета;  } |

В протоколе к платформенным сообщениям об ошибках добавится строка с макросной проверкой (Рисунок 1).

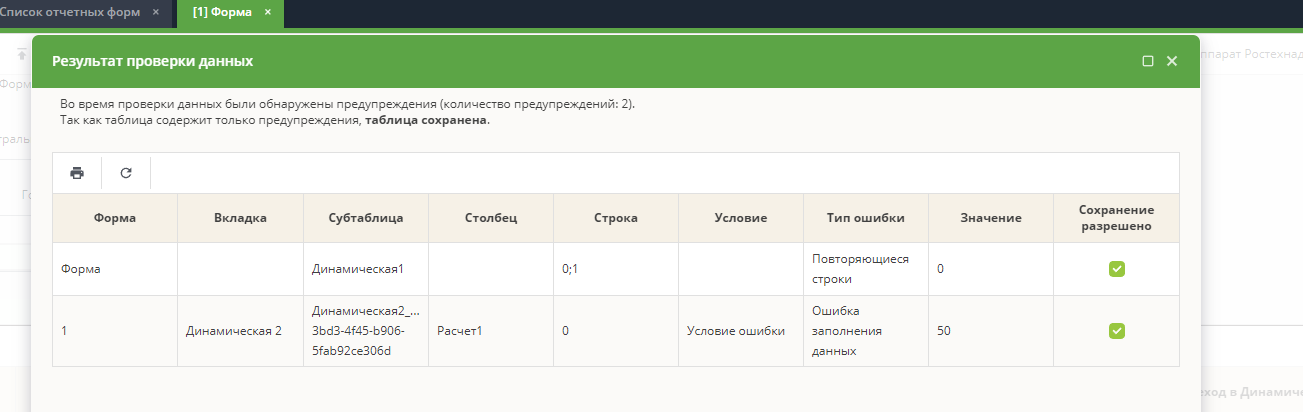


Рисунок 1 – Результат проверки данных

Таблица 3 – Методы и свойства IСверкаКлассификаторов

| Метод или свойство | Тип | Описание |
| --- | --- | --- |
| УчитыватьФильтрИПостОбработку | bool | Если true, то будут учитываться поля ОписаниеСсылкиНаСправочник.Фильтр и ОписаниеСсылкиНаСправочник.ПостОбработка при получении записи справочника в сверке классификаторов, что будет вызывать многочисленные запросы к БД, потому что запись будет запрашиваться с учетом фильтра или постобработки. По умолчанию true. |
| ПроверятьНаПустоту | bool | Производить ли проверку на пустоту. true - да, false - нет. По умолчанию: false |
| УдалятьНеверныеСтроки | bool | Удалять строки не прошедшие проверку. true - да, false - нет. По умолчанию: false |
| ОбработатьСобытиеПроверкиТаблицы | ДелегатОбработатьСобытиеПроверкиТаблицы | Делегат проверки таблицы. Срабатывает для каждой таблицы при сохранении формы и для текущей таблицы при сохранении таблицы. |
| ОбработатьСобытиеПроверкиЯчейки | ДелегатОбработатьСобытиеПроверкиЯчейки | Делегат проверки ячейки формы. Срабатывает для каждой ячейки при сохранении таблицы и формы. |
| ОбработатьСобытиеПроверкиСтроки | ДелегатОбработатьСобытиеПроверкиСтроки | Делегат проверки строки. Срабатывает для каждой строки при сохранении таблицы и формы. |
| Проверить(ДанныеФормы данныеФормы) | List<СтрокаОтчетаСверкиДанных> | Метод проверки классификаторов всех таблиц формы |
| Проверить(ТаблицаДанных таблица) | List<СтрокаОтчетаСверкиДанных> | Метод проверки классификаторов таблицы |
| ПроверитьФорму(ДанныеФормы данныеФормы) | РезультатыВыполненияСверкиДанных | Метод проверки классификаторов всех таблиц формы |

Также реализовано расширение интерфейса сверки классификаторов IformDataRecordsValidationService (Таблица 4).

Таблица 4 – Методы и свойства IFormDataRecordsValidationService

| Метод или свойство | Тип | Описание |
| --- | --- | --- |
| ValidateAsync(ДанныеФормы данныеФормы) | Task<List<СтрокаОтчетаСверкиДанных>> | Асинхронная валидация данных формы |
| ValidateTable(ТаблицаДанных таблица) | List<СтрокаОтчетаСверкиДанных> | Валидация таблицы данных формы |
| ValidateTableAsync(ТаблицаДанных таблица) | Task<List<СтрокаОтчетаСверкиДанных>> | Асинхронная валидация таблицы данных формы |

Если корректно заполнить свойства у строки отчета, то при возникновении ошибки ячейка будет подкрашиваться красным (Рисунок 2). Для этого необходимо указывать коды таблицы, столбца и строки.

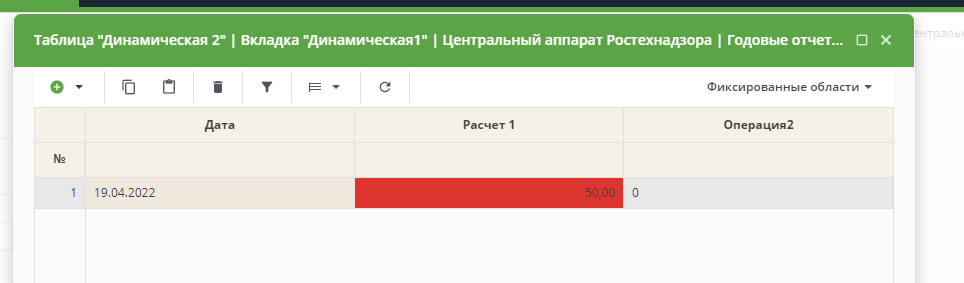


Рисунок 2 – Ошибка в ячейке

Таблица 5 – Свойства класса СтрокаОтчетаСверкиДанных

| Свойство | Тип | Описание |
| --- | --- | --- |
| Форма | string | Возвращает и задает наименование формы, которой соответствует текущая строка сверки. |
| Вкладка | string | Возвращает и задает наименование вкладки формы, которой соответствует текущая строка сверки. |
| Субтаблица | string | Возвращает и задает идентификатор таблицы, которой соответствует текущая строка сверки. |
| Столбец | string | Возвращает и задает код столбца, которому соответствует текущая строка сверки. Если у строки отчета задано "НаименованиеСтолбца", то свойство вернет его значение. |
| НаименованиеСтолбца | string | Возвращает и задает наименование столбца, которому соответствует текущая строка сверки. |
| Строка | string | Возвращает и задает код строки, которой соответствует текущая строка сверки. Если у чтроки отчета задано "НаименованиеСтроки", то свойство вернет его значение. |
| НаименованиеСтроки | string | Возвращает и задает наименование строки, которой соответствует текущая строка сверки. |
| Условие | string | Возвращает и задает строку условия проверки. |
| ТипОшибки | string | Возвращает и задает строку расшифровки ошибки. |
| Значение | decimal | Возвращает и задает значение ячейки или выражения ячеек, не прошедших проверку на корректность введенных данных. |
| СохранениеРазрешено | bool | Возвращает и задает свойство, показывающее, можно ли сохранить данные ячейки, если она не прошла проверку корректности заполнения. |
| RowCode | string | Возвращает код строки, которой соответствует текущая строка сверки. |
| IsDeleted | bool | Признак удаления проверенной строки: true, если проверенная строка была удалена; в противном случае - false. |

# Наложение ограничений на отображение/ редактирование вкладок (таблиц), других элементов формы. Обязательность заполнения ячеек

Для наложения ограничений на отображение вкладок (таблиц) можно использовать один из двух способов:

1. через макрос – СтрокаОграниченияМакросом. Этот метод описан в п. 3 настоящей инструкции;

**Примечание** – Также ограничения можно задать в калькуляторе Условных Атрибутов (см. п. 3).

1. в excel-файле экранной формы.

Для этого в ЭФ выберите нужную вкладку и можете скрыть её либо через пункт меню, либо используя #НеРазмещатьНаЗакладке (Рисунок 3).

Отрабатывает сразу при открытии формы

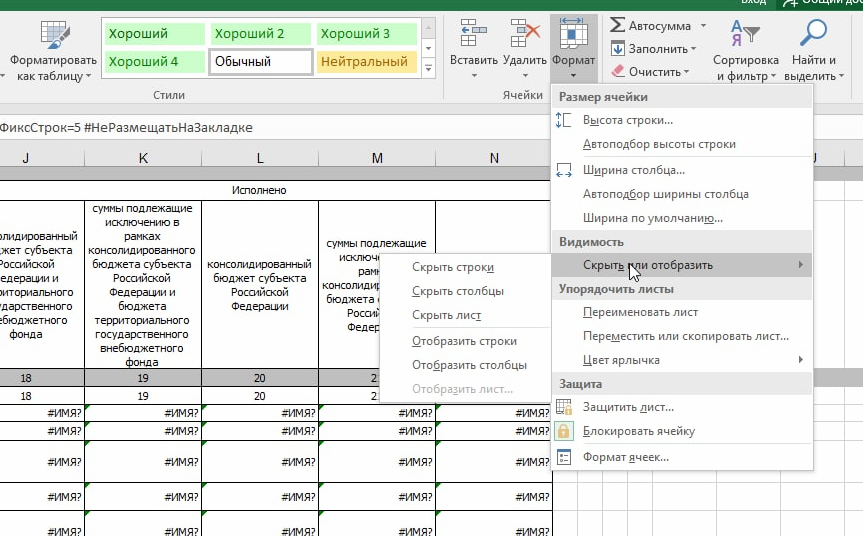


Рисунок 3 – Скрытие вкладки

Для наложения ограничений на редактирование вкладок (таблиц) можно использовать один из двух способов:

1. через «СтрокаОграниченияМакросом»;
2. через Дизайнер.

Во втором случае откройте форму через Дизайнер и в нужной таблице запретите редактирование в ней. Если таблица динамическая, то установите «флажок» в поле «Запрет добавления строк» (Рисунок 4). Отрабатывает сразу при открытии формы.

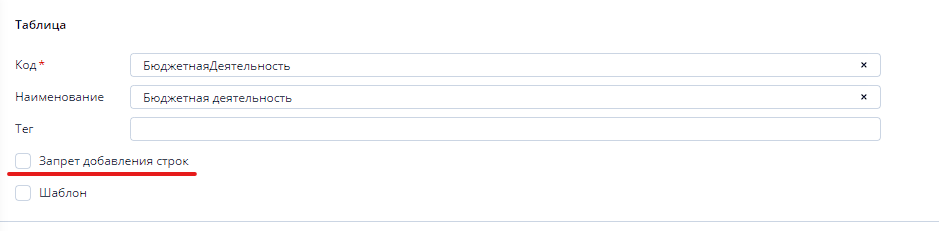


Рисунок 4 – Ограничение на редактирование динамической таблицы

Если таблица статическая, то через столбцы установите «флажок» в поле «Только чтение» (Рисунок 5).

Отрабатывает сразу при открытии формы.

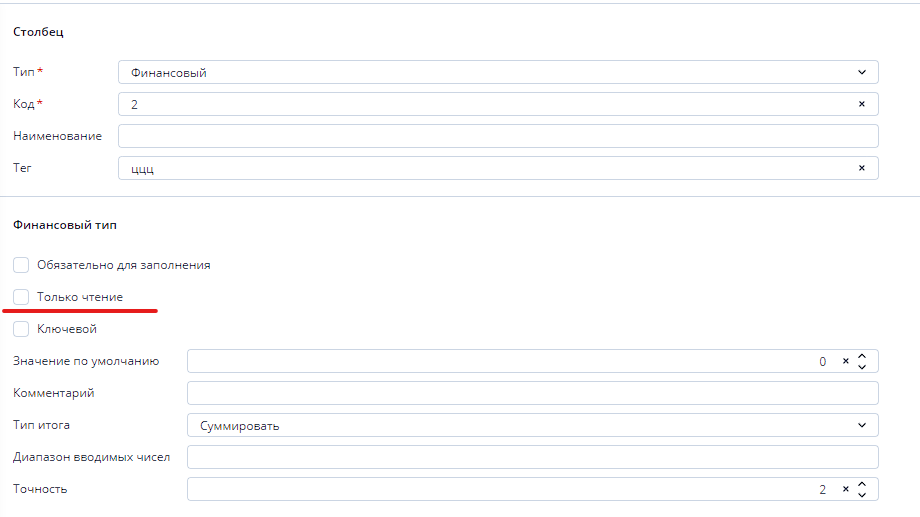


Рисунок 5 – Ограничение на редактирование статической таблицы

Для наложения ограничений на отображение / редактирование ячеек формы:

1. ячейки можно заблокировать на редактирование или отображение через «СтрокаОграниченияМакросом»;
2. ячейки можно заблокировать на редактирование через Дизайнер – в свободных ячейках установите «флажок» в пункте «Только чтение»   
   (Рисунок 5).

Для того чтобы наложить ограничение на отображение, скройте через экранную форму строку/столбец (Рисунок 6). Отрабатывает сразу при открытии формы.

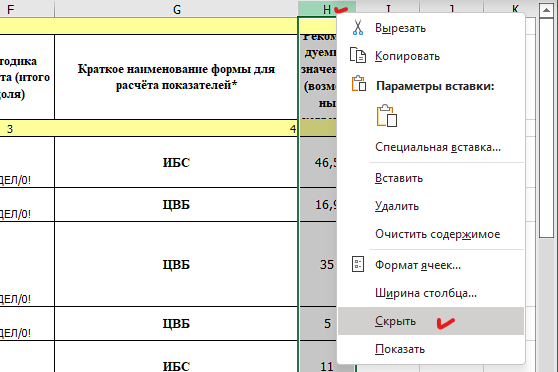


Рисунок 6 – Скрытие строки/ столбца

1. ячейки можно заблокировать на редактирование в самой форме, прописав в макросе логику, где у значения ячейки в свойстве «ТолькоЧтение» проставьте значение «true». Свойство «НеВидимая» не скроет ячейку, несмотря на то, что имеет публичный сеттер (Рисунок 7).

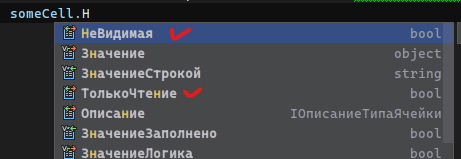


Рисунок 7 – Макрос

Эффект от изменения значения свойства «ТолькоЧтение» будет только при вызове методов при открытии формы. Это обработчики «ОбработатьСобытиеОткрытияФормы», «ОбработатьСобытиеДоПересчетаАвтоблоков», «ОбработатьСобытиеПослеПересчетаАвтоблоков».

**Примечание** – Обработчики «ОбработатьСобытиеДоПересчетаАвтоблоков», «ОбработатьСобытиеПослеПересчетаАвтоблоков» будут иметь эффект только на первом вызове при открытии формы. Например, если обернуть установку значения для «ТолькоЧтение» в условие, которое будет выполняться только после открытия формы, и потом вызывать пересчет – эффекта от изменения значения свойства не будет.

Если заполнять свойство «ТолькоЧтение» у класса «ТипЯчейки», то правки будут внесены только для текущей ячейки, и не отразятся на метаструктуре.

Если заполнять свойство «ТолькоЧтение» через «ОписаниеТипаЯчейки», то правки будут внесены для всего столбца, и отразятся на метаструктуре.

|  |
| --- |
| public override void ОбработатьСобытиеОткрытияФормы(АргументыОткрытияФормы аргументы)  {  var строка = Форма["Статическая"].МатрицаЗначений["строка2"];  if (Форма.Идентификатор.Учреждение.Код == "Центральный аппарат Ростехнадзора")  {  //блокирует только текущую ячейку  строка["столбец1"].ТолькоЧтение = true;  }  else  {  //изменяет метаструктуру и блокирует весь столбец таблицы  строка.Ячейки["столбец2"].Описание.ТолькоЧтение = true;  }  } |

## Обязательность заполнения ячеек

Ячейки в форме можно сделать обязательными для заполнения. Это осуществляется в Дизайнере отчетных форм.

В метаструктуре в свободных ячейках установите «флажок» в поле «Обязательно для заполнения» (Рисунок 8).

Если нужны ячейки таблиц, то через столбцы таблицы установите «флажок» в поле «Обязательно для заполнения» (Рисунок 9).

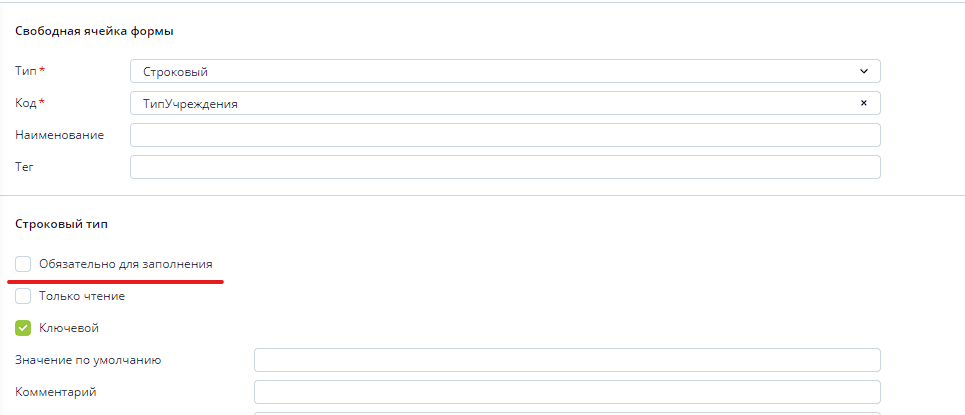


Рисунок 8 – Поле «Обязательно для заполнения» в свободных ячейках

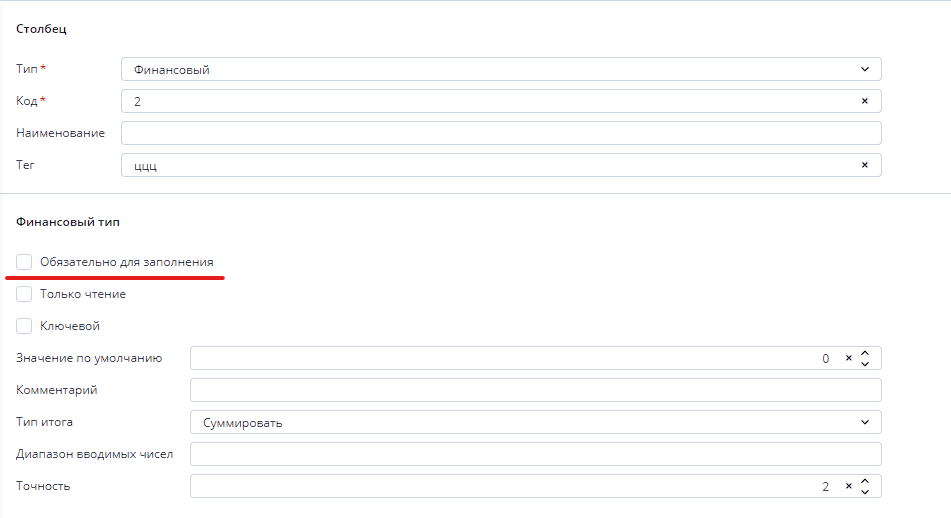


Рисунок 9 – Поле «Обязательно для заполнения»

# Работа с пользовательскими параметрами, зависимость пользовательских параметров

## Новые параметры

### Подготовка прикладного кода

Новый функционал пользовательских параметров находится в сборках BARS.Svody.UserParameters и BARS.Svody.UserParameters.Win.

Для использования описанного в данном разделе функционала необходимо добавить ссылки на пакеты платформы BARS.Svody.UserParameters и BARS.Svody.UserParameters.Win.

### Fluent Builder пользовательских параметров

Для удобства конфигурирования пользовательских параметров реализован builder с fluent-интерфейсом.

В данной статье описаны его возможности и примеры использования.

#### Создание параметров с помощью билдера

Классу СписокПользовательскихПараметров добавлен метод расширения BuildParameter<TParamType>:

|  |
| --- |
| public static void BuildParameter<TParamType>(this СписокПользовательскихПараметров parametersList,  Action<UserParameterBuilder<TParamType>> action) |

**TParamType** – тип пользовательского параметра. В случае новых пользовательских параметров – интерфейс (список интерфейсов параметров и их описание см.в п. 7.1.11 настоящей инструкции).

Данный метод расширения принимает параметром делегат, который должен сконфигурировать параметр с помощью «UserParameterBuilder», который будет передан в делегат.

Таблица 6 – Список методов UserParameterBuilder

| Сигнатура | Описание метода | Примечание |
| --- | --- | --- |
| UserParameterBuilder<TParamType> WithId(string id) | Задать идентификатор. | Старый эквивалент:  userParameter.ИдентификаторПараметра = id; |
| UserParameterBuilder<TParamType> WithTitle(string title) | Задать заголовок. | Старый эквивалент:  userParameter.Заголовок = title; |
| UserParameterBuilder<TParamType> WithDefaultValue(object value) | Задать значение по умолчанию. | Старый эквивалент:  userParameter.ЗначениеПоУмолчанию = value; |
| UserParameterBuilder<TParamType> ApplyConfig(object config) | Применить произвольную конфигурацию | См.п.7.1.2.2 |
| BaseDependentUserParameterBuilder<TParamType, TDependent>  ToDependent<TDependent>(string param1Identifier)  where TDependent : IDependentParameter1 | Сделать параметр зависимым от одного параметра | TDepended – класс, реализующий функциональность зависимости, подробное описание тут  См. п. 7.1.13  Обратить внимание, что возвращаемый тип - BaseDependentUserParameterBuilder |
| BaseDependentUserParameterBuilder<TParamType, TDependent>  ToDependent<TDependent>(string param1Identifier, string param2Identifier)  where TDependent : IDependentParameter2 | Сделать параметр зависимым от двух параметров |
| BaseDependentUserParameterBuilder<TParamType, TDependent>  ToDependent<TDependent>(string param1Identifier, string param2Identifier,  string param3Identifier)  where TDependent : IDependentParameter3 | Сделать параметр зависимым от трёх параметров |
| BaseDependentUserParameterBuilder<TParamType, TDependent>  ToDependent<TDependent>(string param1Identifier, string param2Identifier,  string param3Identifier, string param4Identifier)  where TDependent : IDependentParameter4 | Сделать параметр зависимым от четырёх параметров |

Таблица 7 – Методы расширения класса UserParameterBuilder

| Сигнатура | Описание метода | Примечание |
| --- | --- | --- |
| static UserParameterBuilder<TParamType> Multiselect<TParamType>(  this UserParameterBuilder<TParamType> builder)  where TParamType : IMultiselectableParameter | Задать множественный выбор. | Может быть использован только для параметров реализующих интерфейс  IMultiselectableParameter |
| static UserParameterBuilder<TParamType> WithItems<TParamType, TItemType>(  this UserParameterBuilder<TParamType> builder, IEnumerable<TItemType> items)  where TParamType : IParameterWithItems<TItemType> | Задать список значений для выбора. | Может быть использован только для параметров реализующих интерфейс  IParameterWithItems<TItemType>  TItemType - тип элемента списка значений параметра |
| static UserParameterBuilder<TParamType> WithColumns<TParamType>(  this UserParameterBuilder<TParamType> builder, IEnumerable<Column> columns)  where TParamType : IParameterWithColumns | Задать список столбцов. | Может быть использован только для параметров реализующих интерфейс  IParameterWithColumns  См.п. 7.1.2.3 |

Таблица 8 – Список методов BaseDependentUserParameterBuilder

| Сигнатура | Описание | Примечание |
| --- | --- | --- |
| BaseDependentUserParameterBuilder<TParamType, TDependent> WithId(string id) | Задать идентификатор. | Старый эквивалент:  userParameter.ИдентификаторПараметра = id; |
| BaseDependentUserParameterBuilder<TParamType, TDependent> WithTitle(string title) | Задать заголовок. | Старый эквивалент:  userParameter.Заголовок = title; |
| BaseDependentUserParameterBuilder<TParamType, TDependent> WithDefaultValue(object value) | Задать значение по умолчанию. | Старый эквивалент:  userParameter.ЗначениеПоУмолчанию = value; |
| BaseDependentUserParameterBuilder<TParamType, TDependent> ApplyConfig(object config) | Применить произвольную конфигурацию | См. п. 7.1.2.2 |

Таблица 9 – Методы расширения класса BaseDependentUserParameterBuilder

| Сигнатура | Описание метода | Примечание |
| --- | --- | --- |
| static BaseDependentUserParameterBuilder<TParamType, TDependent> Multiselect<TParamType, TDependent>(  this BaseDependentUserParameterBuilder<TParamType, TDependent> builder)  where TParamType : IMultiselectableParameter   where TDependent : IDependentParameter | Задать множественный выбор. | Может быть использован только для параметров, реализующих интерфейс  IMultiselectableParameter  TDepended - класс реализующий функциональность зависимости, подробное описание см. в п.7.1.13 |
| static BaseDependentUserParameterBuilder<TParamType, TDependent> WithItems<TParamType, TDependent, TItemType>(  this BaseDependentUserParameterBuilder<TParamType, TDependent> builder, IEnumerable<TItemType> items)  where TParamType : IParameterWithItems<TItemType>  where TDependent : IDependentParameter | Задать список значений для выбора. | Может быть использован только для параметров реализующих интерфейс  IParameterWithItems<TItemType>  TItemType - тип элемента списка значений параметра  TDepended - класс реализующий функциональность зависимости, подробное описание см. в п.7.1.13 |
| static BaseDependentUserParameterBuilder<TParamType, TDependent> WithColumns<TParamType, TDependent>(  this BaseDependentUserParameterBuilder<TParamType, TDependent> builder, IEnumerable<Column> columns)  where TParamType : IParameterWithColumns   where TDependent : IDependentParameter | Задать список столбцов. | Может быть использован только для параметров реализующих интерфейс  **IParameterWithColumns**  **TDepended** - *класс реализующий функциональность зависимости,*  подробное описание см. в п.7.1.13, 7.1.2.3 |

##### Примеры использования

|  |
| --- |
| var ec = Форма.Идентификатор.КомпонентОтчетногоПериода.IЦепочкаСдачи.Элементы;    параметры.BuildParameter<IChainElementSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборЭлементаЦепочки2")  .WithTitle("Выбор элемента(ов) цепочки сдачи отчетности2")  .WithItems(ec)  .WithDefaultValue(ec.Take(2));  }); |

|  |
| --- |
| var columns = new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  },  new Column  {  ModelDataIndex = "ГруппаЭквивалентности",  StoredDataIndex = "ГруппаЭквивалентности",  Header = "ГруппаЭквивалентности",  Width = 200  }  };    параметры.BuildParameter<IReportPeriodComponentSelector>(parameter =>  {  parameter.WithId("ПараметрВыборКомпонентовОП2")  .WithTitle("Выбор компонента(ов) отчётного периода2")  .WithItems(((ОтчетныйПериод)op).Компоненты)  .WithDefaultValue(((ОтчетныйПериод)op).Компоненты.Take(1))  .WithColumns(columns);  }); |

|  |
| --- |
| var records = new List<ParamsModel>();  var rec1 = new ParamsModel { Code = "1", Guid = Guid.NewGuid().ToString(), Text = "text1"};  records.Add(rec1);    for (var i = 2; i < 1000; i++)  {  records.Add(new ParamsModel { Code = i.ToString(), Guid = Guid.NewGuid().ToString(), Text = "text" + i});  }    var lastRecord = new ParamsModel { Code = "last", Guid = Guid.NewGuid().ToString(), Text = "textlast" };  records.Add(lastRecord);    параметры.BuildParameter<ITableSelector<ParamsModel>>(  parameter =>  {  parameter.WithId("SingleChoiceMode")  .WithTitle("Одиночный выбор")  .WithItems(records)  .WithDefaultValue( new List<ParamsModel> { records[0] })  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "Code",  StoredDataIndex = "Code",  Header = "Code",  Width = 200  }  });  }); |

|  |
| --- |
| параметры.BuildParameter<IRealDictionarySelector>(  parameter =>  {  parameter.WithId("id1")  .WithTitle("RealDictionarySelectorConfig")  .Multiselect()  .ApplyConfig(  new  {  DictionaryCode = "БюджетнаяОтчетность.КодыДоходов",  DictionaryActualDate = DateTime.Now  });  }); |

#### ApplyConfig

ApplyConfig используется для применения конфигурации к пользовательскому параметру, когда значение свойства параметра нельзя задать с помощью других методов билдера.

Ниже представлена таблица свойств, которые можно заполнить с помощью метода ApplyConfig (Таблица 10).

Таблица 10 – Таблица свойств

| Тип параметра | Свойство |
| --- | --- |
| Выбор из физического справочника, п.7.1.3.1 | DictionaryCode |
| DictionaryActualDate |
| Attributes |
| Выбор из виртуального справочника, п. 7.1.3.2 | Records |
| Attributes |
| Выбор из справочника с ленивой загрузкой, п. 7.1.3.3 | RootNodes |
| Attributes |
| Пользовательский параметр выбора компонента отчётного периода, п. 7.1.5 | ReportPeriodCode |
| Пользовательский параметр выбора элемента цепочки, п. 7.1.8 | ChainCode |
| Пользовательский параметр выбора элементов цепочки из компонентов, п. 7.1.9 | ReportPeriodCode |
| ElementsMultiselect |

В качестве аргумента метода можно передать:

* строго типизированный DTO, например:

|  |
| --- |
| public class DictionaryConfig  {  public string DictionaryCode { get; set; }    public string DictionaryActualDate { get; set; }    public IEnumerable<БазовыйАтрибутСправочника> Attributes { get; set; }  }    var config = new DictionaryConfig  {  DictionaryCode = "БюджетнаяОтчетность.КодыДоходов",  DictionaryActualDate = new DateTime(2015, 1, 1),  Attributes = dictionary.АтрибутыСправочника.Take(1).Select(x => new БазовыйАтрибутСправочника(x))  };    .ApplyConfig(config); |

* анонимный объект:

|  |
| --- |
| .ApplyConfig(new  {  DictionaryCode = "БюджетнаяОтчетность.КодыДоходов",  DictionaryActualDate = new DateTime(2015, 1, 1),  Attributes = dictionary.АтрибутыСправочника.Take(1).Select(x => new БазовыйАтрибутСправочника(x))  }; |

Также стоит обратить внимание, что при реализации зависимых параметров (см. п. 7.1.13) может появиться необходимость в использовании дополнительных свойств в классе, отвечающем за функциональность зависимости.

Для этого можно использовать следующий прием:

|  |
| --- |
| public class ComponentsSelector : DependentParam<List<ReportPeriodValueDto>, IКомпонентОтчетногоПериода>  {  // Кастомное свойство в классе параметра  public ДанныеФормы Form { get; set; }  public override IEnumerable<IКомпонентОтчетногоПериода> GetDataSourceInner(List<ReportPeriodValueDto> dependValue1)  {  if (dependValue1 == null || !dependValue1.Any())  {  return new List<IКомпонентОтчетногоПериода>();  }    var opDto = dependValue1.First();    var op = Выборка<ОтчетныйПериод>.НайтиОбъектПоРеквизиту("Идентификатор", opDto.Id);    var components = new Выборка<КомпонентОтчетногоПериода>();  components.Запрос.ДобавитьУсловиеОтбора("ОтчетныйПериод", op);  components.ЗагрузитьПолностью();    return components;  }  }    .ToDependent<ComponentsSelector>()  .ApplyConfig(new  {  Form = Форма  }; |

Таким образом, в свойство Form класса ComponentsSelector будет установлено значение, соответствующее текущей форме.

#### WithColumns

У параметров, реализующих интерфейс «IparameterWithColumns», есть возможность задать список отображаемых столбцов.

Пример использования:

|  |
| --- |
| параметры.BuildParameter<ITableSelector<ParamsModel>>(  parameter =>  {  parameter.WithId("MultiChoiceMode1st")  .WithTitle("Множественный выбор")  .WithItems(records)  .WithDefaultValue( new List<ParamsModel> { records[0], records[1], records[2] })  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "Code",  StoredDataIndex = "Code",  Header = "Code",  Width = 200  },  new Column  {  ModelDataIndex = "Guid",  StoredDataIndex = "Guid",  Header = "Guid",  Width = 200  }  })  .Multiselect();  });    параметры.BuildParameter<IReportPeriodComponentSelector>(  parameter =>  {  parameter  .WithId("ПараметрВыборКомпонентовОП2")  .WithTitle("Выбор компонента(ов) отчётного периода2")  .WithItems(((ОтчетныйПериод)op).Компоненты)  .WithDefaultValue(((ОтчетныйПериод)op).Компоненты.Take(1))  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  },  new Column  {  ModelDataIndex = "ГруппаЭквивалентности",  StoredDataIndex = "ГруппаЭквивалентности",  Header = "ГруппаЭквивалентности",  Width = 200  }  });  }); |

Описание свойств модели «Column» (Таблица 11).

Таблица 11 – Описание свойств модели «Column»

| Название свойства | Описание | Примечание |
| --- | --- | --- |
| Header | Заголовок столбца |  |
| Width | Ширина столбца | Используется только в WEB |
| StoredDataIndex | Наименование свойства сущности для маппинга значений |  |
| ModelDataIndex | Наименование свойства модели используемой на клиенте для маппинга значений | Используется только в WEB |
| Hidden | Столбец скрыт |  |

В случае web-клиента существует дополнительная возможность задать тип столбца, для этого необходимо использовать наследника «Column» - «ExtendedColumn» (Таблица 12).

Таблица 12 – Описание свойств модели «ExtendedColumn»

| Название свойства | Описание | Примечание |
| --- | --- | --- |
| ColumnType | Тип столбца | Используется только в WEB.  Значением является xtype столбца grid/treegrid в Ext.js |

### Пользовательский параметр выбора из справочника

#### Выбор из физического справочника

Данную конфигурацию параметра необходимо использовать в случае, если пользователю необходимо дать выбор записей из физического справочника, существующего в Системе (см. п. 7.1.2).

Пример использования:

|  |
| --- |
| параметры.BuildParameter<IRealDictionarySelector>(  parameter =>  {  parameter.WithId("id1")  .WithTitle("RealDictionarySelectorConfig")  .Multiselect()  .ApplyConfig(  new  {  DictionaryCode = "БюджетнаяОтчетность.КодыДоходов",  DictionaryActualDate = DateTime.Now  });  }); |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* DictionaryCode;
* DictionaryActualDate;
* Attributes.

Также в класс «СписокПользовательскихПараметров» добавлен новый generic-метод GetParameterValueAs<TParamValueType>.

Возвращаемый тип параметра всегда List<SelectedDictionaryRecordDto>, даже если конфигурация параметра с одиночным выбором

Пример получения значения параметра:

|  |
| --- |
| var selectedRecords = СписокПараметровОперации.GetParameterValueAs<List<SelectedDictionaryRecordDto>>("id1");    foreach (var recordDto in selectedRecords)  {  var code = recordDto.Code;  var id = recordDto.Id;  var fullCode = recordDto.FullCode;  var name = recordDto.Name;  } |

#### Выбор из виртуального справочника

Данную конфигурацию параметра необходимо использовать в случае, если пользователю необходимо дать выбор записей из виртуального списка записей сформированного в памяти приложения (см. п. 7.1.2 настоящей инструкции).

Пример использования:

|  |
| --- |
| var records = new List<IЗаписьСправочника>();  var parentRecord = new JoinedDynamicDictionaryRecord(null, "parentRecord1")  {  Код = "parentRecord",  Наименование = "parentRecord"  };  records.Add(parentRecord);  for (int i = 1; i <= 10000; i++)  {  records.Add(new JoinedDynamicDictionaryRecord(null, i.ToString())  {  Код = i.ToString(),  Наименование = i.ToString(),  РодительскаяЗапись = parentRecord  });  }    var identifier = new ИдентификаторУниверсальногоСправочника  {  ПолноеНаименованиеСправочника = "БюджетнаяОтчетность.КодыДоходов",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  };  var dictionary = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(identifier);    параметры.BuildParameter<IVirtualDictionarySelector>(  parameter =>  {  parameter.WithId("id2")  .WithTitle("VirtualDictionarySelectorConfig")  .ApplyConfig(  new  {  Records = records,  Attributes = dictionary.АтрибутыСправочника.Take(1).Select(x => new БазовыйАтрибутСправочника(x))  });  }); |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* Records;
* Attributes.

Также в класс «СписокПользовательскихПараметров» добавлен новый generic-метод GetParameterValueAs<TParamValueType>.

Возвращаемый тип параметра всегда List<SelectedDictionaryRecordDto>, даже если конфигурация параметра с одиночным выбором.

Пример получения значения параметра:

|  |
| --- |
| var selectedRecords = СписокПараметровОперации.GetParameterValueAs<List<SelectedDictionaryRecordDto>>("id2");    foreach (var recordDto in selectedRecords)  {  var code = recordDto.Code;  var id = recordDto.Id;  var fullCode = recordDto.FullCode;  var name = recordDto.Name;  } |

#### Выбор из справочника с ленивой загрузкой записей

Данную конфигурацию параметра необходимо использовать, если пользователю необходимо дать выбор из физического справочника с кастомной фильтрацией записей, либо из виртуального справочника (см. п. 7.1.2 настоящей инструкции).

Пример использования:

|  |
| --- |
| var identifier = new ИдентификаторУниверсальногоСправочника  {  ПолноеНаименованиеСправочника = "БюджетнаяОтчетность.КодыДоходов",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  };  var dictionary = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(identifier);    var records = dictionary.Записи;    параметры.BuildParameter<ILazyDictionarySelector>(  parameter =>  {  parameter.WithId("id3")  .WithTitle("LazyDictionarySelectorConfig")  .ApplyConfig(new  {  RootNodes = records.Where(x => x.IРодительскаяЗапись == null)  .Select(  x => new DictionaryRecordNode() { Record = x, IsLeaf = !x.ЕстьПодчиненныеЗаписи })  .ToList(),  Attributes = dictionary  .АтрибутыСправочника.Take(1).Select(x => new БазовыйАтрибутСправочника(x))  });  }); |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* RootNodes
* Attributes

Для реализации ленивой подгрузки записей в сборке, из которой добавляется параметр, должен быть реализован класс-провайдер:

Обязательные требования к классу-провайдеру (Таблица 13):

Таблица 13 – Требования к классу-провайдеру

| Требование | Примечание |
| --- | --- |
| Класс должен быть публичным |  |
| Класс должен реализовывать интерфейс ILazyDictionarySelectorDataProvider | интерфейс пустой, используется как маркер для поиска |
| Класс должен содержать публичный метод с сигнатурой public IEnumerable<DictionaryRecordNode> MethodName(Guid parameterName) | * является провайдером записей при подгрузке, на вход принимает идентификатор записи, узел которой разворачивается в этот момент * должен вернуть список DTO дочерних узлов * должен быть помечен атрибутом LazyDictionaryParameterProvider с указанием идентификатора параметра, для которого данный метод является провайдером записей |
| Класс должен содержать публичный метод с сигнатурой public LinkedList<DictionaryRecordNode> MethodName(SearchParameters parameterName) | * метод является провайдером поиска записей в справочнике, на вход принимает DTO параметров поиска со свойствами Field (поле, по которому происходит поиск, например «Код» или «Наименование») и Value (введенный пользователем текст для поиска) * метод должен вернуть двусвязный список DTO найденных узлов * метод должен быть помечен атрибутом  LazyDictionaryParameterSearchProvider с указанием идентификатора параметра, для которого данный метод является провайдером поиска |

Пример реализации:

|  |
| --- |
| public class Provider : ILazyDictionarySelectorDataProvider  {  [LazyDictionaryParameterProvider("id3")]  public IEnumerable<DictionaryRecordNode> GetRecordByGuid(Guid parentGuid)  {  var identifier = new ИдентификаторУниверсальногоСправочника  {  ПолноеНаименованиеСправочника = "БюджетнаяОтчетность.КодыДоходов",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  };  var dictionary = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(identifier);    var idString = Барс.Клиент.Приложение.РежимWebПриложения  ? parentGuid.ToOracleId()  : Барс.Типы.Конвертер.ВСтроку(parentGuid);    var records = dictionary.ПолучитьСписокЗаписей(string.Format("[РодительскаяЗапись] = '{0}'", idString));    return records.Select(x => new DictionaryRecordNode() { Record = x, IsLeaf = !x.ЕстьПодчиненныеЗаписи });  }    [LazyDictionaryParameterSearchProvider("id3")]  public LinkedList<DictionaryRecordNode> MethodForSearch(SearchParameters parameters)  {  var identifier = new ИдентификаторУниверсальногоСправочника  {  ПолноеНаименованиеСправочника = "БюджетнаяОтчетность.КодыДоходов",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  };  var dictionary = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(identifier);    if (parameters.Field.Equals("Код"))  {  return new LinkedList<DictionaryRecordNode>(dictionary.ПолучитьСписокЗаписей(string.Format("[Код] LIKE '%{0}%'", parameters.Value))  .Select(x => new DictionaryRecordNode() { Record = x, IsLeaf = !x.ЕстьПодчиненныеЗаписи }));  }  if (parameters.Field.Equals("Наименование"))  {  return new LinkedList<DictionaryRecordNode>(dictionary.ПолучитьСписокЗаписей(string.Format("[Наименование] LIKE '%{0}%'", parameters.Value))  .Select(x => new DictionaryRecordNode() { Record = x, IsLeaf = !x.ЕстьПодчиненныеЗаписи }));  }  return new LinkedList<DictionaryRecordNode>();  }  } |

Также в класс «СписокПользовательскихПараметров» добавлен новый generic-метод GetParameterValueAs<TParamValueType>.

Возвращаемый тип параметра всегда List<SelectedDictionaryRecordDto>, даже если конфигурация параметра с одиночным выбором

Пример получения значения параметра:

|  |
| --- |
| var selectedRecords = СписокПараметровОперации.GetParameterValueAs<List<SelectedDictionaryRecordDto>>("id1");    foreach (var recordDto in selectedRecords)  {  var code = recordDto.Code;  var id = recordDto.Id;  var fullCode = recordDto.FullCode;  var name = recordDto.Name;  } |

### Пользовательский параметр выбора из таблицы

Реализован пользовательский параметр выбора из таблицы ITableSelector<TRecordType>, где TrecordType – тип-наследник класса **BaseTableRecord**.

Описание базового класса записи таблицы BaseTableRecord:

| Свойство | Описание |
| --- | --- |
| public int ServerId { get; set; } | Служебное свойство. Используется для поиска исходных записей после передачи с сервера на клиент и наоборот. **Не использовать, значение проставляется автоматически платформой** |
| public string Text { get; set; } | Значение данного свойства отображается в редакторе при одиночном выборе. |

#### Описание реализации

Для добавления нового пользовательского параметра в список параметров создайте DTO-класс – наследник класса Приложение.ОтчетнаяФорма.UserParameters.TableSelector.BaseTableRecord.

Добавьте ему необходимые свойства.

**Примечание** – Необходимо иметь в виду, что в версии 5.1 Системы реализована поддержка только строковых значений в свойствах записи таблицы.

Сформируйте список записей для таблицы.

Сформируйте список наименований столбцов таблицы. Наименованию отображаемого столбца должно соответствовать свойство DTO-класса записи.

Добавьте параметр в список параметров через Fluent Builder, указав вместо TRecordType нужный тип.

Для корректной работы свойства SelectedItems необходимо учитывать, что объект должен быть эквивалентен одному объекту из списка Items.

#### Пример использования

|  |
| --- |
| /// <summary> Модель строки таблицы </summary>  public class ParamsModel : BaseTableRecord  {  public string Code { get; set; }    public string Guid { get; set; }  }    protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string operationName)  {  var params = new СписокПользовательскихПараметров();    if(operationName == "Test")  {  var records = new List<ParamsModel>();  var rec1 = new ParamsModel { Code = "1", Guid = Guid.NewGuid().ToString(), Text = "text1"};  records.Add(rec1);    for (var i = 2; i < 1000; i++)  {  records.Add(new ParamsModel { Code = i.ToString(), Guid = Guid.NewGuid().ToString(), Text = "text" + i});  }    var lastRecord = new ParamsModel { Code = "last", Guid = Guid.NewGuid().ToString(), Text = "textlast" };  records.Add(lastRecord);    params.BuildParameter<ITableSelector<ParamsModel>>(  parameter =>  {  parameter.WithId("SingleChoiceMode") // идентификатор параметра  .WithTitle("Одиночный выбор") // заголовок отображаемый рядом с редактором на форме ввода пользовательских параметров  .WithItems(records) // записи таблицы  .WithDefaultValue( new List<ParamsModel> { records[0] }) // выбранная по умолчанию запись  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "Code",  StoredDataIndex = "Code",  Header = "Code",  Width = 200  }  }); // список отображаемых столбцов таблицы  });    params.BuildParameter<ITableSelector<ParamsModel>>(  parameter =>  {  parameter.WithId("MultiChoiceMode1st") // идентификатор параметра  .WithTitle("Множественный выбор") // заголовок отображаемый рядом с редактором на форме ввода пользовательских параметров  .WithItems(records) // записи таблицы  .WithDefaultValue( new List<ParamsModel> { records[0], records[1], records[2] }) // выбранные по умолчанию записи  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "Code",  StoredDataIndex = "Code",  Header = "Code",  Width = 200  },  new Column  {  ModelDataIndex = "Guid",  StoredDataIndex = "Guid",  Header = "Guid",  Width = 200  }  }) // список отображаемых столбцов таблицы  .Multiselect(); // с множественным выбором  });  params.BuildParameter<ITableSelector<ParamsModel>>(  parameter =>  {  parameter.WithId("MultiChoiceMode2nd") // идентификатор параметра  .WithTitle("Ещё один множественный выбор") // заголовок отображаемый рядом с редактором на форме ввода пользовательских параметров  .WithItems(records) // записи таблицы  .WithColumns(new List<Column>  {  new Column  {  ModelDataIndex = "Code",  StoredDataIndex = "Code",  Header = "Code",  Width = 200  }  }) // список отображаемых столбцов таблицы  .Multiselect(); // с множественным выбором  });  }  return params;  }    [АтрибутПунктаОбработки("Test")]  public void Test()  {  var result = new List<string>();    var single = СписокПараметровОперации.GetParameterValueAs<List<ParamsModel>>("SingleChoiceMode"); // получаем значение параметра с одиночным выбором  result.Add(string.Format("Код: {0}", single.FirstOrDefault().Code));    var multi1 = СписокПараметровОперации.GetParameterValueAs<List<ParamsModel>>("MultiChoiceMode1st"); // получаем значение параметра с множественным выбором    foreach (var record in multi1)  {  result.Add(string.Format("Код: {0}", record.Code));  }    var multi2 = СписокПараметровОперации.GetParameterValueAs<List<ParamsModel>>("MultiChoiceMode2nd"); // получаем значение параметра с множественным выбором    foreach (var record in multi2)  {  result.Add(string.Format("Код: {0}", record.Code));  }    // ...  } |

### Пользовательский параметр выбора компонента отчётного периода

Реализован пользовательский параметр выбора компонента отчётного периода IreportPeriodComponentSelector.

Возвращаемый тип параметра всегда List<ReportPeriodComponentValueDto>, даже если конфигурация параметра с одиночным выбором.

Пример использования параметра «ReportPeriodComponentSelector»:

|  |
| --- |
| namespace Барс.Своды  {  public partial class МодульФормы\_0503161 : БазовыйОбработчикФормы  {  protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string Операция)  {  var параметры = new СписокПользовательскихПараметров();  if (Операция == "ОперацияПроверкиВнутреннихУвязок")  {  var op = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод;    параметры.BuildParameter<IReportPeriodComponentSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборКомпонентовОП1")  .WithTitle("Выбор компонента(ов) отчётного периода1")  .ApplyConfig(new  {  ReportPeriodCode = op.Код  });  });    var columns = new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  },  new Column  {  ModelDataIndex = "ГруппаЭквивалентности",  StoredDataIndex = "ГруппаЭквивалентности",  Header = "ГруппаЭквивалентности",  Width = 200  }  };    параметры.BuildParameter<IReportPeriodComponentSelector>(parameter =>  {  parameter  .WithId("ПараметрВыборКомпонентовОП2")  .WithTitle("Выбор компонента(ов) отчётного периода2")  .WithItems(((ОтчетныйПериод)op).Компоненты)  .WithDefaultValue(((ОтчетныйПериод)op).Компоненты.Take(1))  .WithColumns(columns);  });    параметры.BuildParameter<IReportPeriodComponentSelector>(parameter =>  {  parameter  .WithId("ПараметрВыборКомпонентовОП3")  .WithTitle("Выбор компонента(ов) отчётного периода3")  .Multiselect()  .ApplyConfig(new  {  ReportPeriodCode = op.Код  });  });    параметры.BuildParameter<IReportPeriodComponentSelector>(parameter =>  {  parameter  .WithId("ПараметрВыборКомпонентовОП4")  .WithTitle("Выбор компонента(ов) отчётного периода4")  .Multiselect() // с множественным выбором  .WithItems(((ОтчетныйПериод)op).Компоненты)  .WithDefaultValue(((ОтчетныйПериод)op).Компоненты.Take(2))  .WithColumns(columns);  });  }  //...  return параметры;  }  }  } |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* ReportPeriodCode.

**Примечание** – Если список столбцов не задаётся, то в таком случае свойству «Columns» пользовательского параметра «ReportPeriodComponentSelector» присваивается значение по умолчанию:

|  |
| --- |
| /// <summary> Построить список столбцов по умолчанию. </summary>  /// <returns> Список моделей столбцов. </returns>  public IEnumerable<Column> BuildColumns()  {  var result = new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  }  };  return result;  } |

### Пользовательский параметр выбора отчётного периода

Реализован пользовательский параметр выбора отчётного периода – «ReportPeriodSelector».

Возвращаемый тип параметра всегда List<ReportPeriodValueDto>, даже если конфигурация параметра с одиночным выбором.

Пример использования параметра «ReportPeriodSelectorParameter»:

|  |
| --- |
| namespace Барс.Своды  {  public partial class МодульФормы\_0503161 : БазовыйОбработчикФормы  {  protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string Операция)  {  var параметры = new СписокПользовательскихПараметров();  if(Операция == "TestReportPeriod")  {  параметры.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect1")  .WithTitle("Выбор периодов одиночный");  });    параметры.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect2")  .WithTitle("Выбор периодов множественный")  .Multiselect();  });    var periods = new Выборка<ОтчетныйПериод>();  periods.Запрос.ДобавитьУсловиеОтбора("ДатаНачала", new DateTime(2015, 1, 1), ОперацияПараметраКоманды.БольшеИлиРавно);  periods.Загрузить();    параметры.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect3")  .WithTitle("Выбор периодов из списка")  .WithItems(periods);  });    параметры.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect4")  .WithTitle("Выбор периодов из списка множественный")  .WithItems(periods)  .Multiselect();  });  }  //...  return параметры;  }  }  } |

### Пользовательский параметр выбора учреждений

#### CommonAPI

##### DepartmentSelector

###### ../DepartmentProxy

|  |
| --- |
| namespace CommonAPI.DepartmentSelector  {  using System;  using System.Collections.Generic;  using System.Linq;  using BARS.Svody.Bars2db.Contracts;  using Барс;  using Барс.IocContainer;    /// <summary> Класс Заместителя типа Учреждение. </summary>  /// <remarks>  /// Облегчённый прокси-объект. По умолчанию обладает минимальной функциональностью и ограниченным набором хранимых данных.  /// При необходимости материализует ХО и перенаправляет запросы к нему.  /// </remarks>  public class DepartmentProxy : IУчреждение  {  private IУчреждение \_department;    /// <summary> Идентификатор учреждения </summary>  private readonly Guid \_id;    private readonly string \_name;    private readonly string \_code;    /// <summary> Конструктор прокси объекта учреждения </summary>  /// <param name="dto">Dto учреждения</param>  public DepartmentProxy(DepartmentValueDto dto) : this(dto.Id, dto.Code, dto.Name)  {  }    /// <summary> Конструктор прокси объекта учреждения </summary>  /// <param name="department">Учреждение</param>  public DepartmentProxy(IУчреждение department)  : this (department.ИдентификаторОбъекта, department.Код, department.Наименование)  {  \_department = department;  }    private DepartmentProxy(Guid id, string code, string name)  {  \_id = id;  \_code = code;  \_name = name;  }    private IУчреждение Department  {  get  {  if (\_department != null)  {  return \_department;  }    var localId = \_id;    \_department = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>()  .GetQuery<Учреждение>()  .FirstOrDefault(c => c.Идентификатор == localId);  return \_department;  }  }    /// <summary> Получить материализованный ХО учреждения </summary>  /// <returns> Материализованный ХО учреждения </returns>  public Учреждение Unwrap()  {  if (Department is Учреждение storedReportPeriod)  {  return storedReportPeriod;  }    return null;  }    public void ПеречитатьОбъект() => Department.ПеречитатьОбъект();    public Guid ИдентификаторОбъекта => Department.ИдентификаторОбъекта;    public string Наименование  {  get => \_name ?? Department.Наименование;  set => Department.Наименование = value;  }    public DateTime ДатаСозданияОбъекта => Department.ДатаСозданияОбъекта;    public string ПользовательАвторОбъекта => Department.ПользовательАвторОбъекта;    public string[] ПолучитьСписокЗначенийАтрибутов() => Department.ПолучитьСписокЗначенийАтрибутов();    public bool УстановитьСписокЗначенийАтрибутов(string[] СписокЗначений) =>  Department.УстановитьСписокЗначенийАтрибутов(СписокЗначений);    public void ОчиститьСписокЗначенийАтрибутов() => Department.ОчиститьСписокЗначенийАтрибутов();    public string ПолучитьГлаваПоБК(DateTime датаАктуальности) => Department.ПолучитьГлаваПоБК(датаАктуальности);    public List<IГлаваПоБК> ПолучитьГлавыПоБК() => Department.ПолучитьГлавыПоБК();    public IУчреждение ПолучитьУчреждениеНаДату(DateTime датаАктуальности) =>  Department.ПолучитьУчреждениеНаДату(датаАктуальности);    public string Код  {  get => \_code ?? Department.Код;  set => Department.Код = value;  }    public string КороткийКод => Department.КороткийКод;    public string СКИФ => Department.СКИФ;    public string ИНН => Department.ИНН;    public string ПФР => Department.ПФР;    public string КПП => Department.КПП;    public string ОКОНХ => Department.ОКОНХ;    public string ОКУД => Department.ОКУД;    public string ОКПО => Department.ОКПО;    public string ОКАТО => Department.ОКАТО;    public string НомерФилиала => Department.НомерФилиала;    public string ОКВЭД => Department.ОКВЭД;    public string ОКОПФ => Department.ОКОПФ;    public string ОКИН => Department.ОКИН;    public string ОКОГУ => Department.ОКОГУ;    public string ОКФС => Department.ОКФС;    public string ТипНаселенногоПункта  {  get => Department.ТипНаселенногоПункта;  set => Department.ТипНаселенногоПункта = value;  }    public string Регион  {  get => Department.Регион;  set => Department.Регион = value;  }    public string Адрес => Department.Адрес;    public IУчреждение IГлавноеУчреждение => Department.IГлавноеУчреждение;    public IВидУчреждения IВидУчреждения  {  get => Department.IВидУчреждения;  set => Department.IВидУчреждения = value;  }    public IТипУчреждения IТипУчреждения  {  get => Department.IТипУчреждения;  set => Department.IТипУчреждения = value;  }    public IТерриториальноеОтношение IТерриториальноеОтношение  {  get => Department.IТерриториальноеОтношение;  set => Department.IТерриториальноеОтношение = value;  }    public string НаселенныйПункт  {  get => Department.НаселенныйПункт;  set => Department.НаселенныйПункт = value;  }    public string ЗначениеАтрибута1  {  get => Department.ЗначениеАтрибута1;  set => Department.ЗначениеАтрибута1 = value;  }    public string ЗначениеАтрибута2  {  get => Department.ЗначениеАтрибута2;  set => Department.ЗначениеАтрибута2 = value;  }    public string ЗначениеАтрибута3  {  get => Department.ЗначениеАтрибута3;  set => Department.ЗначениеАтрибута3 = value;  }    public string ЗначениеАтрибута4  {  get => Department.ЗначениеАтрибута4;  set => Department.ЗначениеАтрибута4 = value;  }    public string ЗначениеАтрибута5  {  get => Department.ЗначениеАтрибута5;  set => Department.ЗначениеАтрибута5 = value;  }    public string ЗначениеАтрибута6  {  get => Department.ЗначениеАтрибута6;  set => Department.ЗначениеАтрибута6 = value;  }    public string ЗначениеАтрибута7  {  get => Department.ЗначениеАтрибута7;  set => Department.ЗначениеАтрибута7 = value;  }    public string ЗначениеАтрибута8  {  get => Department.ЗначениеАтрибута8;  set => Department.ЗначениеАтрибута8 = value;  }    public string ЗначениеАтрибута9  {  get => Department.ЗначениеАтрибута9;  set => Department.ЗначениеАтрибута9 = value;  }    public string ЗначениеАтрибута10  {  get => Department.ЗначениеАтрибута10;  set => Department.ЗначениеАтрибута10 = value;  }    public List<IУчреждение> ПодчиненныеУчреждения => Department.ПодчиненныеУчреждения;    public string РуководительФИО  {  get => Department.РуководительФИО;  set => Department.РуководительФИО = value;  }    public string ГлБухгалтерФИО  {  get => Department.ГлБухгалтерФИО;  set => Department.ГлБухгалтерФИО = value;  }    public string ПолноеНаименование  {  get => Department.ПолноеНаименование;  set => Department.ПолноеНаименование = value;  }    public string Улица  {  get => Department.Улица;  set => Department.Улица = value;  }    public string Корпус  {  get => Department.Корпус;  set => Department.Корпус = value;  }    public string Дом  {  get => Department.Дом;  set => Department.Дом = value;  }    public string Строение  {  get => Department.Строение;  set => Department.Строение = value;  }    public string РуковПланЭкСлужбы  {  get => Department.РуковПланЭкСлужбы;  set => Department.РуковПланЭкСлужбы = value;  }    public string НачальникБюджУпр  {  get => Department.НачальникБюджУпр;  set => Department.НачальникБюджУпр = value;  }    public string НаименованиеРод  {  get => Department.НаименованиеРод;  set => Department.НаименованиеРод = value;  }    public string ПолноеНаименованиеРод  {  get => Department.ПолноеНаименованиеРод;  set => Department.ПолноеНаименованиеРод = value;  }    public DateTime ДействуетС  {  get => Department.ДействуетС;  set => Department.ДействуетС = value;  }    public DateTime ДатаЛиквидации  {  get => Department.ДатаЛиквидации;  set => Department.ДатаЛиквидации = value;  }    public bool Ликвидировано  {  get => Department.Ликвидировано;  set => Department.Ликвидировано = value;  }    public string ОКТМО  {  get => Department.ОКТМО;  set => Department.ОКТМО = value;  }    public string КраткоеНаименование  {  get => Department.КраткоеНаименование;  set => Department.КраткоеНаименование = value;  }    public string Телефон  {  get => Department.Телефон;  set => Department.Телефон = value;  }    public string this[string константа] => Department[константа];    /// <summary> Возвращает строку наименования текущего учреждения. </summary>  /// <returns> Строковое представление текущего учреждения. </returns>  public override string ToString()  {  return Department.ToString();  }  }  } |

###### ../DepartmentSelector

|  |
| --- |
| namespace CommonAPI.DepartmentSelector  {  using System.Collections.Generic;  using System.Linq;  using Барс;    public class DepartmentSelector : BaseSelector<DepartmentValueDto, IУчреждение, IУчреждение>,  IDepartmentSelector  {  private string \_title;    private IEnumerable<Column> \_columns;    /// <summary> Заголовок параметра. </summary>  public override string Заголовок  {  get => \_title ?? (\_title = "Выбор учреждений");  set => \_title = value;  }    public IEnumerable<Column> Columns  {  get => \_columns == null || !\_columns.Any()  ? BuildColumns()  : \_columns;    set => \_columns = value;  }    protected override DepartmentValueDto CreateDtoBySelectedItem(IУчреждение selectedItem)  {  return new DepartmentValueDto(selectedItem);  }    protected override IУчреждение CreateSelectedItemByDto(DepartmentValueDto dto)  {  return new DepartmentProxy(dto);  }    public bool UseAllDepartments => Items == null || !Items.Any();    /// <summary> Формирование списка отображаемых столбцов по умолчанию </summary>  /// <returns> Список отображаемых по умолчанию столбцов </returns>  private IEnumerable<Column> BuildColumns()  {  return new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "code",  StoredDataIndex = "Код",  Header = "Код",  Width = 200  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  }  };  }  }  } |

###### ../DepartmentValueDto

|  |
| --- |
| namespace CommonAPI.DepartmentSelector  {  using System;  using Newtonsoft.Json;  using Барс;    [Serializable]  public class DepartmentValueDto  {  public DepartmentValueDto()  {    }    public DepartmentValueDto(IУчреждение department)  {  Id = department.ИдентификаторОбъекта;  Name = department.Наименование;  Code = department.Код;  }    /// <summary> Идентификатор учреждения </summary>  [JsonProperty("id")]  public Guid Id { get; set; }    /// <summary> Наименование учреждения </summary>  [JsonProperty("name")]  public string Name { get; set; }    /// <summary> Код учреждения </summary>  [JsonProperty("code")]  public string Code { get; set; }  }  } |

###### ../IDepartmentSelector

|  |
| --- |
| namespace CommonAPI.DepartmentSelector  {  using Барс;  using Приложение.ОтчетнаяФорма.UserParameters;    public interface IDepartmentSelector : IUserParameter<DepartmentValueDto, IУчреждение, IУчреждение>,  IMultiselectableParameter,  IParameterWithColumns  {  /// <summary> Использовать все учреждения </summary>  bool UseAllDepartments { get; }  }  } |

#### CommonWebAPI

##### DеpartmentSelector

###### Controllers

../DеpartmentSelector.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Controllers  {  using System.Web.Mvc;  using Models;  using Services;  using Svody.RIA.ExtjsDirectApi;    /// <summary> Контроллер обрабатывающий запросы пользовательского параметра выбора учреждений </summary>  public class DepartmentSelectorController : Controller  {  /// <summary>  /// Сервис фильтрации и пагинации записей таблицы учреждений  /// при конфигурации параметра с пустым списком учреждений для выбора  /// </summary>  public IDepartmentSelectorService DepartmentSelectorService { get; set; }    /// <summary>  /// Сервис фильтрации и пагинации записей таблицы учреждений  /// при конфигурации параметра со списком учреждений для выбора  /// </summary>  public IDepartmentSelectorFromListService DepartmentSelectorFromListService { get; set; }    /// <summary> Получить учреждения </summary>  /// <param name="gridParams">Параметры грида записей</param>  /// <returns>Страница записей</returns>  public ActionResult GetList(DepartmentSelectorParams gridParams)  {  return this.Direct(new  {  success = true,  data = ResolveService(gridParams).GetRecords(gridParams, out var totalRecordsCount),  total = totalRecordsCount  });  }    private IDepartmentSelectorService ResolveService(DepartmentSelectorParams gridParams)  {  return gridParams.UseAllDepartments  ? DepartmentSelectorService  : DepartmentSelectorFromListService;  }  }  } |

###### Js

../DepartmentSelector.js

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelector = Ext.extend(RIA.UserParameters.SelectorBase, {    rendererField: 'name',  recordValueType: {  constructor: CommonAPI.UserParameters.DepartmentSelectorValue,  fields:['id','name', 'code']  },  choiceWindow: CommonAPI.UserParameters.DepartmentSelectorWindow,  resultValueType: 'System.Collections.Generic.List`1[[CommonAPI.UserParams.DepartmentSelector.DepartmentValueDto, CommonAPI]], mscorlib',    initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorr.superclass.initComponent.call(this);  this.on('beforechoose', this.beforeShowEditor, this);  },    beforeShowEditor: function (config) {    var selectedItems = [];  Ext.each(this.value, function(rec){  selectedItems.push(rec);  });    config.baseParams.parameterId = this.parameterId;  config.baseParams.ParameterStoreOwnerId = this.parameterStoreOwnerId;    if (this.dependencies && this.dependencies.length > 0){  config.baseParams.dependencies = this.dependencies;  }  else{  config.baseParams.UseAllDepartments = this.useAllDepartments;  }    config.title = this.title;  config.columnsCfg = this.columns;  config.multiSelect = this.allowMultiSelect;  config.selectedValues = selectedItems;  config.directController = DepartmentSelector;    return true;  }  });  Ext.reg('CommonAPI.UserParameters.DepartmentSelector', CommonAPI.UserParameters.DepartmentSelector); |

../DepartmentSelectorGrid.js

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorGrid = Ext.extend(RIA.TableSelectorGrid, {  loadMask: { msg: 'Загрузка записей...' },    initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorGrid.superclass.initComponent.call(this);  },    recordEquals: function(rec1, rec2){  return !Ext.isEmpty(rec1.id) && rec1.id === rec2.id;  },    getStoreFields: function() {  var fields = [];    Ext.each(this.columnsCfg, function (column) {  fields.push({  name: column.modelDataIndex,  storedDataIndex: column.storedDataIndex  });  });  return fields;  }  });    Ext.reg('CommonAPI.UserParameters.DepartmentSelectorGrid', CommonAPI.UserParameters.DepartmentSelectorGrid); |

../DepartmentSelectorValue.js

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorValue = Ext.extend(RIA.UserParameters.ParameterValue, {  code: null  }); |

../DepartmentSelectorWindow.js

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorWindow = Ext.extend(RIA.TableSelectorWindow, {  initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorWindow.superclass.initComponent.call(this);  },    createSelectorGrid: function(){  return {  xtype: 'CommonAPI.UserParameters.DepartmentSelectorGrid',  border: false,  readOnlyMode: this.readOnlyMode,  ref: 'itemsGrid',  columnsCfg: this.config.columnsCfg,  multiSelect: this.config.multiSelect,  selectedValues: this.config.selectedValues,  directController: this.config.directController,  baseParams: this.config.baseParams || {},  listeners: {  rowSelected: function(){  this.doSelectAndClose();  },  scope: this  }  }  }  }); |

###### Models

../DepartmentSelectorEditorDto.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Models  {  using System.Collections.Generic;  using Svody.RIA.Common.Editors;    /// <summary> Модель редактора пользовательского параметра выбора учреждений </summary>  public class DepartmentSelectorEditorDto : BaseEditorViewModel  {  /// <summary> Конструктор по умолчанию. </summary>  public DepartmentSelectorEditorDto()  {  EditorType = "CommonAPI.UserParameter.DepartmentsSelector";  }    /// <summary> Идентификатор пользовательского параметра. </summary>  [JsonProperty("parameterId")]  public string ParameterId { get; set; }    /// <summary> Разрешён множественный выбор. </summary>  [JsonProperty("allowMultiSelect")]  public bool AllowMultiSelect { get; set; }    /// <summary> Значение контрола. </summary>  [JsonProperty("value")]  public object Value { get; set; }    /// <summary> Список столбцов таблицы. </summary>  [JsonProperty("columns")]  public List<Column> Columns { get; set; }    /// <summary> Использовать все учреждения. </summary>  [JsonProperty("useAllDepartments")]  public bool UseAllDepartments { get; set; }  }  } |

../DepartmentSelectorParams.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Models  {  using System;  using System.Collections.Generic;  using Барс;    /// <summary> Параметры запроса получения списка учреждений для параметра выбора учреждений </summary>  public class DepartmentSelectorParams : BaseParams  {  // <summary>Конструктор по умолчанию</summary>  public DepartmentSelectorParams()  {  Filters = new Filter[0];  }    /// <summary> Идентификатор владельца хранилища пользовательских параметров </summary>  public Guid ParameterStoreOwnerId { get; set; }    /// <summary> Идентификатор пользовательского параметра. </summary>  public string ParameterId { get; set; }    /// <summary> Использовать все учреждения</summary>  public bool UseAllDepartments { get; set; }    /// <summary> Список моделей зависимостей параметра </summary>  [JsonProperty("dependencies")]  public IEnumerable<UserParameterDependency> Dependencies { get; set; }  }  } |

###### Services

../DepartmentSelectorFromListService.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Services  {  using System.Collections.Generic;  using System.Linq;  using Models;  using Svody.RIA;  using Svody.RIA.Areas.Forms.Services.UserParameters.Dependency;    public class DepartmentSelectorFromListService : DepartmentSelectorService,  IDepartmentSelectorFromListService  {  /// <summary> Провайдер источника данных параметра </summary>  public IUserParameterDataSourceUpdater<IУчреждение> UserParameterDataSourceUpdater { get; set; }    protected override IQueryable<Учреждение> GetDepartmentsInternal(DepartmentSelectorParams gridParams)  {  if (!RIASession.FormsStore.Contains(gridParams.ParameterStoreOwnerId))  {  return new List<Учреждение>().AsQueryable();  }    var formInstance = RIASession.FormsStore[gridParams.ParameterStoreOwnerId];  var userParameterStore = formInstance.UserParameterStore;    var dataSource = UserParameterDataSourceUpdater.GetUpdatedDataSource(formInstance, gridParams.ParameterId, gridParams.Dependencies);    try  {  return dataSource.Value  .Select(department => department is DepartmentProxy proxy  ? proxy.Unwrap()  : (Учреждение)department)  .AsQueryable();  }  finally  {  userParameterStore.Add(gridParams.ParameterId, dataSource);  }  }  }  } |

../DepartmentSelectorService.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Services  {  using System;  using System.Collections.Generic;  using System.Linq;  using System.Linq.Expressions;  using Models;  using Барс;    public class DepartmentSelectorService : IDepartmentSelectorService  {  /// <summary> IQueryable провайдер</summary>  public IDataStore DataStore { get; set; }    protected virtual IQueryable<Учреждение> GetDepartmentsInternal(DepartmentSelectorParams gridParams)  {  return DataStore.GetQuery<Учреждение>().Where(op => !op.Ликвидировано);  }    public IEnumerable<DepartmentValueDto> GetRecords(DepartmentSelectorParams gridParams, out int totalRecordsCount)  {  var filtered = ApplyGridFilters(gridParams);  totalRecordsCount = filtered.Count();  return ApplyGridPaging(gridParams, filtered).ToList();  }    private IQueryable<DepartmentValueDto> ApplyGridFilters(DepartmentSelectorParams gridParams)  {  var departments = GetDepartmentsInternal(gridParams);    departments = (from filter in gridParams.Filters.Where(filter => filter.FilterData.Type == "string")  let lambdaParameter = Expression.Parameter(typeof(Учреждение), "x")  let typedParam = Expression.Convert(lambdaParameter, typeof(Учреждение))  let property = Expression.Property(typedParam, filter.Field)  let constantExpr = Expression.Constant(Convert.ToString(filter.FilterData.Value))  let filterExpr = Expression.Call(property, "Contains", null, constantExpr)  select Expression.Lambda<Func<Учреждение, bool>>(filterExpr, lambdaParameter))  .Aggregate(departments, (current, lambda) => current.Where(lambda));    return departments.Select(c => new DepartmentValueDto  {  Id = c.Идентификатор,  Code = c.Код,  Name = c.Наименование  }).OrderBy(rec => rec.Name);  }    private IQueryable<DepartmentValueDto> ApplyGridPaging(BaseParams gridParams, IQueryable<DepartmentValueDto> sourceCollection)  {  return sourceCollection  .Skip(gridParams.Start)  .Take(gridParams.Limit);  }  }  } |

../IDepartmentSelectorFromListService.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Services  {  /// <summary> Интерфейс маркерный, нужен для регистрации 2ух разных сервисов в контейнере </summary>  public interface IDepartmentSelectorFromListService : IDepartmentSelectorService  {  }  } |

../IDepartmentSelectorService.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector.Services  {  using System.Collections.Generic;  using Models;    /// <summary> Интерфейс сервиса фильтрации и пагинации записей таблицы учреждений. </summary>  public interface IDepartmentSelectorService  {  /// <summary> Получить записи с учетом фильтров и пагинации. </summary>  /// <param name="gridParams"> Параметры грида. </param>  /// <param name="totalRecordsCount"> Количество записей подходящих по фильтрам. </param>  /// <returns> Коллекция записей с учетом фильтров и пагинации. </returns>  IEnumerable<DepartmentValueDto> GetRecords(DepartmentSelectorParams gridParams, out int totalRecordsCount);  }  } |

../DepartmentSelectorEditorModelCreateRule.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector  {  using System.Collections.Generic;  using Models;  using Svody.RIA.Common;  using Svody.RIA.Common.UserParameterEditorCreator.CreateRules.BaseRules;  using Барс;    public class DepartmentSelectorEditorModelCreateRule : BaseEditorModelCreateRule<DepartmentSelector, DepartmentSelectorEditorDto>  {  protected override DepartmentSelectorEditorDto CreateInner(DepartmentSelector userParameter, FormUserParameterStore userParameterStore)  {  var departmentSelectorEditor = new DepartmentSelectorEditorDto  {  ParameterId = userParameter.ИдентификаторПараметра,  AllowMultiSelect = userParameter.Multiselect,  Columns = userParameter.Columns.ToList(),  UseAllDepartments = userParameter.UseAllDepartments  };    if(!userParameter.UseAllDepartments)  {  var dataSource = new UserParameterDataSource<IEnumerable<IУчреждение>>  {  Value = userParameter.Items  };    userParameterStore.Add(userParameter.ИдентификаторПараметра, dataSource);  }    if (userParameter.SelectedItems == null || !userParameter.SelectedItems.Any())  {  return departmentSelectorEditor;  }    if (userParameter.Multiselect)  {  departmentSelectorEditor.Value = new List<DepartmentValueDto>(userParameter.SelectedItems.Select(dep => new DepartmentValueDto(dep)));  }  else  {  var department = userParameter.SelectedItems.FirstOrDefault();    if (department != null)  {  departmentSelectorEditor.Value = new List<DepartmentValueDto>  {  new DepartmentValueDto(department)  };  }  }    return departmentSelectorEditor;  }  }  } |

../DepartmentSelectorEditorRawValueConvertRule.cs

|  |
| --- |
| namespace CommonWebAPI.DepartmentSelector  {  using Svody.RIA.Common.UserParameterRawValueConverter;  using Барс;    public class DepartmentSelectorEditorRawValueConvertRule: IUserParameterRawValueConvertRule  {  /// <summary> Правило может конвертировать значение для параметра </summary>  /// <param name="userParameter">Пользовательский параметр</param>  /// <returns>true - если правило может быть использовано для конвертации значения параметра</returns>  public bool CanConvert(ПользовательскийПараметрДляВвода userParameter)  => userParameter is IDepartmentSelector;    /// <summary> Конвертировать значение </summary>  /// <param name="convertParams">Параметры конвертации значения</param>  /// <returns>Значение отправляемое в макрос отчетной формы</returns>  public object Convert(ConvertParameters convertParams)  {  var (userParam, userParameterStore, rawValue) = convertParams;    var param = (IDepartmentSelector)userParam;  return param.GetParamValue(userParameterStore.Get(param.ИдентификаторПараметра),  rawValue) ?? param.ЗначениеПоУмолчанию;  }  }  } |

#### CommonWinAPI

##### DepartmentSelector

###### ../DepartmentSelectorWin.cs

|  |
| --- |
| namespace CommonWinAPIWIN.DepartmentSelector  {  using System;  using System.Collections.Generic;  using System.Linq;  using System.Windows.Forms;  using DevExpress.XtraEditors.Controls;  using Барс;  using Барс.IocContainer;  using Барс.Интерфейс;    public class DepartmentSelectorWin : DepartmentSelector,  IUserParameterWithDependentDataSource  {  public RepositoryItem CreateRepositoryItem()  {  var editor = new Барс.Интерфейс.ЭлементыТаблицы.КнопкаРедактор  {  DataSource = GetSelector(),  TextEditStyle = TextEditStyles.DisableTextEditor,  NullText = "Выберите значение"  };    editor.Buttons.Clear();    var selectorButton = new EditorButton  {  Tag = "select",  Kind = ButtonPredefines.Ellipsis,  ToolTip = "Выберите значение"  };    var clearButton = new EditorButton  {  Tag = "clear",  Kind = ButtonPredefines.Delete,  ToolTip = "Очистить значение"  };    editor.Buttons.Add(selectorButton);  editor.Buttons.Add(clearButton);    editor.ButtonClick += Editor\_OnButtonClick;  editor.CustomDisplayText += Editor\_CustomDisplayText;    return editor;  }    public IEnumerable<object> PrepareDataSource(IEnumerable<object> newDataSource)  {  return newDataSource.Cast<IУчреждение>()  .Select(period => new DepartmentProxy(period))  .ToList();  }    public void UpdateDataSource(BaseEdit editor, IEnumerable<object> newDataSource)  {  var buttonEdit = (Барс.Интерфейс.ЭлементыТаблицы.КнопкаРедактор)editor.Tag;  var selector = (ВыборИзСпискаЭлементов<IУчреждение>)buttonEdit.DataSource;  selector.SetSourceList(typeof(DepartmentProxy),  newDataSource.Cast<DepartmentProxy>());  }    private ВыборИзСпискаЭлементов<IУчреждение> GetSelector()  {  List<DepartmentProxy> departmentProxies;    var selector = new ВыборИзСпискаЭлементов<IУчреждение>(SelectionMode)  {  ЗаголовокФормы = Заголовок,  ФормаВыбора = ФормаВыбораИзСпискаЭлементов.Таблица,  AllowEmptySelections = true  };    var columns = Columns.Where(c => !c.Hidden).ToList();    foreach (var column in columns)  {  selector.СписокСтолбцов.ДобавитьСтолбец(column.Header, column.StoredDataIndex);  }    var dataIndexList = columns.Select(c => c.StoredDataIndex).ToList();    if (Items == null || !Items.Any())  {  departmentProxies = GetDepartmentsProxies();    }  else  {  departmentProxies = Items.Select(op => new DepartmentProxy(op)).ToList();  }    selector.СформироватьСписокЭлементовДляВыбора(typeof(DepartmentProxy), departmentProxies, dataIndexList, 0);    selector.СравнениеАссоциированныхОбъектов = DepartmentsEquals;    return selector;  }    private List<DepartmentProxy> GetDepartmentsProxies()  {  var departments = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>()  .GetQuery<Учреждение>().ToList();    return departments.Select(m => new DepartmentProxy(m)).ToList();  }    private void Editor\_OnButtonClick(object sender, ButtonPressedEventArgs e)  {  var selectedElements = new List<IУчреждение>();  try  {  var editor = (КнопкаРедактор)sender;  var buttonEdit = (Барс.Интерфейс.ЭлементыТаблицы.КнопкаРедактор)editor.Tag;  var selector = (ВыборИзСпискаЭлементов<IУчреждение>)buttonEdit.DataSource;    switch ((string)e.Button.Tag)  {  case "select":  if (Multiselect)  {  selectedElements = selector.ВыбратьСписокОбъектов(SelectedItems);  }  else  {  var defaultValue = SelectedItems?.FirstOrDefault();  var result = selector.ВыбратьОбъект(defaultValue);  selectedElements = new List<IУчреждение> { result };  }    if (selector.DialogResult == DialogResult.OK)  {  SelectedItems = selectedElements;  SetEditorValue(editor);  }  break;  case "clear":  SelectedItems = new List<IУчреждение>();  SetEditorValue(editor);  break;  }  }  catch (Exception ex)  {  throw new Exception("Ошибка выбора учреждений", ex);  }  }    private void SetEditorValue(КнопкаРедактор editor)  {  var count = SelectedItems.Count();  var viewModelList = SelectedItems.Select(el => new DepartmentValueDto(el)).ToList();    editor.ВыбранныйОбъект = count > 0  ? viewModelList  : null;  }    private void Editor\_CustomDisplayText(object sender, CustomDisplayTextEventArgs e)  {  if (e.Value is IEnumerable<DepartmentValueDto> departmentValueDtos)  {  var count = departmentValueDtos.Count();  e.DisplayText = Multiselect  ? $@"Выбрано {count} учреждений"  : departmentValueDtos.First().Name;  }  else  {  e.DisplayText = "Выберите значение";  }  }    private bool DepartmentsEquals(IУчреждение object1, IУчреждение object2)  {  return object1.ИдентификаторОбъекта.Equals(object2.ИдентификаторОбъекта);  }  }  } |

### Пользовательский параметр выбора элемента цепочки

Реализован пользовательский параметр выбора элемента цепочки – «ChainElementSelector».

Возвращаемый тип параметра всегда List<ChainElementValueDto>, даже если конфигурация параметра с одиночным выбором

Пример использования параметра «ChainElementSelectorParameter»:

|  |
| --- |
| namespace Барс.Своды  {  public partial class МодульФормы\_0503161 : БазовыйОбработчикФормы  {  protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string operation)  {  var parametersList = new СписокПользовательскихПараметров();  if (operation == "Test")  {  parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборЭлементаЦепочки1")  .WithTitle("Выбор элемента(ов) цепочки сдачи отчетности1")  .ApplyConfig(  new  {  ChainCode = "dev"  });  });  var ec = Форма.Идентификатор.КомпонентОтчетногоПериода.IЦепочкаСдачи.Элементы;  parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборЭлементаЦепочки2")  .WithTitle("Выбор элемента(ов) цепочки сдачи отчетности2")  .WithItems(ec)  .WithDefaultValue(ec.Take(2));  });    parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборЭлементаЦепочки3")  .WithTitle("Выбор элемента(ов) цепочки сдачи отчетности3")  .Multiselect()  .ApplyConfig(  new  {  ChainCode = "dev"  });  });  parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  parameter.WithId("ПараметрВыборЭлементаЦепочки4")  .WithTitle("Выбор элемента(ов) цепочки сдачи отчетности4")  .WithItems(ec)  .WithDefaultValue(ec.Take(2))  .Multiselect();  });  }  return parametersList;  }    [АтрибутПунктаОбработки("Test")]  public void Test()  {  var value1 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementValueDto>>("ПараметрВыборЭлементаЦепочки1");  result.Add(string.Format("код {0}", value1.First().Name));    var value2 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementValueDto>>("ПараметрВыборЭлементаЦепочки2");  result.Add(string.Format("код {0}", value2.First().Name));    var value3 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementValueDto>>("ПараметрВыборЭлементаЦепочки3");  result.Add(string.Format("код {0}", value3.First().Name));    var value4 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementValueDto>>("ПараметрВыборЭлементаЦепочки4");  result.Add(string.Format("код {0}", value4.First().Name));  }  }  } |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* ChainCode.

### Пользовательский параметр выбора элементов цепочки из компонентов

Реализован пользовательский параметр выбора элемента цепочки – «ChainElementFromComponentSelector».

Возвращаемый тип параметра всегда List<ChainElementFromComponentValueDto>, даже если конфигурация параметра с одиночным выбором.

Пример использования параметра «ChainElementFromComponentSelector»:

|  |
| --- |
| namespace Барс.Своды  {  public partial class МодульФормы\_0503161 : БазовыйОбработчикФормы  {  protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string operation)  {  var parametersList = new СписокПользовательскихПараметров();  if (operation == "Test")  {  // На выбор даются все компоненты текущего отчетного периода  // с одиночным выбором компонента и одиночным выбором элементов  параметры.BuildParameter<IChainElementFromComponentSelector>(  parameter =>  {  parameter.WithId("id1")  .WithTitle("Компоненты - од.выбор, ЭЦ - од.выбор")  .ApplyConfig(  new  {  ReportPeriodCode =  Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.Код  });  });  // На выбор даются 2 компонента текущего отчетного периода  // с множественным выбором компонентов и одиночным выбором элементов  параметры.BuildParameter<IChainElementFromComponentSelector>(  parameter =>  {  parameter.WithId("id2")  .WithTitle("Компоненты - мн.выбор, ЭЦ - од.выбор")  .Multiselect()  .WithItems(((ОтчетныйПериод)Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод).Компоненты.Take(2));  });  // На выбор даются 3 компонента текущего отчетного периода  // с одиночным выбором компонента и множественным выбором элементов  параметры.BuildParameter<IChainElementFromComponentSelector>(  parameter =>  {  parameter.WithId("id3")  .WithTitle("Компоненты - од.выбор, ЭЦ - мн.выбор")  .WithItems(((ОтчетныйПериод)Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод).Компоненты.Take(2))  .ApplyConfig(  new  {  ElementsMultiselect = true  });  });  // На выбор даются все компоненты текущего отчетного периода  // с множественным выбором компонентов и множественным выбором элементов  // ЭЦ текущего компонента выбраны по умолчанию  параметры.BuildParameter<IChainElementFromComponentSelector>(  parameter =>  {  parameter.WithId("id4")  .WithTitle("Компоненты - мн. выбор, ЭЦ - мн. выбор")  .WithItems(((ОтчетныйПериод)Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод).Компоненты.Take(2))  .Multiselect()  .WithDefaultValue(new List<ComponentElementsComposite>  {  new ComponentElementsComposite  (  Форма.Идентификатор.КомпонентОтчетногоПериода,  Форма.Идентификатор.КомпонентОтчетногоПериода.IЦепочкаСдачи.Элементы  )  })  .ApplyConfig(  new  {  ReportPeriodCode = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.Код,  ElementsMultiselect = true  });  });  }  return parametersList;  }    [АтрибутПунктаОбработки("Test")]  public void Test()  {  var result = new List<string>();  var value1 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementFromComponentValueDto>>("id1");  result.Add(string.Format("код компонента {0}, наименование ЭЦ {1}", value1 .First().Component.Code,  value1 .First().Elements.First().Name));    var value2 = СписокПараметровОперации.GetParameterValueAs<List<ChainElementFromComponentValueDto>>("id2");  result.Add(string.Format("код компонента {0}, наименование ЭЦ {1}", value1 .First().Component.Code,  value1 .First().Elements.First().Name));  }  }  } |

Список свойств, которые можно сконфигурировать только с помощью метода «ApplyConfig»:

* ReportPeriodCode;
* ElementsMultiselect.

### Создание кастомного параметра

#### Общая логика

Общая логика создания кастомного параметра:

1. реализуйте DTO значения параметра:

|  |
| --- |
| using System;  using Newtonsoft.Json;  using Барс;    [Serializable]  public class DepartmentValueDto  {  public DepartmentValueDto()  {    }    public DepartmentValueDto(IУчреждение department)  {  Id = department.ИдентификаторОбъекта;  Name = department.Наименование;  Code = department.Код;  }    /// <summary> Идентификатор учреждения </summary>  [JsonProperty("id")]  public Guid Id { get; set; }    /// <summary> Наименование учреждения </summary>  [JsonProperty("name")]  public string Name { get; set; }    /// <summary> Код учреждения </summary>  [JsonProperty("code")]  public string Code { get; set; }  } |

Помечайте свойства атрибутами для Json-сериализации, т.к. данная модель будет использоваться для передачи на клиент, и она же будет храниться в кэше пользовательских параметров win-клиента.

1. реализуйте Proxy-объект для сущности «Iучреждение»:

|  |
| --- |
| namespace CommonAPI.DepartmentSelector  {  using System;  using System.Collections.Generic;  using System.Linq;  using BARS.Svody.Bars2db.Contracts;  using Барс;  using Барс.IocContainer;  /// <summary> Класс Заместителя типа Учреждение. </summary>  /// <remarks>  /// Облегчённый прокси-объект. По умолчанию обладает минимальной функциональностью и ограниченным набором хранимых данных.  /// При необходимости материализует ХО и перенаправляет запросы к нему.  /// </remarks>  public class DepartmentProxy : IУчреждение  {  private IУчреждение \_department;    /// <summary> Идентификатор учреждения </summary>  private readonly Guid \_id;  private readonly string \_name;    private readonly string \_code;    /// <summary> Конструктор прокси объекта учреждения </summary>  /// <param name="dto">Dto учреждения</param>  public DepartmentProxy(DepartmentValueDto dto) : this(dto.Id, dto.Code, dto.Name)  {  }  /// <summary> Конструктор прокси объекта учреждения </summary>  /// <param name="department">Учреждение</param>  public DepartmentProxy(IУчреждение department)  : this (department.ИдентификаторОбъекта, department.Код, department.Наименование)  {  \_department = department;  }  private DepartmentProxy(Guid id, string code, string name)  {  \_id = id;  \_code = code;  \_name = name;  }  private IУчреждение Department  {  get  {  if (\_department != null)  {  return \_department;  }    var localId = \_id;  \_department = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>()  .GetQuery<Учреждение>()  .FirstOrDefault(c => c.Идентификатор == localId);  return \_department;  }  }    /// <summary> Получить материализованный ХО учреждения </summary>  /// <returns> Материализованный ХО учреждения </returns>  public Учреждение Unwrap()  {  if (Department is Учреждение storedReportPeriod)  {  return storedReportPeriod;  }  return null;  }    public void ПеречитатьОбъект() => Department.ПеречитатьОбъект();  public Guid ИдентификаторОбъекта => Department.ИдентификаторОбъекта;    public string Наименование  {  get => \_name ?? Department.Наименование;  set => Department.Наименование = value;  }    public DateTime ДатаСозданияОбъекта => Department.ДатаСозданияОбъекта;  public string ПользовательАвторОбъекта => Department.ПользовательАвторОбъекта;  public string[] ПолучитьСписокЗначенийАтрибутов() => Department.ПолучитьСписокЗначенийАтрибутов();  public bool УстановитьСписокЗначенийАтрибутов(string[] СписокЗначений) =>  Department.УстановитьСписокЗначенийАтрибутов(СписокЗначений);  public void ОчиститьСписокЗначенийАтрибутов() => Department.ОчиститьСписокЗначенийАтрибутов();  public string ПолучитьГлаваПоБК(DateTime датаАктуальности) => Department.ПолучитьГлаваПоБК(датаАктуальности);  public List<IГлаваПоБК> ПолучитьГлавыПоБК() => Department.ПолучитьГлавыПоБК();  public IУчреждение ПолучитьУчреждениеНаДату(DateTime датаАктуальности) =>  Department.ПолучитьУчреждениеНаДату(датаАктуальности);  public string Код  {  get => \_code ?? Department.Код;  set => Department.Код = value;  }    public string КороткийКод => Department.КороткийКод;    public string СКИФ => Department.СКИФ;    public string ИНН => Department.ИНН;    public string ПФР => Department.ПФР;    public string КПП => Department.КПП;    public string ОКОНХ => Department.ОКОНХ;    public string ОКУД => Department.ОКУД;    public string ОКПО => Department.ОКПО;    public string ОКАТО => Department.ОКАТО;    public string НомерФилиала => Department.НомерФилиала;  public string ОКВЭД => Department.ОКВЭД;    public string ОКОПФ => Department.ОКОПФ;    public string ОКИН => Department.ОКИН;    public string ОКОГУ => Department.ОКОГУ;    public string ОКФС => Department.ОКФС;  public string ТипНаселенногоПункта  {  get => Department.ТипНаселенногоПункта;  set => Department.ТипНаселенногоПункта = value;  }    public string Регион  {  get => Department.Регион;  set => Department.Регион = value;  }    public string Адрес => Department.Адрес;  public IУчреждение IГлавноеУчреждение => Department.IГлавноеУчреждение;    public IВидУчреждения IВидУчреждения  {  get => Department.IВидУчреждения;  set => Department.IВидУчреждения = value;  }    public IТипУчреждения IТипУчреждения  {  get => Department.IТипУчреждения;  set => Department.IТипУчреждения = value;  }    public IТерриториальноеОтношение IТерриториальноеОтношение  {  get => Department.IТерриториальноеОтношение;  set => Department.IТерриториальноеОтношение = value;  }    public string НаселенныйПункт  {  get => Department.НаселенныйПункт;  set => Department.НаселенныйПункт = value;  }    public string ЗначениеАтрибута1  {  get => Department.ЗначениеАтрибута1;  set => Department.ЗначениеАтрибута1 = value;  }    public string ЗначениеАтрибута2  {  get => Department.ЗначениеАтрибута2;  set => Department.ЗначениеАтрибута2 = value;  }    public string ЗначениеАтрибута3  {  get => Department.ЗначениеАтрибута3;  set => Department.ЗначениеАтрибута3 = value;  }    public string ЗначениеАтрибута4  {  get => Department.ЗначениеАтрибута4;  set => Department.ЗначениеАтрибута4 = value;  }    public string ЗначениеАтрибута5  {  get => Department.ЗначениеАтрибута5;  set => Department.ЗначениеАтрибута5 = value;  }    public string ЗначениеАтрибута6  {  get => Department.ЗначениеАтрибута6;  set => Department.ЗначениеАтрибута6 = value;  }    public string ЗначениеАтрибута7  {  get => Department.ЗначениеАтрибута7;  set => Department.ЗначениеАтрибута7 = value;  }    public string ЗначениеАтрибута8  {  get => Department.ЗначениеАтрибута8;  set => Department.ЗначениеАтрибута8 = value;  }    public string ЗначениеАтрибута9  {  get => Department.ЗначениеАтрибута9;  set => Department.ЗначениеАтрибута9 = value;  }    public string ЗначениеАтрибута10  {  get => Department.ЗначениеАтрибута10;  set => Department.ЗначениеАтрибута10 = value;  }  public List<IУчреждение> ПодчиненныеУчреждения => Department.ПодчиненныеУчреждения;    public string РуководительФИО  {  get => Department.РуководительФИО;  set => Department.РуководительФИО = value;  }    public string ГлБухгалтерФИО  {  get => Department.ГлБухгалтерФИО;  set => Department.ГлБухгалтерФИО = value;  }    public string ПолноеНаименование  {  get => Department.ПолноеНаименование;  set => Department.ПолноеНаименование = value;  }    public string Улица  {  get => Department.Улица;  set => Department.Улица = value;  }    public string Корпус  {  get => Department.Корпус;  set => Department.Корпус = value;  }    public string Дом  {  get => Department.Дом;  set => Department.Дом = value;  }    public string Строение  {  get => Department.Строение;  set => Department.Строение = value;  }    public string РуковПланЭкСлужбы  {  get => Department.РуковПланЭкСлужбы;  set => Department.РуковПланЭкСлужбы = value;  }    public string НачальникБюджУпр  {  get => Department.НачальникБюджУпр;  set => Department.НачальникБюджУпр = value;  }    public string НаименованиеРод  {  get => Department.НаименованиеРод;  set => Department.НаименованиеРод = value;  }    public string ПолноеНаименованиеРод  {  get => Department.ПолноеНаименованиеРод;  set => Department.ПолноеНаименованиеРод = value;  }    public DateTime ДействуетС  {  get => Department.ДействуетС;  set => Department.ДействуетС = value;  }    public DateTime ДатаЛиквидации  {  get => Department.ДатаЛиквидации;  set => Department.ДатаЛиквидации = value;  }    public bool Ликвидировано  {  get => Department.Ликвидировано;  set => Department.Ликвидировано = value;  }    public string ОКТМО  {  get => Department.ОКТМО;  set => Department.ОКТМО = value;  }    public string КраткоеНаименование  {  get => Department.КраткоеНаименование;  set => Department.КраткоеНаименование = value;  }    public string Телефон  {  get => Department.Телефон;  set => Department.Телефон = value;  }  public string this[string константа] => Department[константа];    /// <summary> Возвращает строку наименования текущего учреждения. </summary>  /// <returns> Строковое представление текущего учреждения. </returns>  public override string ToString()  {  return Department.ToString();  }  }  } |

1. реализуйте класс пользовательского параметра.

Рассмотрим пример реализации параметра выбора учреждений из линейного списка, похожий по своей логике параметр реализован в платформе –IReportPeriodSelector, поэтому пример реализации будет взят с него.

1. добавьте в проект интерфейс параметра – IdepartmentSelector;
2. наследуйте интерфейс от интерфейса **IUserParameter<out TValueDto, in TItemType, TSelectedItemType>**, в качестве TValueDto используйте тип DepartmentValueDto, реализованный на первом шаге, в качестве TItemType и TSelectedItemType используйте IУчреждение.

|  |
| --- |
| /// <summary> Общий интерфейс пользовательского параметра. </summary>  /// <typeparam name="TValueDto"> Тип dto элемента списка значений параметра. </typeparam>  /// <typeparam name="TItemType"> Тип элемента списка значений параметра</typeparam>  /// <typeparam name="TSelectedItemType"> Тип выбранного элемента списка значений параметра</typeparam>  public interface IUserParameter<out TValueDto, in TItemType, TSelectedItemType> : IBaseUserParameter,  IUserParameterSerialization<TValueDto>,  IParameterWithItems<TItemType>  {  /// <summary> Список идентификаторов подчиненных параметров </summary>  List<string> ПодчиненныеПараметры { get; set; }    /// <summary> Значение параметра по умолчанию </summary>  object ЗначениеПоУмолчанию { get; set; }    /// <summary> Тэг параметра </summary>  object Тэг { get; set; }    /// <summary> Список выбранных элементов </summary>  IEnumerable<TSelectedItemType> SelectedItems { get; set; }    /// <summary> Получить значение параметра в зависимости от переданного с клиента значения </summary>  /// <param name="paramDataStore">Источник данных параметра</param>  /// <param name="rawClientValue">Значение переданное с клиента</param>  /// <returns> Значение параметра восстановленного типа </returns>  IEnumerable<TValueDto> GetParamValue(object paramDataStore, object rawClientValue);  } |

Сам IUserParameter<out TValueDto, in TItemType, TSelectedItemType> наследует еще 2 интерфейса:

|  |
| --- |
| /// <summary> Содержит методы сериализации пользовательских параметров. </summary>  public interface IUserParameterSerialization<out TResult>  {  /// <summary> Сериализовать значение для сохранения в кэш </summary>  /// <param name="currentValue">Значение параметра</param>  /// <returns>Значение параметра для сохранения в кэше</returns>  string SerializeForCache(object currentValue);    /// <summary> Десериализовать значение параметра из кэша </summary>  /// <param name="cachedParam">Dto кэшированного параметра</param>  /// <returns> Значение параметра восстановленного типа </returns>  IEnumerable<TResult> DeserializeFromCache(ICachedUserParameterDto cachedParam);  } |

|  |
| --- |
| /// <summary> Интерфейс параметра со списком элементов для выбора </summary>  /// <typeparam name="TItemType">Тип элемента списка для выбора</typeparam>  public interface IParameterWithItems<in TItemType>  {  /// <summary> Установить список элементов для выбора </summary>  /// <param name="items"> Список элементов </param>  void SetItems(IEnumerable<TItemType> items);  } |

1. наследуйте интерфейс от интерфейса ImultiselectableParameter, добавляя таким образом возможность множественного выбора к параметру;
2. наследуйте интерфейс от интерфейса IparameterWithColumns, добавляя таким образом возможность задавать отображаемые столбцы;
3. добавьте свойство UseAllDepartments для возможности выбирать учреждения либо из конкретного списка, либо из списка всех учреждений существующих в Системе;

|  |
| --- |
| public interface IDepartmentSelector : IUserParameter<DepartmentValueDto, IУчреждение, IУчреждение>,  IMultiselectableParameter,  IParameterWithColumns  {  /// <summary> Использовать все учреждения </summary>  bool UseAllDepartments { get; }  } |

1. добавьте в проект класс параметра – DepartmentSelector;
2. наследуйте DepartmentSelector от класса BaseSelector<TValueDto, TItemType, TSelectedItemType> – для получения автоматической реализации интерфейсов;

IUserParameterSerialization<TValueDto>, IParameterWithItems<TItemType>, IMultiselectableParameter, IBaseUserParameter,   
IUserParameter<out TValueDto, in TItemType, TSelectedItemType>

и остается только реализовать интерфейсы:

IParameterWithColumns и IdepartmentSelector

и абстрактные члены класса BaseSelector<TValueDto, TItemType, TSelectedItemType>.

1. реализуйте интерфейс IDepartmentSelector таким образом, чтобы «флажок» UseAllDepartments принимал значение «true» в том случае, если не задан список элементов для выбора (Items свойство базового класса);

|  |
| --- |
| public bool UseAllDepartments => Items == null || !Items.Any(); |

1. реализуйте интерфейс IparameterWithColumns:

|  |
| --- |
| private IEnumerable<Column> \_columns;    /// <summary> Отображаемые столбцы </summary>  public IEnumerable<Column> Columns  {  get => \_columns == null || !\_columns.Any()  ? BuildColumns()  : \_columns;    set => \_columns = value;  }    /// <summary> Формирование списка отображаемых столбцов по умолчанию </summary>  /// <returns> Список отображаемых по умолчанию столбцов </returns>  private IEnumerable<Column> BuildColumns()  {  return new List<Column>  {  new Column  {  ModelDataIndex = "id",  StoredDataIndex = "Идентификатор",  Header = "Идентификатор",  Hidden = true,  Width = 400  },  new Column  {  ModelDataIndex = "code",  StoredDataIndex = "Код",  Header = "Код",  Width = 200  },  new Column  {  ModelDataIndex = "name",  StoredDataIndex = "Наименование",  Header = "Наименование",  Width = 200  }  };  } |

1. реализуйте абстрактные члены класса BaseSelector<TValueDto, TItemType, TSelectedItemType>:

|  |
| --- |
| protected override DepartmentValueDto CreateDtoBySelectedItem(IУчреждение selectedItem)  {  return new DepartmentValueDto(selectedItem);  }    protected override IУчреждение CreateSelectedItemByDto(DepartmentValueDto dto)  {  return new DepartmentProxy(dto);  } |

#### WEB

Создание кастомного параметр в Web:

1. реализуйте модель редактора параметра:

|  |
| --- |
| using System.Collections.Generic;  using Newtonsoft.Json;  using Svody.RIA.Common.Editors;  using Барс;    /// <summary> Модель редактора пользовательского параметра выбора учреждений </summary>  public class DepartmentSelectorEditorDto : BaseEditorViewModel  {  /// <summary> Конструктор по умолчанию. </summary>  public DepartmentSelectorEditorDto()  {  EditorType = "CommonAPI.UserParameters.DepartmentSelector";  }    /// <summary> Идентификатор пользовательского параметра. </summary>  [JsonProperty("parameterId")]  public string ParameterId { get; set; }    /// <summary> Разрешён множественный выбор. </summary>  [JsonProperty("allowMultiSelect")]  public bool AllowMultiSelect { get; set; }    /// <summary> Значение контрола. </summary>  [JsonProperty("value")]  public object Value { get; set; }    /// <summary> Список столбцов таблицы. </summary>  [JsonProperty("columns")]  public List<Column> Columns { get; set; }    /// <summary> Использовать все учреждения. </summary>  [JsonProperty("useAllDepartments")]  public bool UseAllDepartments { get; set; }  } |

В EditorType укажите название класса редактора в Ext.js.

1. реализуйте модель параметров запроса получения списка значений.

Данная модель содержит параметры фильтрации и пагинации (в базовом классе BaseParams), список зависимостей, идентификатор параметра, идентификатор владельца хранилища пользовательских параметров.

|  |
| --- |
| using System;  using System.Collections.Generic;  using Newtonsoft.Json;  using Барс;    /// <summary> Параметры запроса получения списка учреждений для параметра выбора учреждений </summary>  public class DepartmentSelectorParams : BaseParams  {  // <summary>Конструктор по умолчанию</summary>  public DepartmentSelectorParams()  {  Filters = new Filter[0];  }    /// <summary> Идентификатор владельца хранилища пользовательских параметров </summary>  public Guid ParameterStoreOwnerId { get; set; }    /// <summary> Идентификатор пользовательского параметра. </summary>  public string ParameterId { get; set; }    /// <summary> Использовать все учреждения</summary>  public bool UseAllDepartments { get; set; }    /// <summary> Список моделей зависимостей параметра </summary>  [JsonProperty("dependencies")]  public IEnumerable<UserParameterDependency> Dependencies { get; set; }  } |

1. реализуйте правило для создания редактора параметра.

Наследуйте от класса BaseEditorModelCreateRule<TParam, TEditor>, в качестве TParam укажите тип вашего параметра DepartmentSelector, а в качестве TEditor – тип модели редактора DepartmentSelectorEditorDto.

|  |
| --- |
| using System.Collections.Generic;  using System.Linq;  using CommonAPI.UserParams.DepartmentSelector;  using Svody.RIA.Common;  using Svody.RIA.Common.UserParameterEditorCreator.CreateRules.BaseRules;  using Барс;    public class DepartmentSelectorEditorModelCreateRule : BaseEditorModelCreateRule<DepartmentSelector, DepartmentSelectorEditorDto>  {  protected override DepartmentSelectorEditorDto CreateInner(DepartmentSelector userParameter, FormUserParameterStore userParameterStore)  {  var departmentSelectorEditor = new DepartmentSelectorEditorDto  {  ParameterId = userParameter.ИдентификаторПараметра,  AllowMultiSelect = userParameter.Multiselect,  Columns = userParameter.Columns.ToList(),  UseAllDepartments = userParameter.UseAllDepartments  };    // Если список учреждений задан изначально - создаем модель источника данных и помещаем её в хранилище  if(!userParameter.UseAllDepartments)  {  var dataSource = new UserParameterDataSource<IEnumerable<IУчреждение>>  {  Value = userParameter.Items  };    userParameterStore.Add(userParameter.ИдентификаторПараметра, dataSource);  }    if (userParameter.SelectedItems == null || !userParameter.SelectedItems.Any())  {  return departmentSelectorEditor;  }    // Если есть выбранные учреждения - указываем их в качестве списка в свойстве Value редактора  if (userParameter.Multiselect)  {  departmentSelectorEditor.Value = new List<DepartmentValueDto>(userParameter.SelectedItems.Select(dep => new DepartmentValueDto(dep)));  }  else  {  var department = userParameter.SelectedItems.FirstOrDefault();    if (department != null)  {  departmentSelectorEditor.Value = new List<DepartmentValueDto>  {  new DepartmentValueDto(department)  };  }  }    return departmentSelectorEditor;  } |

1. реализуйте конвертер значений для параметра:

|  |
| --- |
| using Барс;    using CommonAPI.UserParams.DepartmentSelector;    using Svody.RIA.Common.UserParameterRawValueConverter;    public class DepartmentSelectorEditorRawValueConvertRule: IUserParameterRawValueConvertRule  {  /// <summary> Правило может конвертировать значение для параметра </summary>  /// <param name="userParameter">Пользовательский параметр</param>  /// <returns>true - если правило может быть использовано для конвертации значения параметра</returns>  public bool CanConvert(ПользовательскийПараметрДляВвода userParameter)  => userParameter is IDepartmentSelector;    /// <summary> Конвертировать значение </summary>  /// <param name="convertParams">Параметры конвертации значения</param>  /// <returns>Значение отправляемое в макрос отчетной формы</returns>  public object Convert(ConvertParameters convertParams)  {  var (userParam, userParameterStore, rawValue) = convertParams;    var param = (IDepartmentSelector)userParam;  return param.GetParamValue(userParameterStore.Get(param.ИдентификаторПараметра),  rawValue) ?? param.ЗначениеПоУмолчанию;  }  } |

1. реализуйте вспомогательные сервисы получения значений параметра;
2. добавьте в проект интерфейс IdepartmentSelectorService:

|  |
| --- |
| using System.Collections.Generic;  using CommonAPI.UserParams.DepartmentSelector;  using Models.DepartmentSelector;    /// <summary> Интерфейс сервиса фильтрации и пагинации записей таблицы учреждений. </summary>  public interface IDepartmentSelectorService  {  /// <summary> Получить записи с учетом фильтров и пагинации. </summary>  /// <param name="gridParams"> Параметры грида. </param>  /// <param name="totalRecordsCount"> Количество записей подходящих по фильтрам. </param>  /// <returns> Коллекция записей с учетом фильтров и пагинации. </returns>  IEnumerable<DepartmentValueDto> GetRecords(DepartmentSelectorParams gridParams, out int totalRecordsCount);  } |

1. добавьте в проект интерфейс IdepartmentSelectorFromListService:

|  |
| --- |
| /// <summary> Интерфейс маркерный, нужен для регистрации 2ух разных сервисов в контейнере </summary>  public interface IDepartmentSelectorFromListService : IDepartmentSelectorService  {  } |

1. реализуйте интерфейс IDepartmentSelectorService классом DepartmentSelectorService:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Linq.Expressions;  using BARS.Svody.Bars2db.Contracts;  using CommonAPI.UserParams.DepartmentSelector;  using Models.DepartmentSelector;  using Барс;    public class DepartmentSelectorService : IDepartmentSelectorService  {  /// <summary> IQueryable провайдер</summary>  public IDataStore DataStore { get; set; }    protected virtual IQueryable<Учреждение> GetDepartmentsInternal(DepartmentSelectorParams gridParams)  {  return DataStore.GetQuery<Учреждение>().Where(op => !op.Ликвидировано);  }    public IEnumerable<DepartmentValueDto> GetRecords(DepartmentSelectorParams gridParams, out int totalRecordsCount)  {  var filtered = ApplyGridFilters(gridParams);  totalRecordsCount = filtered.Count();  return ApplyGridPaging(gridParams, filtered).ToList();  }    private IQueryable<DepartmentValueDto> ApplyGridFilters(DepartmentSelectorParams gridParams)  {  var departments = GetDepartmentsInternal(gridParams);    departments = (from filter in gridParams.Filters.Where(filter => filter.FilterData.Type == "string")  let lambdaParameter = Expression.Parameter(typeof(Учреждение), "x")  let typedParam = Expression.Convert(lambdaParameter, typeof(Учреждение))  let property = Expression.Property(typedParam, filter.Field)  let constantExpr = Expression.Constant(Convert.ToString(filter.FilterData.Value))  let filterExpr = Expression.Call(property, "Contains", null, constantExpr)  select Expression.Lambda<Func<Учреждение, bool>>(filterExpr, lambdaParameter))  .Aggregate(departments, (current, lambda) => current.Where(lambda));    return departments.Select(c => new DepartmentValueDto  {  Id = c.Идентификатор,  Code = c.Код,  Name = c.Наименование  }).OrderBy(rec => rec.Name);  }    private IQueryable<DepartmentValueDto> ApplyGridPaging(BaseParams gridParams, IQueryable<DepartmentValueDto> sourceCollection)  {  return sourceCollection  .Skip(gridParams.Start)  .Take(gridParams.Limit);  }  } |

GetRecords – основной метод получения списка учреждений в виде DTO.

ApplyGridFilters – метод, применяющий фильтры, заданные в гриде выбора учреждений на клиенте к выборке учреждений.

ApplyGridPaging – метод, применяющий параметры пагинации грида выбора учреждений.

1. реализуйте интерфейс IDepartmentSelectorFromListService классом DepartmentSelectorFromListService и наследуйте его от DepartmentSelectorService:

|  |
| --- |
| using System.Collections.Generic;  using System.Linq;  using CommonAPI.UserParams.DepartmentSelector;  using Models.DepartmentSelector;  using Svody.RIA;  using Svody.RIA.Areas.Forms.Services.UserParameters.Dependency;    public class DepartmentSelectorFromListService : DepartmentSelectorService,  IDepartmentSelectorFromListService  {  /// <summary> Провайдер источника данных параметра </summary>  public IUserParameterDataSourceUpdater<IУчреждение> UserParameterDataSourceUpdater { get; set; }    protected override IQueryable<Учреждение> GetDepartmentsInternal(DepartmentSelectorParams gridParams)  {  if (!RIASession.FormsStore.Contains(gridParams.ParameterStoreOwnerId))  {  return new List<Учреждение>().AsQueryable();  }    var formInstance = RIASession.FormsStore[gridParams.ParameterStoreOwnerId];  var userParameterStore = formInstance.UserParameterStore;    // получаем источник данных параметра по зависимостям переданным в модели  // в случае если зависимостей нет - просто получаем значение из хранилища  var dataSource = UserParameterDataSourceUpdater.GetUpdatedDataSource(formInstance, gridParams.ParameterId, gridParams.Dependencies);    try  {  return dataSource.Value  .Select(department => department is DepartmentProxy proxy  ? proxy.Unwrap()  : (Учреждение)department)  .AsQueryable();  }  finally  {  // складываем полученный источник данных обратно в хранилище чтобы при последующих обращениях получать данный источник  userParameterStore.Add(gridParams.ParameterId, dataSource);  }  }  } |

Переопределите метод GetDepartmentsInternal и получите список учреждений из хранилища источников данных пользовательских параметров. Само хранилище находится в рамках текущей сессии пользователя. Также используйте IUserParameterDataSourceUpdater<IУчреждение>, т.к. нужно сразу заложить возможность того, что этот параметр может зависеть от других параметров, с помощью реализации IUserParameterDataSourceUpdater платформа обратится к классу, реализующему функциональность зависимостей для получения списка значений.

1. реализуйте контроллер, возвращающий список значений для параметра.

|  |
| --- |
| using Models.DepartmentSelector;  using Services.DepartmentSelector;    /// <summary> Контроллер обрабатывающий запросы пользовательского параметра выбора учреждений </summary>  public class DepartmentSelectorController : Controller  {  /// <summary>  /// Сервис фильтрации и пагинации записей таблицы учреждений  /// при конфигурации параметра с пустым списком учреждений для выбора  /// </summary>  public IDepartmentSelectorService DepartmentSelectorService { get; set; }    /// <summary>  /// Сервис фильтрации и пагинации записей таблицы учреждений  /// при конфигурации параметра со списком учреждений для выбора  /// </summary>  public IDepartmentSelectorFromListService DepartmentSelectorFromListService { get; set; }    /// <summary> Получить учреждения </summary>  /// <param name="gridParams">Параметры грида записей</param>  /// <returns>Страница записей</returns>  public ActionResult GetList(DepartmentSelectorParams gridParams)  {  return this.Direct(new  {  success = true,  data = ResolveService(gridParams).GetRecords(gridParams, out var totalRecordsCount),  total = totalRecordsCount  });  }    private IDepartmentSelectorService ResolveService(DepartmentSelectorParams gridParams)  {  return gridParams.UseAllDepartments  ? DepartmentSelectorService  : DepartmentSelectorFromListService;  }  } |

1. зарегистрируйте зависимости:

|  |
| --- |
| container.Register(  Component.For<IDepartmentSelector>().ImplementedBy<DepartmentSelector>().IsDefault().LifestyleTransient(), // регистрируем параметр по умолчанию  Component.For<IDepartmentSelectorService>().ImplementedBy<DepartmentSelectorService>().LifestyleSingleton(), // регистрируем сервисы  Component.For<IDepartmentSelectorFromListService>().ImplementedBy<DepartmentSelectorFromListService>().LifestyleSingleton());    var assembly = GetType().Assembly;  container.RegisterUserParameterEditorCreators(assembly); // регистрируем правила создания моделей редакторов  container.RegisterUserParameterRawValueConverters(assembly); // регистрируем конвертеры значений |

1. реализуйте js-логику редактора:
   1. реализуйте класc значения. Наследуйте от RIA.UserParameters.ParameterValue:

|  |
| --- |
| RIA.UserParameters.ParameterValue = Ext.extend(Object, {  id: null,  name: null,  toString: function () {  return this.name;  }  }); |

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorValue = Ext.extend(RIA.UserParameters.ParameterValue, {  code: null  }); |

* 1. реализуйте класс окна. Наследуйте от RIA.TableSelectorWindow и переопределите метод createSelectorGrid:

|  |
| --- |
| Ext.ns('RIA');    RIA.TableSelectorWindow = Ext.extend(RIA.BaseWindow, {  closable:true,  resizable:true,  bodyBorder:false,  layout:'fit',  autoScroll:true,  modal:true,  width:550,  height:400,    multiSelect:true,  parameterId: null,  parameterStoreOwnerId: null,  fields: [],  selectCallback: Ext.emptyFn,  directController: null,    initComponent: function (constructorConfig) {    this.actSelectAndClose = new Ext.Action({  text:RIA.locale.selectItemsWindow.actApplyAndClose.text,  iconCls:'icon\_tick',  handler:this.doSelectAndClose,  scope:this  });    this.actCancelAndClose = new Ext.Action({  text:RIA.locale.selectItemsWindow.actCancelAndClose.text,  iconCls:'icon\_cross',  handler:this.doCancelAndClose,  scope:this  });    var gridCfg = this.createSelectorGrid();    if (this.multiSelect === false){  gridCfg.listeners = {  rowSelected: function(){  this.doSelectAndClose();  },  scope: this  }  }    var defaultConfig = {  title: this.config ? this.config.title : this.title,    items: gridCfg,  buttons: [  this.actSelectAndClose,  this.actCancelAndClose  ]  };    Ext.apply(this, defaultConfig);  Ext.apply(this, constructorConfig);    RIA.TableSelectorWindow.superclass.initComponent.call(this);  },    createSelectorGrid: function(){  return {  xtype: 'RIA.TableSelectorGrid',  border: false,  readOnlyMode: this.readOnlyMode,  ref: 'itemsGrid',  multiSelect: this.multiSelect,  columnsCfg: this.columnsCfg,  parameterId: this.parameterId,  parameterStoreOwnerId: this.parameterStoreOwnerId,  selectedValues: this.selectedValues,  directController: this.directController,  dependencies: this.dependencies,  pagingParams: {  dependencies: this.dependencies  }  }  },    getSelectedItems: function(){  return this.itemsGrid.multiSelect  ? this.itemsGrid.selectedValues  : Ext.pluck(this.itemsGrid.selModel.getSelections(), 'json');  },    doSelectAndClose: function(){  var selectedItems = this.getSelectedItems();  var allLoadedRecords = Ext.pluck(this.itemsGrid.store.getRange(), 'data');    if (this.fireEvent('valueselected', selectedItems, allLoadedRecords)) {  this.close();  }  },    doCancelAndClose: function () {  this.close();  }  }); |

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorWindow = Ext.extend(RIA.TableSelectorWindow, {  initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorWindow.superclass.initComponent.call(this);  },    createSelectorGrid: function(){  return {  xtype: 'CommonAPI.UserParameters.DepartmentSelectorGrid',  border: false,  readOnlyMode: this.readOnlyMode,  ref: 'itemsGrid',  columnsCfg: this.config.columnsCfg,  multiSelect: this.config.multiSelect,  selectedValues: this.config.selectedValues,  directController: this.config.directController,  baseParams: this.config.baseParams || {},  listeners: {  rowSelected: function(){  this.doSelectAndClose();  },  scope: this  }  }  }  }); |

* 1. реализуйте класс грида. Наследуйте от RIA.TableSelectorGrid и переопределите методы «recordEquals» и «getStoreFields»:

|  |
| --- |
| Ext.ns('RIA');    RIA.TableSelectorGrid = Ext.extend(RIA.Common.BaseGridPanel, {  multiSelect: true,  loadMask: { msg: RIA.locale.forms.userParameters.tableSelector.tableSelectorGrid.loadMask },  pageSize: 20,  localSelectedRecords: [],  directController: null,    initComponent: function () {  var config = {  pagingParams: this.baseParams || this.createParams()  };    RIA.TableSelectorGrid.superclass.initComponent.call(this, config);    this.store.on('load', this.afterStoreLoad, this);    if (this.multiSelect === false){  this.addEvents('rowSelected');  this.on('rowdblclick', this.rowDoubleClicked, this);  }  },    createParams: function () {  return {  ParameterId: this.parameterId,  ParameterStoreOwnerId: this.parameterStoreOwnerId,  dependencies: this.dependencies  }  },    getSelModel: function() {  return this.multiSelect  ? this.getMultiSelectSelModel()  : new Ext.grid.RowSelectionModel({ singleSelect: true });  },    getMultiSelectSelModel: function(){  var recordEquals = this.recordEquals;  return new Ext.grid.CheckboxSelectionModel({  sortable: true,  moveEditorOnEnter: true,  listeners: {  beforerowselect: function (sm, rowIndex, keepExisting, record) {  return keepExisting;  },  rowselect: function(selModel, rowIndex, record){    var selected = this.selectedValues.find(function(rec){  return recordEquals(rec, record.json)  });    if (!selected){  this.selectedValues.push(record.json);  }  },  rowdeselect: function(selModel, rowIndex, record){  for(var i = 0; i < this.selectedValues.length; i++){    var selectedRecord = this.selectedValues[i];  if (recordEquals(selectedRecord, record.json)){  this.selectedValues.splice(i, 1);  }  }  },  scope: this  }  });  },    recordEquals: function(rec1, rec2){  return !Ext.isEmpty(rec1.ServerId) && rec1.ServerId === rec2.ServerId;  },    getStoreFields: function() {  var fields = [  { name: 'ServerId' },  { name: 'Text' }  ];    Ext.each(this.columnsCfg, function (column) {  fields.push({  name: column.modelDataIndex,  storedDataIndex: column.storedDataIndex  });  });  return fields;  },    getStoreApi: function () {  return {  read: this.directController.GetList  };  },    getColumns: function() {  var columns = [];    Ext.each(this.columnsCfg, function (column) {  columns.push({  header: column.header,  dataIndex: column.modelDataIndex,  hidden: column.hidden,  sortable: true,  width: column.width,  filter: { type: 'string' },  xtype : column.xtype,  filterable: true  })  });    return columns;  },    getFilterData: function () {  var filters = [], i, len;  var fields = this.store.fields;  this.filters.each(function (f) {  if (f.active) {  var field = fields.get(f.dataIndex);  var dataIndex = field && field.storedDataIndex  ? field.storedDataIndex  : f.dataIndex;  var d = [].concat(f.serialize());  for (i = 0, len = d.length; i < len; i++) {  filters.push({  field: dataIndex,  data: d[i]  });  }  }  });  return filters;  },    addSelModelToColModel: function(){  return this.multiSelect;  },    rowDoubleClicked: function(){  this.fireEvent('rowSelected');  },    afterStoreLoad: function () {  var rowsIdx = [];  var recordEquals = this.recordEquals;    Ext.each(this.selectedValues, function (selectedRec) {    var index = 0;  this.store.each(function(storedRec){  if (recordEquals(storedRec.data, selectedRec)){  rowsIdx.push(index);  }    index++;  });  }, this);    if (rowsIdx.length > 0){  var sm = this.getSelectionModel();    if (this.multiSelect){  sm.selectRows(rowsIdx);  }  else{  sm.selectRow(rowsIdx[0]);  }  }  }  });    Ext.reg('RIA.TableSelectorGrid', RIA.TableSelectorGrid); |

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelectorGrid = Ext.extend(RIA.TableSelectorGrid, {  loadMask: { msg: 'Загрузка записей...' },    initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorGrid.superclass.initComponent.call(this);  },    recordEquals: function(rec1, rec2){  return !Ext.isEmpty(rec1.id) && rec1.id === rec2.id;  },    getStoreFields: function() {  var fields = [];    Ext.each(this.columnsCfg, function (column) {  fields.push({  name: column.modelDataIndex,  storedDataIndex: column.storedDataIndex  });  });  return fields;  }  });    Ext.reg('CommonAPI.UserParameters.DepartmentSelectorGrid', CommonAPI.UserParameters.DepartmentSelectorGrid); |

* 1. реализуйте класс редактора. Наследуйте базовый редактор RIA.UserParameters.SelectorBase, где
  + rendererField – отображаемое свойство модели;
  + recordValueType – тип записи со свойствами fields (список полей модели) и constructor (ссылка на класс значения);
  + choiceWindow – ссылка на класс окна;
  + resultValueType – строка, указывающая на тип, в который необходимо конвертировать значение, полученное из данного редактора на сервере.

Подпишитесь на событие beforechoose, в котором заполните «baseParams» и свойства окна:

|  |
| --- |
| Ext.ns('RIA.UserParameters');    // Поле выбора из представления – таблицы, дерева и т.д.  RIA.UserParameters.SelectorBase = Ext.extend(RIA.BaseObjectEditor, {  editable: false,  maxLength: 6000,  enforceMaxLength: 6000,  width: 190,    initComponent: function () {  // событие перед открытием окна выбора  // обработчик должен заполнить baseParams  this.addEvents('beforechoose');    RIA.UserParameters.SelectorBase.superclass.initComponent.call(this);  },    valueRenderer: function (value) {  if (this.allowMultiSelect) {    if (!Ext.isEmpty(value)) {  var length = value.length;  return String.format(RIA.locale.selectorBase.selectedElementsCount, length);  }  return RIA.locale.selectorBase.emptySelection;  }  else{  return this.elementRenderer(value);  }  },    setValue: function (v) {  this.value = [];  var selectedItems = [];    if (v && v['$values']) {  v = v['$values'];  }    Ext.each(v, function (recordObj) {  var record = new this.recordValueType.constructor();    if (this.isNotObject(record)) {  throw new Error('Тип возвращаемого значения не является объектом.');  }    Ext.each(this.recordValueType.fields, function (attr) {  record[attr] = recordObj[attr]  });  selectedItems.push(record);  }, this);    this.selectedValues = selectedItems;  this.value = selectedItems;    if (this.rendered) {  this.el.dom.value = this.valueRenderer(v);  this.validate();  }    return this;  },    isNotObject: function (obj) {  return !(obj instanceof Object)  },    getValue: function () {  if (!this.rendered) {  return this.prepareValue(this.value);  }    return this.prepareValue(this.value);  },    prepareValue: function (value) {  var rendererField = this.rendererField;  return {  '$type': this.resultValueType,  '$values': value || [],  toString: function () {  var str = '';    Ext.each(this['$values'], function (value, i) {  str += (i > 0 ? '|' : '') + (value[rendererField] || '');  });    return str;  }  };  },    isValueEmpty: function(value){  if (!value || !value['$values']){  return false;  }    if (Ext.isArray(value['$values'])){  return value['$values'].length === 0;  }    return false;  },    onTriggerClick: function () {  var localConfig = {  title: this.title,  directController: this.directController,  baseParams: this.baseParams || {}  };    if (!this.fireEvent('beforechoose', localConfig, this)) {  return;  }    this.beforeShowWindow();    Ext.each(this.value, function(v) {  v.RecordId = this.recordId;  }, this);    var modifiedCount = 0;  var modifiedColumnCodes = [];    if (this.gridEditor && this.gridEditor.record && this.gridEditor.record.modified){  Ext.iterate(this.gridEditor.record.modified, function (key) {  modifiedCount++;  modifiedColumnCodes.push(key);  }, this);    if (modifiedCount === 1){  var modifiedColumnKey = modifiedColumnCodes[0];  if (modifiedColumnKey === localConfig.baseParams.Column){  modifiedCount = 0;  }  }  }    var tableStore = modifiedCount > 0 ? this.gridEditor.record.store : null;    var choiceWnd = new this.choiceWindow({  config: localConfig,  tableStore: tableStore,  initialValue: this.value,  listeners: {  'valueselected': this.onValueSelected,  'close': function () {  if (this.component) {  this.component.modalSelectorWindowOpened = false;  this.fireEvent('blur');  }  },  scope: this  }  });    choiceWnd.show(this);  },    elementRenderer: function (value) {  var rendererField = this.rendererField;    if (value && value['$values']) {  value = value['$values'];  }    if (Ext.isArray(value)) {  if (value.length === 0){  return RIA.locale.selectorBase.emptySelection;  }  return Ext.pluck(value, rendererField).join(' | ');  }    return (value && value[rendererField]  ? value[rendererField]  : RIA.locale.selectorBase.emptySelection);  }  });  Ext.reg('RIA.UserParameters.SelectorBase', RIA.UserParameters.SelectorBase); |

|  |
| --- |
| Ext.ns('CommonAPI.UserParameters');    CommonAPI.UserParameters.DepartmentSelector = Ext.extend(RIA.UserParameters.SelectorBase, {    rendererField: 'name',  recordValueType: {  constructor: CommonAPI.UserParameters.DepartmentSelectorValue,  fields:['id','name', 'code']  },  choiceWindow: CommonAPI.UserParameters.DepartmentSelectorWindow,  resultValueType: 'System.Collections.Generic.List`1[[CommonAPI.UserParams.DepartmentSelector.DepartmentValueDto, CommonAPI]], mscorlib',    initComponent: function () {  CommonAPI.UserParameters.DepartmentSelectorr.superclass.initComponent.call(this);  this.on('beforechoose', this.beforeShowEditor, this);  },    beforeShowEditor: function (config) {  var selectedItems = [];  Ext.each(this.value, function(rec){  selectedItems.push(rec);  });    config.baseParams.parameterId = this.parameterId;  config.baseParams.ParameterStoreOwnerId = this.parameterStoreOwnerId;    if (this.dependencies && this.dependencies.length > 0){  config.baseParams.dependencies = this.dependencies;  }  else{  config.baseParams.UseAllDepartments = this.useAllDepartments;  }    config.title = this.title;  config.columnsCfg = this.columns;  config.multiSelect = this.allowMultiSelect;  config.selectedValues = selectedItems;  config.directController = DepartmentSelector;    return true;  }  });  Ext.reg('CommonAPI.UserParameters.DepartmentSelector', CommonAPI.UserParameters.DepartmentSelector); |

#### Полный код DepartmentSelector

### Соответствие интерфейсов пользовательских параметров

Таблица 14 – Соответствие интерфейсов пользовательских параметров

| Название интерфейса | Описание параметра | Ссылка на описание параметра |
| --- | --- | --- |
| ITableSelector<TRecordType> | Параметр выбора из таблицы, где TRecordType -  тип записи таблицы наследующий BaseTableRecord | См. п. 7.1.4 |
| IReportPeriodSelector | Параметр выбора отчетных периодов | См. п. 7.1.6 |
| IReportPeriodComponentSelector | Параметр выбора компонентов отчетного периода | См. п. 7.1.5 |
| IChainElementSelector | Параметр выбора элементов цепочки | См. п. 7.1.8 |
| IChainElementFromComponentSelector | Параметр выбора элементов цепочки из компонентов отчетных периодов | См. п. 7.1.9 |
| IRealDictionarySelector | Параметр выбора из физического справочника | См. п. 7.1.3.1 |
| IVirtualDictionarySelector | Параметр выбора из виртуального справочника | См. п. 7.1.3.2 |
| ILazyDictionarySelector | Параметр выбора из справочника с ленивой загрузкой записей | См. п. 7.1.3.3 |

### Соответствие пользовательского параметра и элемента списка его значений

Таблица 15 – Соответствие пользовательского параметра и элемента списка его значений

| Тип параметра | Тип элемента списка значений |
| --- | --- |
| ITableSelector<TRecordType> | TRecordType |
| IReportPeriodSelector | IОтчетныйПериод |
| IReportPeriodComponentSelector | IКомпонентОтчетногоПериода |
| IChainElementSelector | IЭлементЦепочки |
| IChainElementFromComponentSelector | IКомпонентОтчетногоПериода |
| IRealDictionarySelector | IЗаписьСправочника |
| IVirtualDictionarySelector | IЗаписьСправочника |
| ILazyDictionarySelector | IЗаписьСправочника |

### Зависимость параметров

#### Реализация зависимостей между пользовательскими параметрами

По условиям задачи пользовательский параметр может зависеть максимум от 4-х других параметров, поэтому в платформе реализовано 4 абстрактных класса:

* public abstract class DependentParam<TD1, TItem> – класс для реализации зависимости от одного параметра
* public abstract class DependentParam<TD1, TD2, TItem> – класс для реализации зависимости от двух параметров
* public abstract class DependentParam<TD1, TD2, TD3, TItem> – класс для реализации зависимости от трёх параметров
* public abstract class DependentParam<TD1, TD2, TD3, TD4, TItem> – класс для реализации зависимости от четырёх параметров
* TD1, TD2, TD3, TD4 – типы значений, возвращаемые параметрами от которых зависит текущий параметр
* TItem – тип элемента списка значений текущего параметра

Ниже рассмотрен пример создания зависимостей между параметрами:

1. добавьте параметр выбора отчетных периодов (см. п. 7.1.2 настоящей инструкции):

|  |
| --- |
| var parametersList = new СписокПользовательскихПараметров();  parametersList.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect")  .WithTitle("Выбор периодов");  }); |

1. создайте класс, реализующий зависимость параметра выбора компонентов отчетного периода в зависимости от параметра выбора отчетного периода:

|  |
| --- |
| // Параметр зависит от параметра выбора отчетных периодов -  // поэтому в качестве TD1 здесь будет List<ReportPeriodValueDto>  // и т.к. текущий параметр является параметром выбора компонентов отчетного периода -  // в качестве TItem будет IКомпонентОтчетногоПериода  public class ComponentsSelector : DependentParam<List<ReportPeriodValueDto>, IКомпонентОтчетногоПериода>  {  public override IEnumerable<IКомпонентОтчетногоПериода> GetDataSourceInner(List<ReportPeriodValueDto> dependValue1)  {  if (dependValue1 == null || !dependValue1.Any())  {  return new List<IКомпонентОтчетногоПериода>();  }    var opDto = dependValue1.First();    var op = Выборка<ОтчетныйПериод>.НайтиОбъектПоРеквизиту("Идентификатор", opDto.Id);    var components = new Выборка<КомпонентОтчетногоПериода>();  components.Запрос.ДобавитьУсловиеОтбора("ОтчетныйПериод", op);  components.ЗагрузитьПолностью();    return components;  }  } |

1. добавьте зависимый параметр выбора компонентов отчетных периодов:

|  |
| --- |
| parametersList.BuildParameter<IReportPeriodComponentSelector>(  parameter =>  {  parameter.WithId("copSelector")  .WithTitle("Выбор компонентов")  .ToDependent<ComponentsSelector>("opSelect"); // делаем параметр зависимым от параметра добавленного на первом шаге  }); |

1. создайте класс, реализующий зависимость параметра выбора элементов цепочки. Этот параметр в свою очередь будет зависеть от параметров выбора отчетного периода и выбора компонентов:

|  |
| --- |
| // Параметр зависит от параметра выбора отчетных периодов и параметра выбора компонентов -  // поэтому в качестве TD1 здесь будет List<ReportPeriodValueDto>, а в качестве TD2 - List<ReportPeriodComponentValueDto>  // и т.к. текущий параметр является параметром выбора элементов цепочки -  // в качестве TItem будет IЭлементЦепочки  public class ChainElementSelector : DependentParam<List<ReportPeriodValueDto>, List<ReportPeriodComponentValueDto>, IЭлементЦепочки>  {  public override IEnumerable<IЭлементЦепочки> GetDataSourceInner(List<ReportPeriodValueDto> dependValue1, List<ReportPeriodComponentValueDto> dependValue2)  {  if (dependValue2 == null || !dependValue2.Any())  {  return new List<IЭлементЦепочки>();  }    var component = dependValue2.First();    var chain = Выборка<ЦепочкаСдачиОтчетности>.НайтиОбъектПоРеквизиту("Код", component.ChainCode);    var elements = new Выборка<ЭлементЦепочки>();  elements.Запрос.ДобавитьУсловиеОтбора("ЦепочкаСдачи", chain);  elements.ЗагрузитьПолностью();    return elements;  }  } |

1. добавьте зависимый параметр выбора элементов цепочки:

|  |
| --- |
| parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  // порядок вызова методов билдера не важен  parameter.ToDependent<ChainElementSelector>("opSelect", "copSelector") // делаем параметр зависимым от параметров добавленных на 1 и 3 шагах  .WithId("elementsSelector")  .WithTitle("Выбор элементов");  }); |

### UML

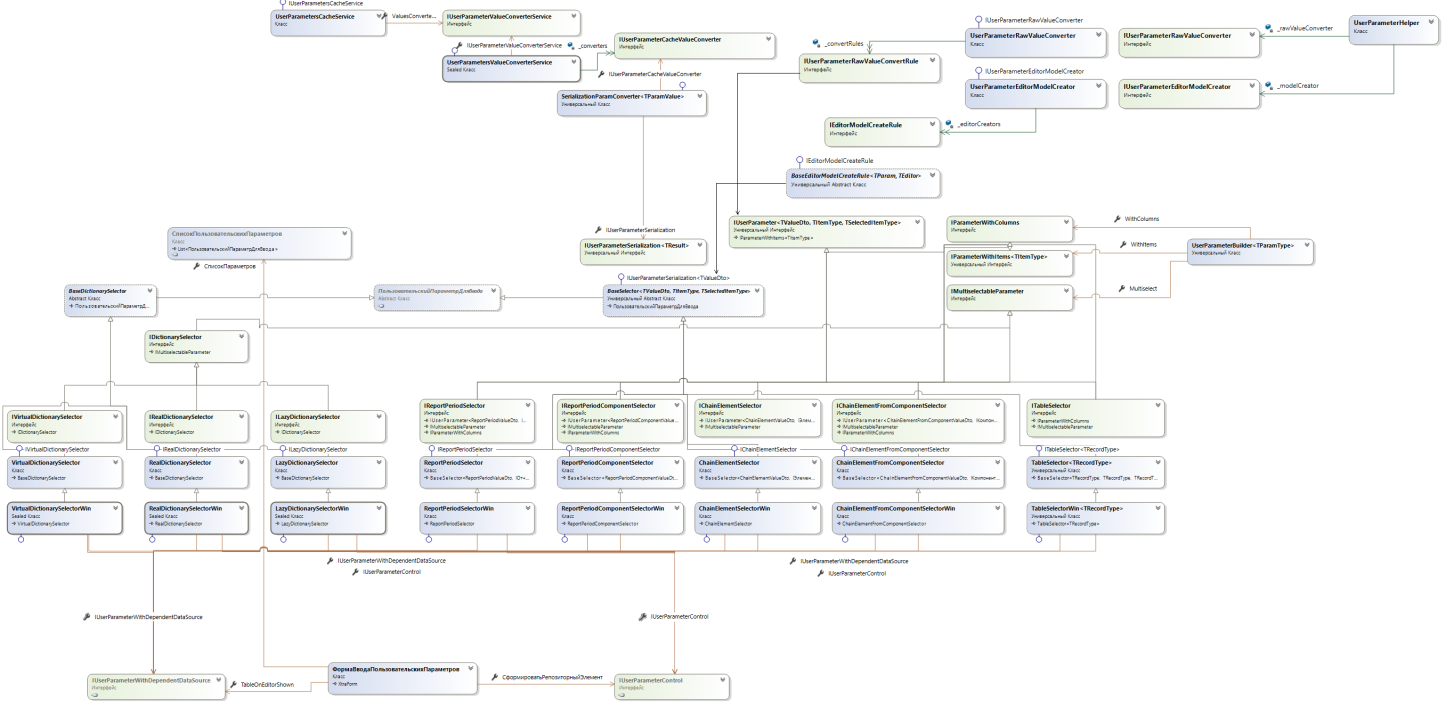


Рисунок 10 – UML

## Старые параметры

### Формирование списка пользовательских параметров

Для формирования списка пользовательских параметров переопределите метод СформироватьСписокПользовательскихПараметров():

|  |
| --- |
| protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string операция)  {  var параметры = new СписокПользовательскихПараметров();    //Формирование различных пользовательских параметров в зависимости от "операция"  switch (операция)  {  //В случае необходимости формирования пользовательских параметров перед выполнением обработки  case "НаименованиеМетодаОбработки":  параметры.ДобавитьСтроковыйПараметр("ИдентификаторСтроковогоПараметра", "Наименование строкового параметра");  break;    //В случае необходимости формирования пользовательских параметров перед проверкой межформенных увязок  case "ОперацияПроверкиМежформенныхУвязок":  var списокЗначений = new List<string>();  параметры.ДобавитьПараметрВыборСтроковогоЗначенияИзСписка("ИдентификаторПараметрВыборСтроковогоЗначенияИзСписка", "Наименование параметра", списокЗначений);  break;    //В случае необходимости формирования пользовательских параметров перед проверкой внутриформенных увязок  case "ОперацияПроверкиВнутреннихУвязок":  параметры.ДобавитьПараметрФлажок("ИдентификаторФлага", "Наименование параметра");  break;  }    return параметры;  }    //Обработка для примера  [АтрибутПунктаОбработки("Отображаемое в форме наименование обработки")]  public void НаименованиеМетодаОбработки()  {  //логика обработки  } |

### Получение значений пользовательских параметров

|  |
| --- |
| //Метод ПолучитьЗначениеПараметра() возвращает объект типа object, поэтому для получения значения требуется явное приведение типа  var выбранныйСтроковыйПараметр = СписокПараметровОперации.ПолучитьЗначениеПараметра("ИдентификаторСтроковогоПараметра").ToString();  var выбранныйЛогическийПараметр = (bool)СписокПараметровОперации.ПолучитьЗначениеПараметра("ИдентификаторФлага"); |

### Список добавляемых пользовательских параметров

Таблица 16 – Список параметров

| Наименование | Метод | |
| --- | --- | --- |
| Числовой параметр | ДобавитьЧисловойПараметр(string идентификатор, string заголовок); | |
| ДобавитьЧисловойПараметр(string идентификатор, string заголовок, decimal значениеПоУмолчанию); | |
| ДобавитьЧисловойПараметр(string идентификатор, string заголовок, decimal значениеПоУмолчанию, int точность); | |
| Целочисленный параметр | ДобавитьЦелочисленныйПараметр(string идентификатор, string заголовок, int значениеПоУмолчанию, int минимальноеЗначение, int максимальноеЗначение); | |
| ДобавитьЦелочисленныйПараметр(string идентификатор, string заголовок); | |
| Неотрицательный целочисленный параметр | ДобавитьНеотрицательныйЦелочисленныйПараметр(string идентификатор, string заголовок, int максимальноеЗначение) | |
| ДобавитьНеотрицательныйЦелочисленныйПараметр(string идентификатор, string заголовок); | |
| Положительный целочисленный параметр | ДобавитьПоложительныйЦелочисленныйПараметр(string идентификатор, string заголовок, int максимальноеЗначение) | |
| ДобавитьПоложительныйЦелочисленныйПараметр(string идентификатор, string заголовок) | |
| Параметр дата время | ДобавитьПараметрДатаВремя(string идентификатор, string заголовок) | |
| ДобавитьПараметрДатаВремя(string идентификатор, string заголовок, DateTime значениеПоУмолчанию) | |
| Строковый параметр | ДобавитьСтроковыйПараметр(string идентификатор, string заголовок) | |
| ДобавитьСтроковыйПараметр(string идентификатор, string заголовок, string значениеПоУмолчанию) | |
| Параметр флажок | ДобавитьПараметрФлажок(string идентификатор, string заголовок) | |
| ДобавитьПараметрФлажок(string идентификатор, string заголовок, bool значениеПоУмолчанию) | |
| Параметр редактор с кнопкой | ДобавитьПараметрРедакторСКнопкой(string идентификатор, string заголовок, string значениеПоУмолчанию) | |
| Параметр выбора файла | ДобавитьПараметрВыбораФайла(string идентификатор, string заголовок) | |
| ДобавитьПараметрВыбораФайла(string идентификатор, string заголовок, string значениеПоУмолчанию) | |
| Параметр выбора каталога  (только WIN) | ДобавитьПараметрВыбораКаталога(string идентификатор, string заголовок) | |
| ДобавитьПараметрВыбораКаталога(string идентификатор, string заголовок, string значениеПоУмолчанию) | |
| Параметр выбор строкового значения из списка | ДобавитьПараметрВыборСтроковогоЗначенияИзСписка(string идентификатор, string заголовок, IEnumerable<string> списокЗначений) | |
| ДобавитьПараметрВыборСтроковогоЗначенияИзСписка(string идентификатор, string заголовок, IEnumerable<string> списокЗначений, string значениеПоУмолчанию) | |
| Параметр выбор объекта из списка | ДобавитьПараметрВыборОбъектаИзСписка(string идентификатор, string заголовок, object списокЗначений, string светимоеПоле, object значениеПоУмолчанию) | |
| ДобавитьПараметрВыборОбъектаИзСписка(string идентификатор, string заголовок, object списокЗначений, IEnumerable<string> светимыеПоля, object значениеПоУмолчанию) | |
| Параметр выбор из справочника | IRealDictionarySelector - Интерфейс пользовательского параметра выбора из физического справочника | |
| Свойства | string DictionaryCode //Полный код справочника |
| DateTime DictionaryActualDate //Дата актуальности справочника |
| IVirtualDictionarySelector - Интерфейс пользовательского параметра выбора из виртуального справочника | |
| Свойства | IEnumerable<IЗаписьСправочника> Records //Список записей справочника |
| ILazyDictionarySelector - Интерфейс пользовательского параметра выбора из справочника с ленивой подгрузкой записей из макроса формы | |
| Свойства | IEnumerable<DictionaryRecordNode> RootNodes //Корневые узлы дерева |
| string LoaderAssemblyName //Полное имя сборки, из которой необходимо загрузить делегат загрузки записей |
| Параметр выбор из таблицы | ITableSelector - Интерфейс пользовательского параметра выбора из таблицы | |
| Методы, которые необходимо определить | IEnumerable<object> GetItems() //Получить записи таблицы |
| IEnumerable<object> GetSelectedItems() //Получить выбранные записи таблицы |
| IEnumerable<object> GetValueForMacro(object paramDataStore, object rawClientValue) //Получить значение параметра в зависимости от переданного с клиента значения, где paramDataStore - источник данных параметра, rawClientValue - значение, переданное с клиента |
| Параметр редактор фильтра (WIN) | ДобавитьПараметрРедакторФильтра(string идентификатор, string заголовок, IEnumerable<СтолбецПараметраФильтр> столбцыФильтра, string значениеПоУмолчанию) | |
| Параметр выбор из дерева (WIN) | ДобавитьПараметрВыборИзДерева( string идентификатор, string заголовок, object источникДанных, string ключевоеПоле, string ссылкаНаРодителя, IEnumerable<СтолбецПараметраДерево> столбцыДерева, object значениеПоУмолчанию) | |

## Зависимость параметров

По условиям задачи пользовательский параметр может зависеть максимум от 4-х других параметров, поэтому в платформе реализовано 4 абстрактных класса:

* public abstract class DependentParam<TD1, TItem> – класс для реализации зависимости от одного параметра;
* public abstract class DependentParam<TD1, TD2, TItem> – класс для реализации зависимости от двух параметров;
* public abstract class DependentParam<TD1, TD2, TD3, TItem> – класс для реализации зависимости от трёх параметров;
* public abstract class DependentParam<TD1, TD2, TD3, TD4, TItem> – класс для реализации зависимости от четырёх параметров.

TD1, TD2, TD3, TD4 – типы значений параметров, от которых зависит текущий параметр TItem.

TItem – тип элемента списка значений текущего параметра. См. п. 7.1.12 настоящей инструкции.

Рассмотрим пример создания зависимостей между параметрами:

* добавьте параметр выбора отчетных периодов (см. п. 7.1.2):

|  |
| --- |
| var parametersList = new СписокПользовательскихПараметров();  parametersList.BuildParameter<IReportPeriodSelector>(  parameter =>  {  parameter.WithId("opSelect")  .WithTitle("Выбор периодов");  }); |

* создайте класс, реализующий зависимость параметра выбора компонентов отчетного периода от параметра выбора отчетного периода:

|  |
| --- |
| // Параметр зависит от параметра выбора отчетных периодов -  // поэтому в качестве TD1 здесь будет List<ReportPeriodValueDto>  // и т.к. текущий параметр является параметром выбора компонентов отчетного периода -  // в качестве TItem будет IКомпонентОтчетногоПериода  public class ComponentsSelector : DependentParam<List<ReportPeriodValueDto>, IКомпонентОтчетногоПериода>  {  public override IEnumerable<IКомпонентОтчетногоПериода> GetDataSourceInner(List<ReportPeriodValueDto> dependValue1)  {  if (dependValue1 == null || !dependValue1.Any())  {  return new List<IКомпонентОтчетногоПериода>();  }    var opDto = dependValue1.First();    var op = Выборка<ОтчетныйПериод>.НайтиОбъектПоРеквизиту("Идентификатор", opDto.Id);    var components = new Выборка<КомпонентОтчетногоПериода>();  components.Запрос.ДобавитьУсловиеОтбора("ОтчетныйПериод", op);  components.ЗагрузитьПолностью();    return components;  }  } |

* добавьте зависимый параметр выбора компонентов отчетных периодов:

|  |
| --- |
| parametersList.BuildParameter<IReportPeriodComponentSelector>(  parameter =>  {  parameter.WithId("copSelector")  .WithTitle("Выбор компонентов")  .ToDependent<ComponentsSelector>("opSelect"); // делаем параметр зависимым от параметра добавленного на первом шаге  }); |

* создайте класс, реализующий зависимость параметра выбора элементов цепочки от параметров выбора отчетного периода и выбора компонентов:

|  |
| --- |
| // Параметр зависит от параметра выбора отчетных периодов и параметра выбора компонентов -  // поэтому в качестве TD1 здесь будет List<ReportPeriodValueDto>, а в качестве TD2 - List<ReportPeriodComponentValueDto>  // и т.к. текущий параметр является параметром выбора элементов цепочки -  // в качестве TItem будет IЭлементЦепочки  public class ChainElementSelector : DependentParam<List<ReportPeriodValueDto>, List<ReportPeriodComponentValueDto>, IЭлементЦепочки>  {  public override IEnumerable<IЭлементЦепочки> GetDataSourceInner(List<ReportPeriodValueDto> dependValue1, List<ReportPeriodComponentValueDto> dependValue2)  {  if (dependValue2 == null || !dependValue2.Any())  {  return new List<IЭлементЦепочки>();  }    var component = dependValue2.First();    var chain = Выборка<ЦепочкаСдачиОтчетности>.НайтиОбъектПоРеквизиту("Код", component.ChainCode);    var elements = new Выборка<ЭлементЦепочки>();  elements.Запрос.ДобавитьУсловиеОтбора("ЦепочкаСдачи", chain);  elements.ЗагрузитьПолностью();    return elements;  }  } |

* добавьте зависимый параметр выбора элементов цепочки:

|  |
| --- |
| parametersList.BuildParameter<IChainElementSelector>(  parameter =>  {  // порядок вызова методов билдера не важен  parameter.ToDependent<ChainElementSelector>("opSelect", "copSelector") // делаем параметр зависимым от параметров добавленных на 1 и 3 шагах  .WithId("elementsSelector")  .WithTitle("Выбор элементов");  }); |

# Работа с хранимыми блокировками

Блокировка формы необходима, если нужно изменить данные внешней формы.

Блокировка формы/снятие блокировки через отчетную форму данных:

|  |
| --- |
| ОтчетнаяФормаДанных отчетнаяФорма = ДополнительныеМетодыПолученияДанных.ПолучитьОтчетнуюФорму("КодФормы",  Форма.Идентификатор.Учреждение, Форма.Идентификатор.КомпонентОтчетногоПериода, Форма.Идентификатор.ЭлементЦепочки);    if (отчетнаяФорма.ЗаблокироватьФорму())  {  try  {  //что-то делаем  отчетнаяФорма.СохранитьДанные();  }  finally  {  отчетнаяФорма.СнятьБлокировкуФормы();  }  } |

Таблица 17 – Описание методов

| Название метода | Тип | Описание |
| --- | --- | --- |
| ЗаблокироватьФорму() | bool | Блокирует данные текущей формы. Возвращает true, если блокировка прошла успешно; в противном случае - false. |
| СнятьБлокировкуФормы() | bool | Убирает блокировку текущего пользователя с формы. Возвращает true, если блокировка снята успешно; в противном случае - false. |
| СнятьБлокировкуФормы(string userLogin, string ipAddress) | bool | Убирает блокировку указанного пользователя с формы. Возвращает true, если блокировка снята успешно; в противном случае - false. |

Заблокировка формы/снятие блокировки через «IformBlockingManager»:

|  |
| --- |
| IFormBlockingManager formBlockingManager = КонтейнерЗависимостей.Текущий.Resolve<IFormBlockingManager>();  if(formBlockingManager.TryLockForWrite(отчетнаяФорма.ИдентификаторДанныхФормы) == EBlockResult.Успешно)  {  try  {  //что-то делаем  }  finally  {  formBlockingManager.TryUnlock(отчетнаяФорма.ИдентификаторДанныхФормы);  }  } |

| Название метода | Тип | Описание |
| --- | --- | --- |
| TryLockForWrite(ИдентификаторДанныхФормы formDataId) | EBlockResult | Попытка заблокировать отчетную форму на редактирование |
| TryLockForWriteExtended(ИдентификаторДанныхФормы formDataId) | BlockInfoData | Попытка заблокировать отчетную форму на редактирование |
| TryUnlock(ИдентификаторДанныхФормы formDataId) | bool | Попытка разблокировать форму от текущего пользователя |
| TryUnlock(ИдентификаторДанныхФормы formDataId, string userLogin, string ipAddress) | bool | Попытка разблокировать форму от указанного пользователя |
| IsUserIsBlockOwner(string login, string userIp, ИдентификаторДанныхФормы formDataId) | BlockInfoData | Проверка блокировки формы на принадлежность пользователю |
| IsUserIsBlockOwner(Guid userId, string userIp, ИдентификаторДанныхФормы formDataId) | BlockInfoData | Проверка блокировки формы на принадлежность пользователю |
| GetBlockedFormIdentifiers(List<ИдентификаторДанныхФормы> formIdentifiers) | List<ИдентификаторДанныхФормы> | Получить список идентификаторов заблокированных форм |

В классе BlockInfoData передается информация о блокировке. Свойства класса BlockInfoData:

* string BlockOwner – владелец блокировки (информация о пользователе);
* string BlockOwnerName – наименование пользователя-владельца блокировки;
* EBlockResult BlockResult – результат блокировки;
* DateTime DateEdit – дата блокировки.

Перечислением EBlockResult указывается результат блокировки. Возможные значения: НоваяБлокировка, УжеЗаблокировано, ЗаблокированоДругимПользователем, МенеджерБдНеИнициализирован, ДругаяОшибка, Успешно.

# Асинхронные обработки и управление прогресс баром в ПФ

## Пример реализации асинхронной операции в прикладном коде.

Обработки в форме (начиная с версии 4.1) поддерживают следующие сигнатуры:

public void TestHandler1(MacrosWorkerContext ctx)

public void TestHandler2()

public async Task<object> TestHandler11(MacrosWorkerContext ctx)

public async Task<object> TestHandler22()

Описание поведения каждой сигнатуры в Web и Win приведено в таблице ниже (Таблица 18).

Таблица 18 – Описание поведения сигнатур в Web и Win

| Сигнатура | Поведение Web | Поведение Win |
| --- | --- | --- |
| public void TestHandler1(MacrosWorkerContext ctx) | Синхронное выполнение без прогресс бара. | Синхронное выполнение без прогресс бара. |
| public void TestHandler2() | Синхронное выполнение без прогресс бара. Легаси вариант. | Синхронное выполнение без прогресс бара. |
| public async Task<object> TestHandler11(MacrosWorkerContext ctx) | Полное асинхронное выполнение с прогресс баром которым можно управлять с помощью объекта MacrosWorkerContext. | Синхронное выполнение без прогресс бара. |
| public async Task<object> TestHandler22() | Полное асинхронное выполнение с фейковым прогресс баром. | Синхронное выполнение без прогресс бара. |

Примеры:

* выполнение с прогресс баром, которым можно управлять с помощью объекта «MacrosWorkerContext»:

|  |
| --- |
| public void TestHandler1(MacrosWorkerContext ctx)  {  var result = new List<string>();  for (var i = 1; i <= 20; i++)  {  result.Add($"Итерация № {i}");  Thread.Sleep(2000);  ctx.UpdateProgress(5,$"Итерация № {i}");  ctx.CancellationToken.ThrowIfCancellationRequested();  }  РезультатВыполненияОперации = Сформировать(result);  } |

* синхронное выполнение без прогресс бара. Легаси вариант:

|  |
| --- |
| public void TestHandler2()  {  var result = new List<string>();  for (var i = 1; i <= 20; i++)  {  result.Add($"Итерация № {i}");  }  РезультатВыполненияОперации = Сформировать(result);  } |

* полное асинхронное выполнение с прогресс баром, которым можно управлять с помощью объекта MacrosWorkerContext:

|  |
| --- |
| public async Task<object> TestHandler11(MacrosWorkerContext ctx)  {  var result = new List<string>();  for (var i = 1; i <= 20; i++)  {  result.Add($"Итерация № {i}");  await Task.Delay(TimeSpan.FromSeconds(2));  ctx.UpdateProgress(5,$"Итерация № {i}");  ctx.CancellationToken.ThrowIfCancellationRequested();  }  РезультатВыполненияОперации = Сформировать(result);    return 1;  } |

* полное асинхронное выполнение с фейковым прогресс баром:

|  |
| --- |
| public async Task<object> TestHandler22()  {  var result = new List<string>();  for (var i = 1; i <= 20; i++)  {  result.Add($"Итерация № {i}");  await Task.Delay(TimeSpan.FromSeconds(2));  }  РезультатВыполненияОперации = Сформировать(result);    return 1;  } |

## Пример управления прогресс баром в макросе Печатной форме

**Примечание** – Необходимо проверять PrintFormWorkerContext на null, т.к. в Win данные параметр не передаётся.

Печатные формы теперь поддерживают следующие сигнатуры:

public void МакросОбработкиПечатнойФормы(ref ПараметрыОтчета параметры, ОтчетнаяФормаДанных отчетнаяФорма)

public void МакросОбработкиПечатнойФормыAsync(ПараметрыОтчета параметры, ОтчетнаяФормаДанных отчетнаяФорма, PrintFormWorkerContext workerContext)

| Сигнатура | Поведение Web | Поведение Win |
| --- | --- | --- |
| public void МакросОбработкиПечатнойФормы(ref ПараметрыОтчета параметры, ОтчетнаяФормаДанных отчетнаяФорма) | Синхронное выполнение без прогресс бара | Поведение не изменилось |
| public void МакросОбработкиПечатнойФормыAsync(ПараметрыОтчета параметры, ОтчетнаяФормаДанных отчетнаяФорма, PrintFormWorkerContext workerContext) | Синхронное выполнение с прогресс баром | Поведение не изменилось |

Примеры:

* cинхронное выполнение с прогресс баром:

|  |
| --- |
| public void МакросОбработкиПечатнойФормыAsync(ПараметрыОтчета параметры, ОтчетнаяФормаДанных отчетнаяФорма, PrintFormWorkerContext workerContext)  {  workerContext.Report(10,"НАЧАЛО");  Секция секция = параметры.СтруктурныеПараметрыОтчета.ДобавитьСекцию("СекцияПользователи");  Секция секцияКоличество = параметры.СтруктурныеПараметрыОтчета.ВыбратьСекцию("СекцияКоличество");  Выборка<ОператорУчреждения> выборкаОператоров = new Выборка<ОператорУчреждения>();  выборкаОператоров.Запрос.ДобавитьУровеньСортировки("DateLastEntry", Запрос.ПорядокСортировки.ПоВозрастанию);  выборкаОператоров.ЗагрузитьПолностью();  foreach (ОператорУчреждения оператор in выборкаОператоров)  {  if (оператор.ПометкаУдаления || оператор.СистемныйПользователь == null || оператор.СистемныйПользователь.ПользовательЗаблокирован)  {  // нет требования  continue;  }  секция.ДобавитьСтроку();  секция["\_1"] = оператор.СистемныйПользователь.Логин;  секция["\_2"] = оператор.СистемныйПользователь.Наименование;  секция["\_3"] = оператор.РабочееУчреждение != null ? оператор.РабочееУчреждение.Наименование : string.Empty;  switch (оператор.ClientTypeWhenLastEntry)  {  case SvodyClientTypeEnum.WebClient:  секция["\_4"] = "Web";  break;  case SvodyClientTypeEnum.WinClient:  секция["\_4"] = "Win";  break;  }  if (оператор.DateLastEntry != default(DateTime))  {  секция["\_5"] = оператор.DateLastEntry.ToString("dd.MM.yyyy HH:mm:ss");  }  }  секцияКоличество.ДобавитьСтроку();  секцияКоличество["\_1"] = выборкаОператоров.Count(x => x.ClientTypeWhenLastEntry.Equals(SvodyClientTypeEnum.WinClient));  секцияКоличество["\_2"] = выборкаОператоров.Count(x => x.ClientTypeWhenLastEntry.Equals(SvodyClientTypeEnum.WebClient));  выборкаОператоров.Очистить();    var kk = 10;    for (var i = 1; i <= 18; i++)  {  Thread.Sleep(2000);  kk = kk+5;  workerContext.Report(kk,$"Итерация № {i}");  workerContext.CancellationToken.ThrowIfCancellationRequested();  }  } |

## Пример файла svody.config с заданным количеством параллельно выполняемых обработок

Пример секции в файле svody.config, в которой можно настроить количество параллельно выполняемых обработок:

|  |
| --- |
| <Bars.MacrosManager>  <ConcurrencyLimit>10</ConcurrencyLimit>  </Bars.MacrosManager> |

# Описание виртуальных методов обработчика с их параметрами

Таблица 19 – Описание виртуальных методов обработчика с их параметрами

| Название | Принимаемые параметры | Возвращаемый тип | Действие |
| --- | --- | --- | --- |
| СформироватьСписокПользовательскихПараметров | * string Операция   наименование операции | * СписокПользовательскихПараметров   Класс для работы с пользовательскими параметрами  См.п. 7.1 настоящей инструкции | Позволяет получить список пользовательских параметров по названию. В базовой реализации возвращает null |
| ОбработатьСобытиеФормированияВнутреннихУвязок | * АргументыФормированияВнутриформенныхУвязок аргументы   содержит в себе список выбранных закладок | * List<ОбъектУвязки>   Список объектов увязок  См. п.15 настоящей инструкции | Вызывается перед запуском проверки внутренних увязок. В базовой реализации вызывает метод *ОбработатьСобытиеФормированияВнутреннихУвязок* без параметров. При переопределении в дочернем классе позволяет добавлять к списку системных увязок новые увязки с помощью макроса |
| ОбработатьСобытиеФормированияВнутреннихУвязок |  | * List<ОбъектУвязки>   Список объектов увязок  См. п.15 настоящей инструкции | Вызывается перед запуском проверки внутренних увязок. В базовой реализации создает новый пустой список объектов увязки и возвращает его. При переопределении в дочернем классе позволяет добавлять к списку системных увязок новые увязки с помощью макроса |
| ОбработатьСобытиеФормированияМежформенныхУвязок |  | * List<ОбъектУвязки>   Список объектов увязок  См. п.15 настоящей инструкции | Вызывается перед запуском проверки межформенных увязок. В базовой реализации создает новый пустой список объектов увязки и возвращает его. При переопределении в дочернем классе позволяет добавлять к списку системных увязок новые увязки с помощью макроса |
| ОбработатьСобытиеПослеУстановкиЗначенияЯчейкиСправочника | * АргументыПослеУстановкиЗначенияЯчейки аргументы   содержит в себе таблицу, код строки и код столбца ячейки, в которой было установлено значение, флаги обновления значения и пересчета автоблоков | void | Вызывается после установки значения ячейки со справочником |
| ОбработатьСобытиеПослеИзмененияЗначенияЯчейки | * АргументыСобытияПослеИзмененияЗначенияЯчейки аргументы   содержит в себе таблицу и код ячейки, в которой было изменено значение | void | Вызывается после изменения данных ячейки таблицы |
| ОбработатьСобытиеОткрытияФормы | * АргументыОткрытияФормы аргументы   содержит в себе режимы открытия и сборки сводной экранной формы | void | Вызывается при открытии формы |
| ОбработатьСобытиеЗакрытияФормы | * АргументыЗакрытияФормы аргументы   содержит в себе флаг с отменой закрытия | * АргументыЗакрытияФормы   содержит в себе флаг с отменой закрытия | Вызывается при закрытии формы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеДоСохраненияФормы | * АргументыСохраненияФормы аргументы   содержит в себе флаг с отменой сохранения и флаг с сохранением сводной формы | * АргументыСохраненияФормы   содержит в себе флаг с отменой сохранения и флаг с сохранением сводной формы | Вызывается перед сохранением формы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеСохраненияФормы | * АргументыСохраненияФормы аргументы   содержит в себе флаг с отменой сохранения и флаг с сохранением сводной формы | * АргументыСохраненияФормы   содержит в себе флаг с отменой сохранения и флаг с сохранением сводной формы | Вызывается после сохранения формы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьНеверноЗаполненныеЯчейкиДоЗакрытияСубтаблицы | * АргументыОбработкиНеВерноЗаполненныхЯчеек аргументы   содержит в себе таблицу данных и флаг - найдены ли неверно заполненные ячейки | * АргументыОбработкиНеВерноЗаполненныхЯчеек   содержит в себе таблицу данных и флаг - найдены ли неверно заполненные ячейки | Вызывается если перед закрытия субтаблицы были обнаружены неверно заполненные ячейки. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеСохраненияСубтаблицы | * АргументыСохраненияСубтаблицы аргументы   содержит в себе идентификатор субтаблицы, данные субтаблицы и флаг отмены сохранения | * АргументыСохраненияСубтаблицы   содержит в себе идентификатор субтаблицы, данные субтаблицы и флаг отмены сохранения | Вызывается перед сохранением субтаблицы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеСохраненияСубтаблицы | * АргументыПослеСохраненияСубтаблицы аргументы   содержит в себе идентификатор субтаблицы и данные субтаблицы | * АргументыПослеСохраненияСубтаблицы   содержит в себе идентификатор субтаблицы и данные субтаблицы | Вызывается после сохранения субтаблицы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеЗакрытияСубтаблицы | * АргументыЗакрытияСубтаблицы аргументы   содержит в себе таблицу данных, флаг отмены закрытия и флаг закрытия с сохранением | * АргументыЗакрытияСубтаблицы   содержит в себе таблицу данных, флаг отмены закрытия и флаг закрытия с сохранением | Вызывается при закрытии субтаблицы. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПередПроверкойУвязок | * АргументыПередПроверкойУвязок аргументы   содержит в себе тип увязок, флаг с отменой операции, и список увязок | * АргументыПередПроверкойУвязок   содержит в себе тип увязок, флаг с отменой операции, и список увязок | Вызывается перед проверкой увязок. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеПроверкиУвязок | * АргументыПослеПроверкиУвязок аргументы   содержит тип увязок, количество проверенных увязок и список ошибок проверки | * АргументыПослеПроверкиУвязок   содержит тип увязок, количество проверенных увязок и список ошибок проверки | Вызывается после проверки увязок. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеЗагрузкиАрхива | * АргументыПослеЗагрузкиАрхива аргументы   содержит в себе список архивов и менеджер истории | void | Вызывается после загрузки архива |
| ОбработатьСобытиеДоПересчетаАвтоблоков | * АргументыПослеУстановкиЗначенияЯчейки аргументы   содержит в себе таблицу, код строки и код столбца ячейки, в которой было установлено значение, флаги обновления значения и пересчета автоблоков | void | Вызывается перед пересчетом автоблоков |
| ОбработатьСобытиеПослеПересчетаАвтоблоков | * АргументыПослеУстановкиЗначенияЯчейки аргументы   содержит в себе таблицу, код строки и код столбца ячейки, в которой было установлено значение, флаги обновления значения и пересчета автоблоков | void | Вызывается после пересчета автоблоков |
| ОбработатьСобытиеПередЗапускомМакроса | * АргументыПередЗапускомМакроса аргументы   содержит в себе имя макроса и флаг с отменой запуска этого макроса | * АргументыПередЗапускомМакроса   содержит в себе имя макроса и флаг с отменой запуска этого макроса | Вызывается перед запуском любого макроса. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеЗапускаМакроса | * АргументыПослеЗапускаМакроса аргументы   содержит в себе имя макроса | void | Вызывается после запуска любого макроса |
| ОбработатьСобытиеПередИмпортомДанных | * АргументыПередИмортомДанных аргументы   содержит наименование  источника импорта данных и флаг отмены импорта данных | * АргументыПередИмортомДанных   содержит наименование источника импорта данных и «флажок» отмены импорта данных | Вызывается перед импортом данных. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеИмпортаДанных | * АргументыПослеИмпортаДанных аргументы   содержит наименование источника импорта данных | void | Вызывается после импорта данных |
| ОбработатьСобытиеПередЭкспортомДанных | * АргументыПередЭкспортомДанных аргументы   содержит наименование источника экспорта данных и флаг отмены экспорта данных | * АргументыПередЭкспортомДанных   содержит наименование источника экспорта данных и «флажок» отмены экспорта данных | Вызывается перед экспортом данных. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеЭкспортаДанных | * АргументыПослеЭкспортаДанных аргументы   содержит наименование источника экспорта данных | void | Вызывается после экспорта данных |
| ОбработатьСобытиеПередВставкойИзБО | * АргументыПередВставкойИзБО аргументы   содержит в себе форму и таблицу, в которую происходит вставка, текст из буфера обмена, массив кодов столбцов и флаг отмены операции вставки данных из буфера обмена | * АргументыПередВставкойИзБО   содержит в себе форму и таблицу, в которую происходит вставка, текст из буфера обмена, массив кодов столбцов и флаг отмены операции вставки данных из буфера обмена | Вызывается перед вставкой из буфера обмена. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПослеВставкиИзБО | * АргументыПослеВставкиИзБО аргументы   содержит в себе таблицу, в которую произошла вставка, результат сверки данных и флаг обработки события | * АргументыПослеВставкиИзБО   содержит в себе таблицу, в которую произошла вставка, результат сверки данных и флаг обработки события | Вызывается после вставки из буфера обмена. В базовой реализации возвращает аргумент без изменений |
| ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы | * АргументыПередУдалениемСтрокиДинамическойТаблицы аргументы   содержит в себе динамическую таблицу, строку данных, флаг отмены удаления и список сообщений операции | void | Вызывается перед удалением строки из любой динамической таблицы |
| ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы | * АргументыДобавленияСтрокиДинамическойТаблицы аргументы   содержит в себе динамическую таблицу и строку данных | void | Вызывается после добавления строки в любой динамической таблице |
| ОбработатьСобытиеПередОткрытиемТаблицы | * АргументыПередОткрытиемТаблицы аргументы   содержит в себе таблицу данных, параметры обработки и заголовок таблицы | void | Вызывается перед открытием любой таблицы |
| ОбработатьСобытиеПриОткрытииТаблицы | * АргументыПриОткрытииТаблицы аргументы   содержит в себе таблицу данных и параметры обработки | void | Вызывается при открытии любой таблицы |
| BeforeVerifyData | * BeforeVerifyDataArgs arguments   содержит в себе флаг отмены проверки данных и дальнейшего сохранения формы и сообщение, которое будет показано пользователю в случае отмены проверки данных и дальнейшего сохранения формы | * BeforeVerifyDataArgs   содержит в себе флаг отмены проверки данных и дальнейшего сохранения формы и сообщение, которое будет показано пользователю в случае отмены проверки данных и дальнейшего сохранения формы | Вызывается до проверки данных формы. В базовой реализации возвращает аргумент без изменений |
| СформироватьОграниченияМакрасом |  | * Dictionary<string, Dictionary<string, string>>   Словарь словарей ограничений | Позволяет добавить ограничения из макроса, в базовой реализации возвращает null |
| GetPrintFormTableParameters | * string macroName,   наименование макроса формы, обрабатывающего ПФ   * string printFormTemplateName   наименование шаблона ПФ | * PrintFormTableParameters   содержит флаг отмены заполнения параметров отчета по таблицам формы и список кодов таблиц, для которых необходимо заполнить параметры отчета | Позволяет получить DTO для определения варианта взаимодействия с данными таблиц формы при заполнении параметров отчета печатной формы, в базовой реализации возвращает null |
| ПередОбработкойИстории | * ИсторияСборкиИтоговыхОтчетов История, * ТаблицаДанных Таблица, * ref Секция Секция | bool | Обработка истории сборки формы, метод должен возвратить false, если необходима стандартная обработка. В базовой реализации возвращает false |
| ПослеОбработкиИстории | * ИсторияСборкиИтоговыхОтчетов История, * ТаблицаДанных Таблица, * ref Секция Секция | void | Пост-обработка истории сборки формы |
| ПередОбработкойЗаписиИстории | * ЗаписьИсторииСборкиИтоговыхОтчетов ЗаписьИстории, * ТаблицаДанных Таблица, * ref Секция Секция | bool | Обработка записи истории сборки формы, метод должен возвратить false, если необходима стандартная обработка. В базовой реализации возвращает false |
| ПослеОбработкиЗаписиИстории | * ЗаписьИсторииСборкиИтоговыхОтчетов ЗаписьИстории, * ТаблицаДанных Таблица, * ref Секция СекцияОтчета | void | Пост-обработка записи истории сборки формы |
| ПолучитьМодельЭкраннойФормы |  | МодельЭкраннойФормы | При переопределении в дочернем классе получает объект МодельЭкраннойФормы |
| ПолучитьВнутриформенныеУвязкиДляПроверки | * СписокНовыхУвязок увязки,   список увязок, из которых будет происходить заполнение увязок формы   * List<string> закладки   список кодов таблиц, ячейки которых учавствуют во внутриформенных увязках | void | В базовой реализации заполняет список увязок формы увязками из переданного списка, теми которые используют ячейки из переданных закладок |
| ПроверитьУвязки | * СписокНовыхУвязок списокУвязок, * РезультатПроверкиУвязок результатПроверкиУвязок | void | В базовой реализации метод проверяет увязки из первого параметра, и заносит результаты проверки во второй параметр |
| ПолучитьМежформенныеУвязкиДляПроверки | * СписокНовыхУвязок увязки   список увязок, из которых будет происходить заполнение увязок формы | void | Заполняет список увязок формы увязками из переданного списка |
| ПолучитьФормыДляПроверкиМежформенныхУвязок |  | * List<string>   Список идентификаторов форм | При переопределении в дочернем классе метод получения идентификаторов форм, используемых при проверке межформенных увязок. В базовой реализации |
| ПолучитьФормыДляПроверкиМежформенныхУвязокСОтчетнымиПериодами |  | * List<Пара<string,IКомпонентОтчетногоПериода>>   Список объектов идентификаторов форм и компонента отчетного периода текущей формы | В базовой реализации получает идентификаторы форм, используемых при проверке межформенных увязок, связанные с компонентом отчетного периода текущей формы |
| ЗаполнитьФормы | * List<ДанныеФормы> Формы   Список данных форм | void | При переопределении в дочернем классе метод используется для заполнения форм |
| ПолучитьСловарьКонстант |  | * Dictionary<string, object>   Словарь значений констант по кодам констант | При переопределении в дочернем классе метод используется для получения словаря пользовательских констант. Далее пользовательские константы могут быть использованы в шаблонах экранной и печатной формы. |
| ПолучитьСписокЗапросовФорм | * string ИмяОперации   наименование операции, для которой происходит получение данных форм | * List<ЗапросНаПолучениеФормы>   список запросов данных форм, из которого будут получаться данные форм в соотвествующей обработке | При переопределении в дочернем классе метод позволяет получить данные других форм, необходимые для тех или иных обработок форм. |
| ОсвободитьДополнительныеРесурсы |  | void | При переопределении в дочернем классе метод используется для освобождения дополнительных ресурсов памяти. Вызывается в методе ОсвободитьРесурсы() базового обработчика формы |

# Работа с константами

**Константа экранной формы** – текст определенного формата, предназначенный для адресации к данным справочников или системы. В экранной форме вместо констант автоматически подставляются значения, к которым они обращаются (например, код учреждения из справочника учреждений, или текущий отчетный период из справочника отчетных периодов). Константы могут использоваться как в шапке экранной формы, так и в ее таблицах или закладках.

Для объявления константы используется следующий синтаксис:

#Константа#

Также константы можно использовать в тексте. В этом случае для объявления константы используется следующий синтаксис:

Произвольный текст **#Константа#** продолжение текста.

Таблица 20 – Перечень доступных констант

| Константа | Описание |
| --- | --- |
| Учреждение.Наименование | подтягивается полное наименование из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.НаименованиеРод | подтягивается полное наименование из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы в родительном падеже |
| Учреждение.ИНН | подтягивается реквизит ИНН из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ПФР | подтягивается реквизит ПФР из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.КПП | подтягивается реквизит КПП из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКОНХ | подтягивается реквизит ОКОНХ из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКУД | подтягивается реквизит ОКУД из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКПО | подтягивается реквизит ОКПО из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКАТО | подтягивается реквизит ОКАТО из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.НомерФилиала | подтягивается реквизит НомерФилиала из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКВЭД | подтягивается реквизит ОКВЭД из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКОПФ | подтягивается реквизит ОКОПФ из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКИН | подтягивается реквизит ОКИН из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКОГУ | подтягивается реквизит ОКОГУ из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКФС | подтягивается реквизит ОКФС из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.ОКТМО | подтягивается реквизит ОКТМО из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.Адрес | подтягивается адрес из справочника «УЧРЕЖДЕНИЯ» по учреждению текущей отчетной формы |
| Учреждение.АдресФиас | подтягивается адрес Фиас |
| Учреждение.АдресГар | подтягивается адрес Фиас |
| Учреждение.ГлаваПоБК | подтягивает строковое значение «Глава по БК» ближайшее к дате начала отчетного периода. Если для текущего учреждения не найден объект ГлаваПоБК, то в константу подтянется  "000". |
| КомпонентОтчетногоПериода | подтягивается код компонента отчетного периода текущей отчетной формы |
| КомпонентОтчетногоПериода.ПолноеНаименование | подтягивается полное наименование компонента отчетного периода текущей отчетной формы |
| ОтчетныйПериод | подтягивается код отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ПолноеНаименование | подтягивается наименование отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ДатаНачала | подтягивается дата начала отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ДатаНачалаПодотчетногоПериода | подтягивается дата начала подотчетного отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ДатаОкончания | подтягивается дата окончания отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ДатаОкончанияПодотчетногоПериода | подтягивается дата окончания подотчетного отчетного периода текущей отчетной формы |
| ОтчетныйПериод.ПрофильПользователя | подтягивается профиль пользователя отчетного периода текущей отчетной формы |
| ОтчетныйПериод.КодСПериодом | подтягивается код отчетного периода текущей отчетной формы в строковом формате (Пример: Годовой отчетный период (01.01.2008 - 31.01.2008)). |
| ОтчетныйПериод.Квартал | подтягивается квартал подотчетного периода текущей отчетной формы в строковом формате (Пример: Третий) |
| ОтчетныйПериод.КварталОтчетногоПериода | подтягивается квартал отчетного периода текущей отчетной формы в строковом формате (Пример: Первый) |
| ОтчетныйПериод.НомерКвартала | подтягивается номер квартала отчетного периода текущей отчетной формы (Пример: 3) |
| ТекущаяДата | подтягивается текущая дата |
| Сегодня | подтягивается текущая дата |
| Сейчас | подтягиваются текущие дата и время |
| НачалоГода | подтягивается 1 января текущего года |
| КонецГода | подтягивается 31 декабря текущего года |
| Год | подтягивается текущий год |
| Месяц | подтягивается текущий месяц |
| День | подтягивается текущий день |
| Метаструктура.Авторство | подтягивается автор формы |
| Метаструктура.ВерсияМетаописания | подтягивается версия формы |
| Метаструктура.Группа | подтягивается наименование группы формы |
| Метаструктура.ДатаНачалаДействия | подтягивается дата начала периода действия формы |
| Метаструктура.ДатаОкончанияДействия | подтягивается дата окончания периода действия формы |
| Метаструктура.Код | подтягивается код метаструктуры формы |
| Метаструктура.Наименование | подтягивается наименование формы |
| Метаструктура.Тег | подтягивается тег |
| ЭЦП.Подробно | подтягивает информацию о сертификате, например,  L=Казань, O=Барс Груп, CN=Иван |
| ЭЦП.Действует до | подтягивает информацию о сроке окончания действия сертификата |
| ЭЦП.Выдано | подтягивает информацию о том, кем выдан сертификат |
| ЭЦП.Алгоритм ключа | подтягивает алгоритм ключа |
| ЭЦП.Серийный номер | подтягивает серийный номер сертификата |
| ЭЦП.Дата подписания | подтягивает дату подписания формы ЭЦП |

**Учреждение**

Константы учреждения заполняются не на основе текущего учреждения (что отображает консоль администрирования), а на основе значений учреждения, которые были на момент окончания ОП.

Система берет ближайшую запись в истории изменения ХО и заполняет форму на основе этих данных. Подробный алгоритм:

1. взять дату окончания ОП;
2. получить записи истории изменения учреждения, такие где «ДатаПоследнегоИзменения» меньше или равна дате окончания ОП;
3. отсортировать по убыванию по значению ДатаПоследнегоИзменения;
4. взять первую запись;
5. если записей нет, то вернуть текущее учреждение;
6. проверить, если «ДатаПоследнегоИзменения» у текущего учреждения больше даты окончания ОП, то вернуть запись, найденную на шаге г);
7. если «ДатаПоследнегоИзменения» у текущего учреждения ближе к дате окончания ОП, то вернуть текущее учреждение, иначе вернуть запись, найденную на шаге г).

**ОтчетныйПериод.Квартал**

Для вычисления используется «ДатаНачалаПодотчетногоПериода».

«ДатаНачалаПодотчетногоПериода» вычисляется следующим образом:

Если поле «Начало подотчетного периода» не заполнено, то вычисляется следующим образом:

Из «ДатаНачала» отчетного периода вычитатся количество месяцев или дней, в зависимости от «ТипОтчетногоПериода».

Если у ОП не заполнено поле «Начало подотчетного периода», то берется дата начала:

01.01.2021 – 3 месяца = 01.10.2020, что и является четвертым кварталом.

01.04.2021 – 3 месяца = 01.01.2020, что и является первым кварталом.

В случае «КварталОтчетногоПериода» для вычисления используется «ДатаНачала».

## Форматирование констант

Предусмотрена возможность получения форматированной константы. Пример форматирования констант приведен ниже (Таблица 21).

Таблица 21 – Форматирование констант

|  |  |
| --- | --- |
| #ОтчетныйПериод.ДатаНачала, MMMMM yyyy# г. | возвращает строку: июнь 2008 г. |
| #ОтчетныйПериод.ДатаНачала, MM yyyy# г. | возвращает строку: 06 2008 г. |
| #ОтчетныйПериод.ДатаНачала, D# | отображает дату отчетного периода |

## Заполнение констант через макрос

В Дизайнере форм существует возможность создания собственных констант, например, константы: #ПрошлыйГод#. Создание пользовательских констант осуществляется в разделе «Макросы» Дизайнера форм.

Для заполнения словаря констант пользовательскими константами необходимо в форме переопределить метод **ПолучитьСловарьКонстант()**.

Вызов осуществляется при открытии формы. Метод возвращает список констант типа Dictionary<string, object>, где ключ – наименование константы, значение – значение константы.

Пример заполнения констант:

|  |
| --- |
| public override Dictionary<string, object> ПолучитьСловарьКонстант()  {  var constans = new Dictionary<string, object>  {  { "ПредыдущийГод", GetLastYear() }  };    return constans;  }    //Метод, определяющий значение константы ПредыдущийГод  private int GetLastYear()  {  var reportingPeriodComponent = Форма.Идентификатор.КомпонентОтчетногоПериода;    if (reportingPeriodComponent?.IОтчетныйПериод == null)  {  return DateTime.MinValue.Year;  }    return reportingPeriodComponent.IОтчетныйПериод.ДатаНачала.Year - 1;  } |

Пример получения констант:

|  |
| --- |
| private Dictionary<string, object> GetСonstants()  {  var constants = ПолучитьСловарьКонстант();    return constants;  } |

# Создание печатной формы

## XLSX и XLSM

Печатная форма (ПФ) представляет собой готовый к печати документ, который содержит данные отчетной формы. Печатная форма создается на основе шаблона печатной формы. Шаблон определяет внешний вид печатной формы и содержащиеся в ней данные отчетной формы.

В шаблоне содержится разметка, по которой автоматически производится заполнение данными в процессе формирования ПФ: данные свободных ячеек, данные статических таблиц и данные динамических таблиц.

Например, для заполнения данными свободных ячеек в шаблоне указывается код свободной ячейки (в начале и в конце введенного кода необходимо поставить символ $). Подробнее о формировании шаблона ПФ описано в инструкции по работе с Дизайнером отчетных форм.

Если функции автоматического формирования и заполнения ПФ по шаблону недостаточно, необходимо использовать макросную обработку.

Посмотреть текущий или назначить свой макрос можно:

* в конфигурационном xml-файле (с наменованием: (ПФ).НаименованиеФормы.xml) (Рисунок 11);

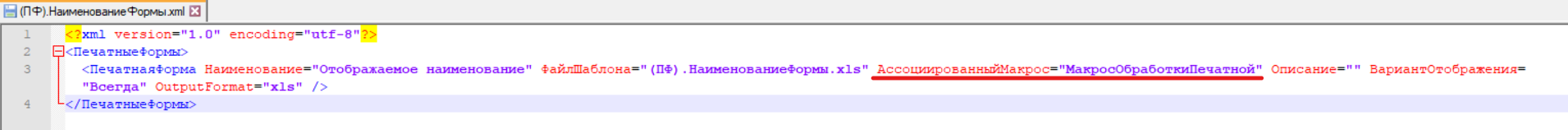


Рисунок 11 – Конфигурационный xml файл

* в дизайнере отчетных форм (Рисунок 12).

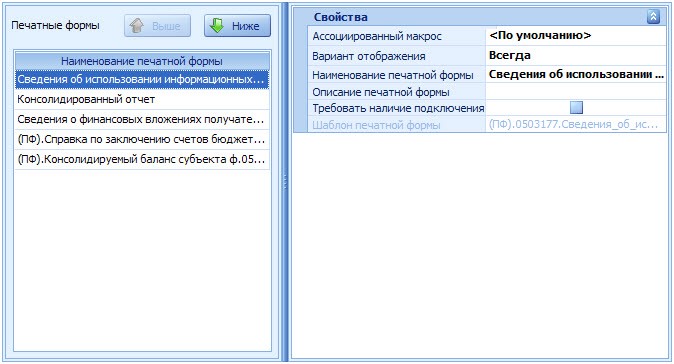


Рисунок 12 – Дизайнер отчетных форм

Далее в макросе реализуется дополнительная логика формирования ПФ.

Дополнительный логику заполнения ПФ можно реализовать на основе секций (см.п.12.4.) или через IopenXmlWorkbook.

Для того чтобы изменить шаблон через код, выполните следующие действия:

* скопируйте выгружаемый файл в отдельное место (если оставите как есть, то, ввиду того, что с одним файлом может работать несколько человек, изменения в файле шаблона из метаструктуры формы отразятся и на остальных пользователях. Как подменить выгружаемый файл см. п. 12.4.
* откройте изменяемый шаблон с помощью IOpenXmlWorkbook и внесите необходимые корректировки.

### Работа с файлами excel через IOpenXmlWorkbook

#### Раскрытие

Для работы с excel документом необходимо из контейнера получить IOpenXmlWorkbook

|  |
| --- |
| var openXmlWorkbook = КонтейнерЗависимостей.Текущий.Resolve<IOpenXmlWorkbook>();  openXmlWorkbook.Open("\*Полный путь до excel файла строкой\*"); |

#### Свойства и методы IOpenXmlWorkbook (документа)

Таблица 22 – Свойства IopenXmlWorkbook

| Свойство | Тип | Описание |
| --- | --- | --- |
| ActiveSheet | ISheet | Ссылка на текущий лист, с которым работаем (назначается с помощью метода SetActiveSheet()) |

Таблица 23 – Методы IOpenXmlWorkbook

| Метод | Возвращаемый тип | Параметры | Описание |
| --- | --- | --- | --- |
| CopySheet(string name, string newSheetName); | ISheet | Name - Имя копируеумого листа  newSheetName - Имя нового листа | Скопировать лист |
| CopySheet(IOpenXmlWorkbook sourceFile, string sourceSheetname, string newSheetName); | ISheet | sourceFile –документ из которого копируем лист  sourceSheetname – наименование исходного листа  newSheetName – наименование нового листа | Скопировать лист |
| Create(string path); | void |  | Создать нвый файл xlsx |
| CreateExcelStyle(ref CellStyle cellStyle, string styleName); | void | cellStyle - Описание стиля styleName - Имя стиля | Зарегистрировать именнованный стиль ячейки |
| CreateNewSheet(string name = "Report"); | void | name - Наименование листа (по умолчанию – Report) | Создать новый лист |
| GetByName(string name); | ISheet | Name - Имя листа | Получить лист по наименованию |
| HideSheet(string name); | void | Name - Наименование листа | Скрыть лист |
| IsSheetProtected(int sheetIndex); | bool | sheetIndex - номер листа | Проверка защиты листа |
| Open(string path); | void | Path - Путь до существующего файла | Открыть файл xlsx |
| Protect(string password); | void | Password - пароль | Защитить книгу паролем |
| Protect(int sheetIndex, string password, SheetProtectionModes? modes = null); | void | sheetIndex - номер листа password – пароль  modes - Режим защиты | Защитить лист книги паролем |
| Save(); | void |  | Сохранить файл |
| SaveAs(string path); | void |  | Сохранить по новому пути |
| SetActiveSheet(int index); | void | Index - Номер листа | Выбрать текущий лист |
| SetActiveSheet(string name); | void | Name - Имя листа | Выбрать текущий лист |
| SetDefaultFontSize(float size); | void | Size - Размер шрифта | Установить размер шрифта по умолчанию |
| SheetName(int sheetIndex); | string | sheetIndex - Индекс листа | Получить имя листа |
| UnProtect(int sheetIndex, string password); | void | sheetIndex - номер листа  password - пароль | Снять защиту с листа |
| WorksheetsCount(); | int |  | Число листов |
| RenameSheet(int sheetIndex, string newName); | void | sheetIndex - Номер листа newName - Новое наименование листа | Переименовать лист |
| RemoveSheet(string name); | void | Name - Имя листа | Удалить лист |

Для работы с листом необходимо назначить его через метод SetActiveSheet и получить объект ISheet через свойство ActiveSheet.

#### Свойства и методы ISheet (листа)

Таблица 24 – Свойства ISheet

| Свойство | Тип | Описание |
| --- | --- | --- |
| Name | string | Наименование листа |
| Index | int | Номер листа |
| IsProtected | bool | Проверить, является ли лист защищенным |
| DisableMergeValidation | bool | Отключить валидацию перед объединением диапазона ячеек листа |

Таблица 25 – Методы ISheet

| Метод | Возвращаемый тип | Параметры | Описание |
| --- | --- | --- | --- |
| AutofitColumns(); | void |  | Выставить автоширину колонок  (Ячейки с формулами, ячейки в WrapText = true и объединенные ячейки игнорируются) |
| CopyRow(int sourceIndex, int targetIndex); | void | sourceIndex - Номер строки-источника  targetIndex - Номер строки-приемника | Скопировать строку |
| DeleteColumn(int columnIndex); | void | columnIndex - Индекс столбца | Удалить столбец |
| DeleteRow(int rowIndex); | void | rowIndex - Индекс строки | Удалить строку |
| GetAddress(int startRow, int startCol, int? endRow = null, int? endCol = null); | string | startRow - Первая строка диапазона  startCol - Первый столбец диапазона endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Получить относительный адрес диапазона ячеек |
| GetAddressGlobal(int startRow, int startCol, int? endRow = null, int? endCol = null); | string | startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Получить глобальный адрес диапазона ячеек, включающий имя листа |
| GetAddressGlobal(string rangeName); | string | rangeName - Имя именованного диапазона ячеек | Получить глобальный адрес диапазона ячеек, включающий имя листа |
| GetDimensions(); | SheetDimensions |  | Размеры листа |
| GetFormula(int row, int col); | string | row - Строка  col - Столбец | Получить формулу, сохраненную в ячейке |
| GetMergeAreaDimensions(int startRow, int startCol, int? endRow = null, int? endCol = null); | MergeAreaDimensions? | startRow - Первая строка диапазона  startCol - Первый столбец диапазона endRow - Последняя строка диапазона – endCol - Последний столбец диапазона | Получить параметры объедиеннной ячейки |
| GetNamedRangesNames(); | IEnumerable<string> |  | Получить итератор по именам именованных диапазонов ячеек |
| GetText(int startRow, int startCol, int? endRow = null, int? endCol = null); | string | startRow - Первая строка диапазона startCol - Первый столбец диапазона endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Получить текстовое представление значения диапазона ячеек |
| GetFormat(int startRow, int startCol, int? endRow = null, int? endCol = null); | string | startRow - Первая строка диапазона startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Получить формат ячейки |
| GetValue(int row, int col); | object | Row – Строка  Col - Столбец | Получить значение ячейки листа |
| GetValueAs<TValue>(int row, int col); | TValue | Row – Строка  Col - Столбец | Получить значение ячейки листа |
| GetValueByRangeName(string rangeName); | object | rangeName - Имя диапазона | Получить значение ячейки листа |
| Group(CellsGroupMode groupMode, int from, int to, bool collapsed = true); | void | groupMode - Режим группировки  from - Индекс перового столбца/строки  to - Индекс последнего столбца/строки  collapsed - Группа свернута | Сгруппировать строки/столбцы |
| HasFormula(int row, int col); | bool | Row – Строка  Col - Столбец | Флаг наличия формулы в ячейке |
| Hide(); | void |  | Скрыть лист |
| void HideColumn(int columnIndex); |  | columnIndex - Индекс столбца | Скрыть столбец |
| HideRow(int rowIndex); | void | rowIndex - Индекс строки | Скрыть строку |
| IncreaseColumnWidth(int columnIndex, double width); | void | columnIndex - Номер колонки  width - Ширина | Если текущая ширина меньше указанной, ширина будет увеличена |
| InsertColumn(int columnIndex, int count = 1); | void | columnIndex - Индекс столбца, начиная с которого будут добавлены новые столбцы  count - Количество вставляемых столбцов | Вставить столбцы |
| InsertRow(int rowIndex, int count = 1); | void | rowIndex - Индекс строки, начиная с которого будут добавлены новые строки  count - Количество вставляемых строк | Вставить строки |
| IsMerged(int startRow, int startCol, int? endRow = null, int? endCol = null); | bool | startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Проверить, принадлежит ли левая верхняя ячейка к объединенной ячейке |
| Protect(string password, SheetProtectionModes? modes = null); | void | Password – Пароль  Modes - Режимы защиты | Защитить лист |
| ReduceColumnWidth(int columnIndex, double width); | void | columnIndex - Номер колонки  width - Ширина | Если текущая ширина больше, либо равна указанной, ширина будет уменьшена |
| SetAutoFilters(int startRow, int startCol, int? endRow = null, int? endCol = null); | void | startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Установить автофильтр в ячейках диапазона |
| SetAutoHeightToRow(int rowIndex); | void | rowIndex - Индекс строки | Установить автовысоту строки (Работает, только если строка не содержит объединенных ячеек) |
| SetColumnStyle(int columnIndex, ref CellStyle style); | void | columnIndex - Номер столбца  style - Описание стиля ячеек | Установить стиль столбца |
| SetColumnWidth(int columnIndex, double width); | void | columnIndex - Индекс столбца  width - Ширина столбца | Установить ширину столбца |
| SetFormula(int row, int col, string formula); | void | Row – Строка  Col – Столбец  Formula - Формула | Установить формулу в качестве значения ячейки |
| SetFormula(string address, string formula); | void | Address - Адрес ячейки  Formula – Формула | Установить формулу в качестве значения ячейки |
| SetFormulaArray(int row, int col, string formula); | void | Row – Строка  Col – Столбец  Formula - Формула | Установить формулу для операций над векторами ячеек |
| SetNamedRange(string name, int startRow, int startCol, int? endRow = null, int? endCol = null); | void | Name – Наименование  startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Задать именованный диапазон ячеек |
| SetNumber(int row, int col, decimal number, string excelFormatString = "#,##0.00"); | void | Number – Число  Row – Строка  Col – Столбец  excelFormatString - Формат отображения числа | Установить значением ячейки число |
| SetNumberWithDefaultCellStyle(int row, int col, decimal number, string excelFormatString = "#,##0.00"); | void | Row – Строка  Col – Столбец  Number – Число  excelFormatString - Формат отображения числа | Установить значением ячейки число и применить стиль ячейки по умолчанию (Шрифт, тонкие границы) |
| SetRepeatRowsAddress(string address); | void | Address - Адрес диапазона ячеек | Установить диапазон ячеек, которые будут выводиться на печать на каждой странице |
| SetRowHeight(int rowIndex, double height); | void | rowIndex - Индекс строки  height - Высота строки | Установить высоту строки |
| SetRowStyle(int rowIndex, ref CellStyle style); | void | rowIndex - Номер строки  style - Описание стиля ячеек | Установить стиль строки |
| SetText(int row, int col, string text); | void | Row – Строка  Col – Столбец  Text - Строка | Установить строковое значение ячейки |
| SetTextWithDefaultCellStyle(int row, int col, string text); | void | Row – Строка  Col – Столбец  Text – Строка | Установить строковое значение ячейки и применить стиль ячейки по умолчанию (Шрифт, тонкие границы) |
| SetupPage(PageOrientation? orientation = null,                decimal? topMargin = null,                decimal? bottomMargin = null,                decimal? leftMargin = null,                decimal? rightMargin = null,                int? zoom = null,                bool? centerHorizontally = null,                bool isGridLineVisible = true); | void | Orientation – Ориентация  topMargin - Верхнее поле bottomMargin - Нижнее поле leftMargin - Левое поле  rightMargin - Правое поле  zoom – Масштаб  centerHorizontally - Центрировать горизонтально  isGridLineVisible - Отображать границы ячеек в excel при просмотре файла | Установить параметры печатной страницы |
| SetValue(int row, int col, object value); | void | Row – Строка  Col – Столбец  Value - Значение | Установить значение ячейки листа |
| SetTypedValue<TValue>(int row, int col, TValue value, string excelFormatString = null); | void | Row – Строка  Col – Столбец  Value – Значение  excelFormatString - Строка формата значения для excel | Установить значение ячейки листа |
| TryGetRangePositions(string rangeName, out int row, out int col); | bool | rangeName - Уникальное имя ячейки  row – Строка  col - Столбец | Получить координаты именованной ячейки |
| UnMerge(int startRow, int startCol, int? endRow = null, int? endCol = null); | void | startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона | Убрать объединения со всех ячеек диапазона |
| UnProtect(string password); | void | Password - Пароль | Снять защиту с листа |
| UsingRange(int startRow, int startCol, int? endRow = null, int? endCol = null, Action<ICellsRange> action = null); | void | startRow - Первая строка диапазона  startCol - Первый столбец диапазона  endRow - Последняя строка диапазона  endCol - Последний столбец диапазона  action - Действие над диапазоном ячеек | Выполнить действия над диапазоном ячеек |

|  |
| --- |
| public void PrintFormMacros(ref Барс.Отчеты.ПараметрыОтчета параметрыОтчета, ОтчетнаяФормаДанных отчетнаяФорма)  {  var pathOfCopyOfPrintFormTemplate = GetPathOfCopyOfPrintFormTemplate(параметрыОтчета.ПутьКШаблонуПечатнойФормы);  var openXmlWorkbook = КонтейнерЗависимостей.Текущий.Resolve<IOpenXmlWorkbook>();  openXmlWorkbook.Open(pathOfCopyOfPrintFormTemplate);    FillPrintForm(openXmlWorkbook);    параметрыОтчета.ПутьКШаблонуПечатнойФормы = pathOfCopyOfPrintFormTemplate;  }    private string GetPathOfCopyOfPrintFormTemplate(string sourcePrintFormTemplatePath)  {  var templateFilePath = Path.Combine(Path.GetTempPath(), string.Format("{0}{1}", Guid.NewGuid(), Path.GetExtension(sourcePrintFormTemplatePath)));    File.Copy(sourcePrintFormTemplatePath, templateFilePath, true);  File.SetAttributes(templateFilePath, FileAttributes.Normal);    return templateFilePath;  }    private void FillPrintForm(IOpenXmlWorkbook книгаXls)  {  var currentSheet = книгаXls.ActiveSheet;    var value = currentSheet.GetValue(row: 10, col: 10).ToString();    currentSheet.SetText(row:10, col: 10, text: "testText");    книгаXls.Save();  } |

## DOCX

Точкой входа для начала работы с docx является интерфейс IOpenXmlWordDocument.

Получить его можно с помощью DI либо через Service locator КонтейнерЗависимостей (КонтейнерЗависимостей.Текущий.Resolve<IOpenXmlWordDocument>()).

Таблица 26 – Методы

| Метод | Описание |
| --- | --- |
| IWordDocument Create(string filePath = "") | Создать новый документ.  filePath - путь для сохранения нового документа |
| IWordDocument Open(string filePath) | Открыть существующий документ.  filePath - путь к файлу для открытия |
| IWordDocument Open(Stream stream) | Открыть документ.  stream - поток файла для открытия |

### Примеры кода

#### Добавление текста со стилями

|  |
| --- |
| using System.Drawing;  using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example1  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить новый параграф и сразу указать его форматирование  document.AddParagraph("Первый параграф. Жирный курсив зеленого цвета",  new Formatting { Bold = true, Italic = true, Color = Color.Green });    // Добавить новый параграф  document.AddParagraph("Второй параграф. Шрифт Open Sans 24 размера с серым фоном")  .SetFont("Open Sans") // Задать шрифт  .SetFontSize(24) // Размер шрифта  .SetBackgroundColor(Color.Gray); // Цвет фона    document.AddParagraph("Третий параграф. Отступ первой строки 25")  .AppendLine("Вторая строка")  .AppendLine("Третья строка")  .SetIndentationOfFirstLine(25); // Задать отступ первой строки слева 25    // Сохранить документ  document.SaveAs("document.docx");  }  } |

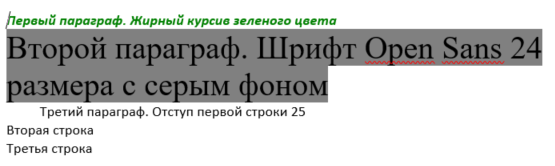


Рисунок 13 – Результат

#### Выравнивание текста

|  |
| --- |
| using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example2  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить новый параграф  document.AddParagraph("1 параграф. Выравнивание по центру")  .SetTextAlignment(HorizontalAlignment.Center); // Задать выравнивание текста по горизонтали    document.AddParagraph("2 параграф. Выравнивание по правому краю")  .SetTextAlignment(HorizontalAlignment.Right);    document.AddParagraph("3 параграф. Выравнивание по левому краю")  .SetTextAlignment(HorizontalAlignment.Left);    document.AddParagraph("4 параграф. Выравнивание по ширине")  .SetTextAlignment(HorizontalAlignment.Both);    // Сохранить документ  document.SaveAs("document.docx");  }  } |

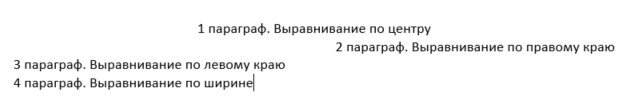


Рисунок 14 – Результат

#### Интервалы и отступы

|  |
| --- |
| using System.Drawing;  using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example3  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить новый параграф  document.AddParagraph("1 параграф. Интервал после 10. Отступ слева 50")  .SetSpacingAfter(10) // Задать интервал 10  .SetIndentationBefore(50); // Задать отступ перед параграфом слева 50    document.AddParagraph("2 параграф. Интервал перед 10. Отступ слева 75")  .SetSpacingBefore(10)  .SetIndentationBefore(75);    // Добавить новый параграф  document.AddParagraph("3 параграф междустрочный интервал 25")  .AppendLine("Вторая строка") // Добавить новую строку в параграф  .AppendLine("Третья строка")  .SetLineSpacing(25); // Задать междустрочный интервал 25    // Сохранить документ  document.SaveAs("document.docx");  }  } |

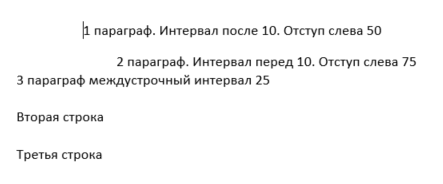


Рисунок 15 – Результат

#### Добавление списков

|  |
| --- |
| using System.Drawing;  using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example4  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить новый параграф  var par = document.AddParagraph();  // Создать нумерованный список  document.CreateNumberedList()  .AddItem("текст") // добавить элемент в список  .AddItem("текст")  .AddItem("текст")  .InsertAfter(par); // вставить список после параграфа    // Создать маркированный список  document.CreateBulletedList()  .AddItem("текст") // добавить элемент в список  .AddItem("текст")  .AddItem("текст")  .InsertAfter(par); // вставить список после параграфа    // Сохранить документ  document.SaveAs("document.docx");  }  } |

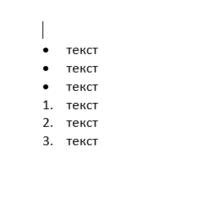


Рисунок 16 – Результат

#### Добавление верхних и нижних колонтитулов и нумерация страниц

|  |
| --- |
| using System.Drawing;  using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example5  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить нижние колонтитулы  document.AddFooters();    // Добавить верхние колонтитулы  document.AddHeaders();    // Особый колонтитул для первой страницы  document.DifferentHeaderAndFooterForFirstPage = true;    // Разные колонтитулы для четных и нечетных страниц  document.DifferentHeadersAndFootersForOddAndEvenPages = true;    // На первой странице есть номер в формате "Страница 1 из N"  document.Footers.First  .AddParagraph("Страница ")  .AppendPageNumber(PageNumberFormat.Normal)  .AppendText(" из ")  .AppendPageCount(PageNumberFormat.Normal);    // На нечетных страницах нет номеров  document.Footers.Odd.PageNumbers = false;    // На четных страницах есть номера  document.Footers.Even.PageNumbers = true;    // Сохранить документ  document.SaveAs("document.docx");  }  } |

#### Копирование текста из параграфа в параграф

|  |
| --- |
| using System.Linq;  using Барс.Отчеты;    public class Example6  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Открыть существующий документ document1.docx  using var document1 = WordDocument.Open("document1.docx");    // Берем параграф из первого документа  var paragraphFromDocument1 = document1.Paragraphs.First();    // Получаем текст параграфа  var textFromDocument1 = paragraphFromDocument1.GetText();    // Создаем новый документ  using var document2 = WordDocument.Create();    // Добавляем параграф с текстом из первого документа  document2.AddParagraph(textFromDocument1);    // Сохраняем документ  document2.SaveAs("document2.docx");  }  } |

#### Работа с изображениями

|  |
| --- |
| using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example7  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавить параграф и вставить в него изображение размером 800х600  var picture = document.AddParagraph()  .AppendPicture("picture.png", 600, 800);    // Повернуть изображение на 45 градусов  picture.Rotation = 45;    // Установить форму изображения эллипс  picture.SetShape(BasicShapes.Ellipse);    // Сохраняем документ  document.SaveAs("document.docx");  }  } |

#### Работа с таблицами

|  |
| --- |
| using System.Drawing;  using Барс.Отчеты;    public class Example8  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Добавляем параграф с текстом "Таблица"  var paragraph = document.AddParagraph("Таблица");    // Добавляем таблицу на 30 строк и 5 столбцов после параграфа  var table = paragraph.AddTableAfterSelf(30, 5);    var rowIndex = 0;  foreach (var row in table.Rows)  {  if (rowIndex == 0)  {  // Строка заголовка  row.TableHeader = true;    var columnIndex = 0;  foreach (var cell in row.Cells)  {  cell.AddParagraph($"Заголовок столбца {columnIndex}");  columnIndex++;  }  }  else  {  var columnIndex = 0;  foreach (var cell in row.Cells)  {  cell.AddParagraph($"Строка {rowIndex}. Столбец {columnIndex}");  columnIndex++;  }  }    rowIndex++;  }    // Удалить последнюю строку  table.RemoveRow();    // Добавить новую строку в конец  var lastRow = table.AddRow();    // Удалить последний столбец  table.RemoveColumn();    // Добавить новый столбец в конец  table.AddColumn();    foreach (var cell in lastRow.Cells)  {  // Установить цвет фона ячейки таблицы  cell.BackgroundColor = Color.Gray;    // Добавить ячейке правую границу красного цвета одинарного стиля 2ого размера  cell.SetBorder(TableCellBorderType.Right, BorderStyle.Single, BorderSize.Two, 0, Color.Red);  }    var columnWidths = new float[table.ColumnCount];    for (var i = 0; i < columnWidths.Length; i++)  {  columnWidths[i] = 20 \* (i + 1);  }    // Установить ширины столбцов  table.SetWidths(columnWidths);    // Сохранить документ  document.SaveAs("document.docx");  }  } |

#### Задание разрыва страницы и параметров страницы

|  |
| --- |
| using Барс.Отчеты;    public class Example9  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Альбомная ориентация всех страниц  document.PageLayout.Orientation = PageOrientation.Landscape;    // Поля страниц по 200 (7,05 см)  document.MarginTop = 200;  document.MarginRight = 200;  document.MarginBottom = 200;  document.MarginLeft = 200;    // Добавить новый параграф  var paragraph = document.AddParagraph("Первый параграф");    // Добавить разрыв страницы после параграфа  paragraph.AddPageBreakAfterSelf();    // Добавить новый параграф  document.AddParagraph("Второй параграф на второй странице");    // Добавить разрыв секции  // Разрыв секции определяет конец секции, таким образом секция включает в себя страницы 1 и 2  var section1 = document.AddSectionPageBreak();    // Меняем ориентацию страниц относящихся к секции на портретную  section1.PageLayout.Orientation = PageOrientation.Portrait;    // Добавить новый параграф  document.AddParagraph("Третий параграф на третьей странице");    // Сохранить документ  document.SaveAs("document.docx");  }  } |

#### Добавление защиты документа с паролем и без него

|  |
| --- |
| using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example10  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    // Запрещаем редактировать, разрешаем только добавлять изменять и удалять комментарии  document.AddProtection(EditRestrictions.Comments);    // Убираем защиту  document.RemoveProtection();    // Защищаем документ паролем и запрещаем всё редактирование  document.AddPasswordProtection(EditRestrictions.ReadOnly, "P@ssw0rd");    // Сохранить документ указав пароль  document.SaveAs("document.docx", "P@ssw0rd");  }  } |

#### Диаграммы

Гистограмма/Линейчатая диаграмма:

|  |
| --- |
| using System.Collections.Generic;  using System.Drawing;    using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example11  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    var rub = ChartData.CreateRubExpenses();  var usd = ChartData.CreateUsdExpenses();  var euro = ChartData.CreateEuroExpenses();    document.CreateBarChart() // Создать гистограмму или линейчатую диаграмму  .SetBarDirection(BarDirection.Column) // Установить направление осей столбцами (гистограмма), BarDirection.Bar для линейчатой диаграммы  .SetBarGrouping(BarGrouping.Standard) // Стандартная группировка  .SetGapWidth(100) // Ширина зазора 100  .AddLegend(ChartLegendPosition.Left, false) // Добавить легенду слева  .AddSeries("RUB", Color.CornflowerBlue, rub, "Category", "Expenses") // Добавить ось RUB по категории Category и значению Expenses цвета CornflowerBlue  .AddSeries("USD", Color.LightGreen, usd, "Category", "Expenses") // Добавить ось USD по категории Category и значению Expenses цвета LightGreen  .AddSeries("EUR", Color.Gold, euro, "Category", "Expenses") // Добавить ось EUR по категории Category и значению Expenses цвета Gold  .Insert(); // Вставить диаграмму в документ    // Сохранить документ  document.SaveAs("document.docx");  }  }    internal class ChartData  {  public string Category { get; set; }  public double Expenses { get; set; }    public static List<ChartData> CreateRubExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 100 },  new() { Category = "Housing", Expenses = 120 },  new() { Category = "Transportation", Expenses = 140 },  new() { Category = "Health Care", Expenses = 150 }  };  }    public static List<ChartData> CreateUsdExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 200 },  new() { Category = "Housing", Expenses = 150 },  new() { Category = "Transportation", Expenses = 110 },  new() { Category = "Health Care", Expenses = 100 }  };  }    public static List<ChartData> CreateEuroExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 125 },  new() { Category = "Housing", Expenses = 80 },  new() { Category = "Transportation", Expenses = 110 },  new() { Category = "Health Care", Expenses = 60 }  };  }  } |

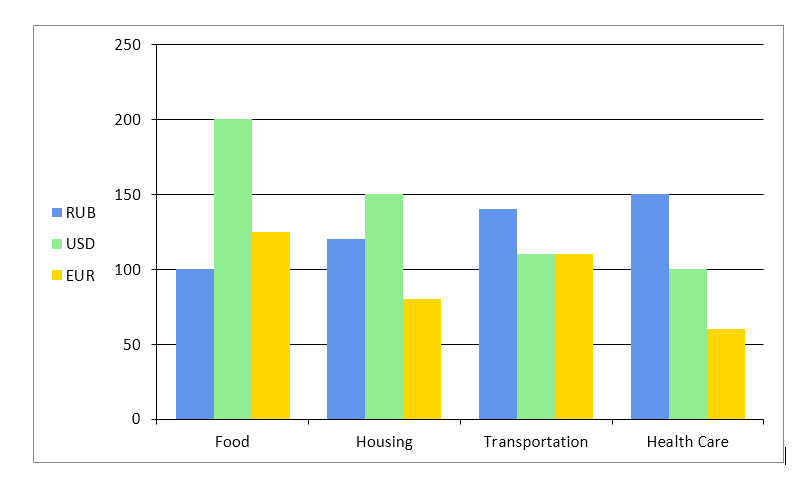


Рисунок 17 – Результат. Гистограмма/Линейчатая диаграмма

График:

|  |
| --- |
| using System.Collections.Generic;  using System.Drawing;    using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example12  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    var rub = ChartData.CreateRubExpenses();  var usd = ChartData.CreateUsdExpenses();  var euro = ChartData.CreateEuroExpenses();    document.CreateLineChart() // Создать график  .AddLegend(ChartLegendPosition.Left, false) // Добавить легенду слева  .AddSeries("RUB", Color.CornflowerBlue, rub, "Category", "Expenses") // Добавить ось RUB по категории Category и значению Expenses цвета CornflowerBlue  .AddSeries("USD", Color.LightGreen, usd, "Category", "Expenses") // Добавить ось USD по категории Category и значению Expenses цвета LightGreen  .AddSeries("EUR", Color.Gold, euro, "Category", "Expenses") // Добавить ось EUR по категории Category и значению Expenses цвета Gold  .Insert(); // Вставить диаграмму в документ    // Сохранить документ  document.SaveAs("document.docx");  }  }    internal class ChartData  {  public string Category { get; set; }  public double Expenses { get; set; }    public static List<ChartData> CreateRubExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 100 },  new() { Category = "Housing", Expenses = 120 },  new() { Category = "Transportation", Expenses = 140 },  new() { Category = "Health Care", Expenses = 150 }  };  }    public static List<ChartData> CreateUsdExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 200 },  new() { Category = "Housing", Expenses = 150 },  new() { Category = "Transportation", Expenses = 110 },  new() { Category = "Health Care", Expenses = 100 }  };  }    public static List<ChartData> CreateEuroExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 125 },  new() { Category = "Housing", Expenses = 80 },  new() { Category = "Transportation", Expenses = 110 },  new() { Category = "Health Care", Expenses = 60 }  };  }  } |



Рисунок 18 – Результат. График

Круговая:

|  |
| --- |
| using System.Collections.Generic;  using System.Drawing;    using Барс.Отчеты;  using Ядро.ГенераторОтчетов.ГенераторОтчетовWord.Enums;    public class Example13  {  public IOpenXmlWordDocument WordDocument { get; set; }    public void CreateDocument()  {  // Создать новый документ  using var document = WordDocument.Create();    var rub = ChartData.CreateRubExpenses();    var pieChart = document.CreatePieChart() // Создать круговую диаграмму  .AddLegend(ChartLegendPosition.Left, false); // Добавить легенду слева    // Добавить ось RUB  var series = pieChart.AddSeries("RUB");    // Привязать данные по категории Category и значению Expenses  series.Bind(rub, "Category", "Expenses");    // Вставить диаграмму в документ  pieChart.Insert();    // Сохранить документ  document.SaveAs("document.docx");  }  }    internal class ChartData  {  public string Category { get; set; }  public double Expenses { get; set; }    public static List<ChartData> CreateRubExpenses()  {  return new List<ChartData>  {  new() { Category = "Food", Expenses = 100 },  new() { Category = "Housing", Expenses = 120 },  new() { Category = "Transportation", Expenses = 140 },  new() { Category = "Health Care", Expenses = 150 }  };  }  } |

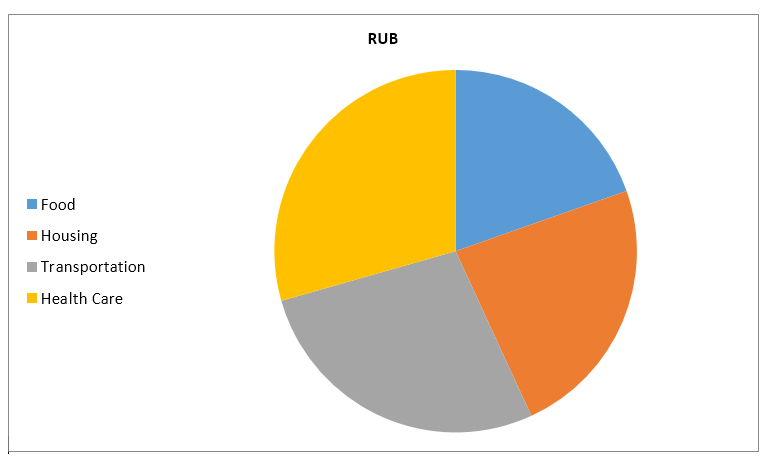


Рисунок 19 – Результат. Круговая

## PDF

Для выгрузки печатной формы (на основе шаблона с расширением .xlsx) в формате pdf необходимо изменить расширение выгружаемого файла в Дизайнере или в конфигурационном файле «(ПФ).НаименованиеФормы.xml».

Изменение формата в дизайнере: в поле «Формат выгрузки» необходимо выбрать «pdf» (Рисунок 20).

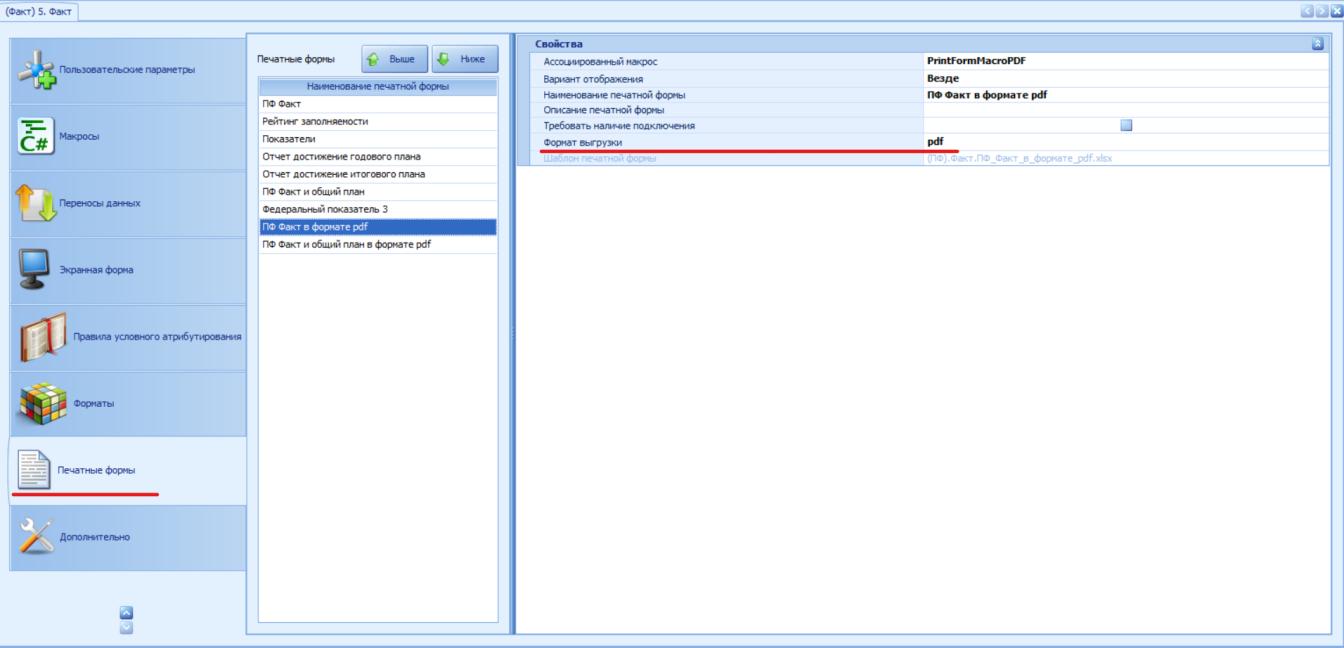


Рисунок 20 – Изменение формата в дизайнере

В конфигурационном файле изменить значение параметра: OutputFormat="pdf" (Рисунок 21).

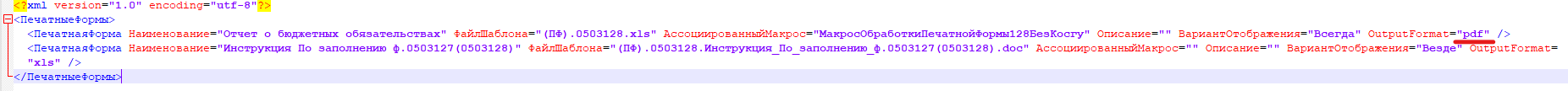


Рисунок 21 – Параметр OutputFormat

Если требуется макросом сформировать и выгрузить файл в формате .pdf, необходимо сначала сформировать и заполнить файл в формате .xlsx (см. п. 12.1), а затем конвертировать его в формат .pdf.

Формирование и выгрузка pdf с помощью обработки:

|  |
| --- |
| [АтрибутПунктаОбработки("Тестовый Лист ПДФ")]  public void PrintPdf()  {  var directory = Path.GetTempPath();  var fileName = "fileName.pdf";  var path = Path.Combine(directory, fileName);    //Создание исходного xlsx файла  var openXmlDocument = КонтейнерЗависимостей.Текущий.Resolve<IOpenXmlWorkbook>();  openXmlDocument.Create(path);    //Заполнение xlsx файла необходимыми данными  var sheet = openXmlDocument.ActiveSheet;  sheet.SetText(1, 1, "TestPdfText");  openXmlDocument.Save();    //Получение сервиса для работы с excel  var excelReport = КонтейнерЗависимостей.Текущий.Resolve<ИнтерфейсВзаимодействияСExcel>(new Arguments() { { "путьКДокументу", path } });  //Конвертирование в pdf  excelReport.КонвертироватьВPdf(path);    РезультатВыполненияОперацииФайл = new РезультатВыполненияОперацииФайл  {  ПотокФайла = new FileStream(path, FileMode.Open),  ИмяФайла = fileName  };  } |

## Параметры отчета

Одним из входных параметров метода обработки печатных форм, указанного в мете, является объект типа «ПараметрыОтчета».Этот объект содержит как информацию о выгружаемом шаблоне печатной формы, так и обо всех секциях/именованных значениях, которые могут подставляться в шаблон самой системой.

Сигнатура метода обработки ПФ в метаструктуре формы (пример):

|  |
| --- |
| public void StandartPrintForm(ref ПараметрыОтчета параметрыОтчета, ОтчетнаяФормаДанных отчетнаяФорма) |

Сам класс имеет следующий вид:

|  |
| --- |
| public class ПараметрыОтчета  {  public ПараметрыОтчета();  public ПараметрыОтчета(string путьКШаблонуПечатнойФормы); //конструкторы объекта    public int КоличествоСтолбцовДляОтображения { get; set; }  public string ПутьКШаблонуПечатнойФормы { get; set; }  public string НаименованиеПечатнойФормы { get; set; }  public ФорматДокументаПечатнойФормы ФорматДокументаПечатнойФормы { get; set; }  public bool ОтменитьПечать { get; set; } //поля с информацией о выгружаемой ПФ    public ПростыеПараметрыОтчета ПростыеПараметрыОтчета { get; set; } //объект содержащий данные из формы (их можно корректировать в коде)    public СтруктурныеПараметрыОтчета СтруктурныеПараметрыОтчета { get; set; } //объект содержащий информацию о секциях в формате xml  } |

### Общая информация о выгружаемой печатной форме

К полям, относящимся к этой категории, можно отнести:

Таблица 27 – Поля категории «Общая информация о выгружаемой печатной форме»

| Наименование поля | Тип | Предназначение |
| --- | --- | --- |
| КоличествоСтолбцовДляОтображения | Целое число | Данное поле служит для ручной установки количества столбцов для печати в случае неправильного определения LastColumn у объекта IWorksheet. |
| ПутьКШаблонуПечатнойФормы | Строка | Содержит полный путь до шаблона печатной формы. По умолчанию путь будет указывать на шаблон ПФ, указанный в xml файле, в метаструктуре формы в КэшФайловФормРазработчика, но путь можно изменить (в случае, например, если необходимо как-либо исправить сам шаблон с помощью OfficeOpenXml).  Пример изменения пути к шаблону:   |  | | --- | | var pathToSave = Path.Combine(Path.GetTempPath(), $"{Guid.NewGuid()}.xlsx"); //новый путь до шаблона ПФ во временной папке    File.Copy(параметрыОтчета.ПутьКШаблонуПечатнойФормы, pathToSave, true); //копирование файла шаблона по новому пути  File.SetAttributes(pathToSave, FileAttributes.Normal); //установка атрибутов файла, для предотвращения возможных ошибок доступа к файлу    параметрыОтчета.ПутьКШаблонуПечатнойФормы = pathToSave; //непосредственное изменение пути к шаблону ПФ |   **Примечание** – В режиме web-приложения меняйте аккуратно, не забывайте, что пользователей, работающих с одним файлом, может быть много. |
| НаименованиеПечатнойФормы | Строка | Содержит имя файла шаблона печатной формы без расширения. |
| ФорматДокументаПечатнойФормы | Enum | Содержит расширение файла шаблона ПФ.   | Расширение | Файл | | --- | --- | | Unknown = 0 | Неизвестный тип файла | | Xls = 1 | Файл Microsoft Excel старого формата (устаревшая версия, больше не используется) | | Xlsx = 2 | Файл Microsoft Excel нового формата | | Docx = 3 | Файл Microsoft Word | | Pdf = 4 | Файл Portable Document Format (PDF) | | Xlsm = 5 | Файл Microsoft Excel с поддержкой Visual Basic скриптов | |
| ОтменитьПечать | Логика | Отменяет дальнейшую обработку печатной формы, но уже после выполнения метода ПФ из метаструктуры. Т.е. если необходимо отменить печать, и не требуется выполнение логики по обработке печатной формы в макросе ПФ из метаструктуры, то после установления в это поле значения *true,*необходимо прервать работу метода с помощью *return*. В результате в приложении будет выведено соответствующее окно с сообщением:  Описание: _scroll_external/attachments/image2022-2-3_10-23-47-1e1828019961a23a806a658b04de0a3107bf87394dbfe4cf1276ddea0e8a796a.png  Сообщение неизменяемо средствами, доступными в классе метаструктуры формы. |

### ПростыеПараметрыОтчета

Объект типа «ПростыеПараметрыОтчета» содержит различную информацию о самой форме и её данных, указанные в этом объекте параметры могут быть подставлены в шаблон печатной формы автоматически, при условии, что шаблон содержит соответствующее объявление поля (об этом подробнее ниже). Основная часть этого поля заполняется системой автоматически, но есть возможность добавить собственные значения.

|  |
| --- |
| public class ПростыеПараметрыОтчета : ИнтерфейсПараметровОтчета  {  public Dictionary<string, object> СписокПараметров { get; set; }    public string ИмяСекции { get; }    public object this[string имяПараметра] { get; set; }    public void ДобавитьПараметр(  string имяПараметра,  object значение,  ТипПолученияСпискаСвойств типПолученияСпискаСвойств);    public void ДобавитьПараметр(string имяПараметра, object значение);    public bool УдалитьПараметр(string имяПараметра);    public void ОчиститьСписокПараметров();    public bool ПараметрСуществуетВСписке(string имяПараметра);    public bool ЗначениеСуществуетВСписке(object значение);  } |

Назначение методов и полей понятно из их названия. Рассмотрим подробнее только «СписокПараметров».

Посмотреть заполнение Системой стандартных полей в «СписокПараметров»у «ПростыхПараметровОтчета» можно в методе «ПодготовитьПараметрыОтчета»,который находится по пути:

**BARS.Svody.DataForm/БазоваяОтчетнаяФормаДанных/ПечатнаяФорма/(Отчет).ПечатнаяФормаОтчета.cs**

Краткий обзор заполняемых Системой категорий (примеры названий, заполняемых сводами, полей указано в кавычках - "..."):

| Категория информации | Подробности |
| --- | --- |
| "ТекущаяДата" / "Report.StoredFormData.FormData\_Id" (Идентификатор данных формы) /  "Report.StoredFormData.IsArc", "Report.StoredFormData.ArchiveDataRecordId" Идентификатор записи истории. | В печатную форму можно подставить текущую дату, без необходимости написания отдельного макроса. |
| Учреждение: "Учреждение.Код", "Учреждение.Наименование", "Учреждение.ИНН",...,  а так же все атрибуты и остальные поля учреждения формы. | Практически все поля из объекта I**Учреждение** доступны в простых параметрах для подстановки в ПФ, отдельного макроса не требуется, если нет никакой дополнительной обработки для этих сведений. |
| Отчетный период формы: "ОтчетныйПериод.Код", "ОтчетныйПериод.ДатаНачала", "ОтчетныйПериод.Квартал". | Так же доступны все поля из **IОтчетныйПериод.** |
| Информация по ЭЦП: "EdsSubjectString", "ЭЦП.действует до", "ЭЦП.выдано", "ЭЦП.алгоритм ключа", "EdsSerialNumber", "ЭЦП.дата подписания". | К данной категории относятся все параметры, начинающиеся с "ЭЦП." и поля "EdsSubjectString" (подробно о подписи) , "EdsSerialNumber" (серийный номер подписи). |
| Пользовательские константы. | Все данные из словаря пользовательских констант добавляются как есть, параметры имеют то же название, что и у самой константы без изменений. |
| Текущий пользователь: "ТекущийПользователь.Имя", "ТекущийПользователь.Логин", "ТекущийПользователь.Роль". | Базовая информация об активном пользователе. |
| Время: "Сейчас" (через точку можно уточнить - .Час/.Минута/.Секунда), "Сегодня" (.День/.Месяц/.Год),  "Сегодня.ДеньНедели". | Текущее серверное время (DateTime.Now). |
| "КомпонентОтчетногоПериода" | Для компонента только его код. |
| Свободные ячейки формы. | Все свободные ячейки формы, указанные в метаструктуре, добавляются по коду. |
| Таблицы формы. | Через переопределение метода *GetPrintFormTableParameters* в метаструктуре формы можно указать для каких таблиц будут собираться параметры или указать значение **IgnoreAllTables**для того чтобы ни одна таблица не попала в параметры.  Учитываются ограничения на элементы, если для таблицы установлено *ЗапретПросмотра*, то заполнится единственный параметр по ней - "ЗапретПросмотра:*\*НаименованиеСкрытойТаблицы\**".  Параметры по динамической таблице заполняются в **СтруктурныеПараметрыОтчета** как **Секция.**  Для статической таблицы параметры заполняются по всем ячейкам, наименование параметров для них будет - "\**КодТаблицы\**:*\*КодСтолбца\** : *\*КодСтроки\*"*(например, "Таблица1:3:001" - ячейка в таблице с кодом "Таблица1" в столбце "3" и строке "001".  Свободные ячейки таблицы заполняются аналогично - "\**КодТаблицы\**:*\*КодСвободнойЯчейки\*"*, так же и с не редактируемыми ячейками. |
| Собственные параметры. | Помимо вышеуказанных параметров, можно добавлять собственные.  Пример добавления собственного простого параметра:   |  | | --- | | параметрыОтчета.ПростыеПараметрыОтчета.ДобавитьПараметр("\_testParam", 123);  //добавление параметра \_testParam со значением 123 |   Объявление заданного параметра в шаблоне:  Описание: _scroll_external/attachments/image2022-2-4_9-36-16-d8374b9a52e14836fbbc10aa9d4845a00f851f49eb905aebcf94efd671313b82.png  **Примечание** – Во избежание возможных коллизий в параметрах, собственные параметры лучше именовать с нижнего подчеркивания, как в примере. |

#### Использование «ПростыеПараметрыОтчета» в шаблонах ПФ

«ПростыеПараметрыОтчета» позволяют добавлять значения в печатные формы непосредственно из системы. Для этого в шаблоне печатной формы необходимо указать наименование параметра из объекта «ПростыеПараметрыОтчета», обрамленное с начала и конца символом $, например:

Укажите в шаблоне некоторые ячейки статической таблицы с кодом «Таблица1», а также выведите наименование учреждения формы в родительном падеже.

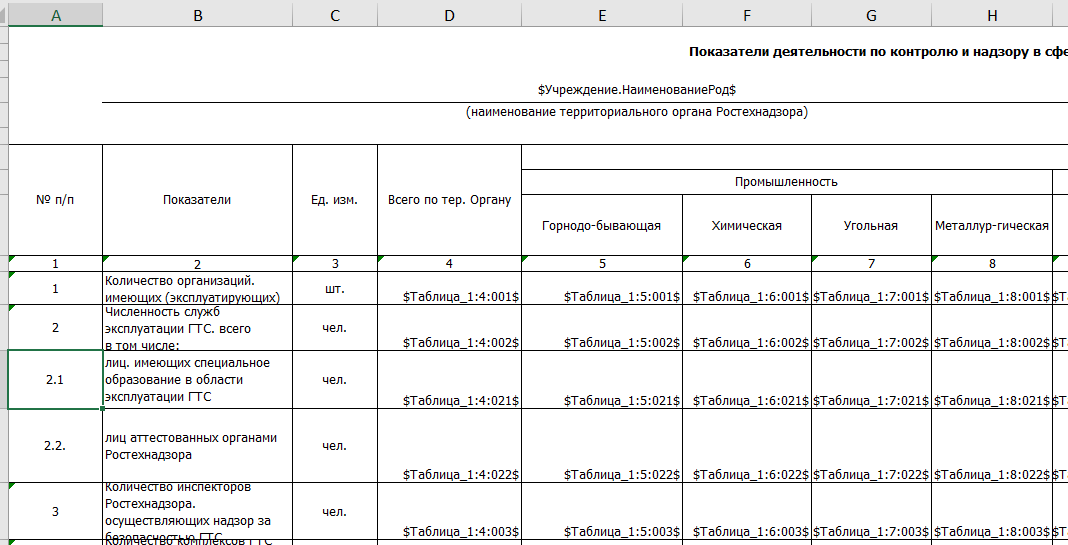


Рисунок 22 – Шаблон печатной формы

Тогда после выгрузки ПФ получите (Рисунок 23, Рисунок 24):

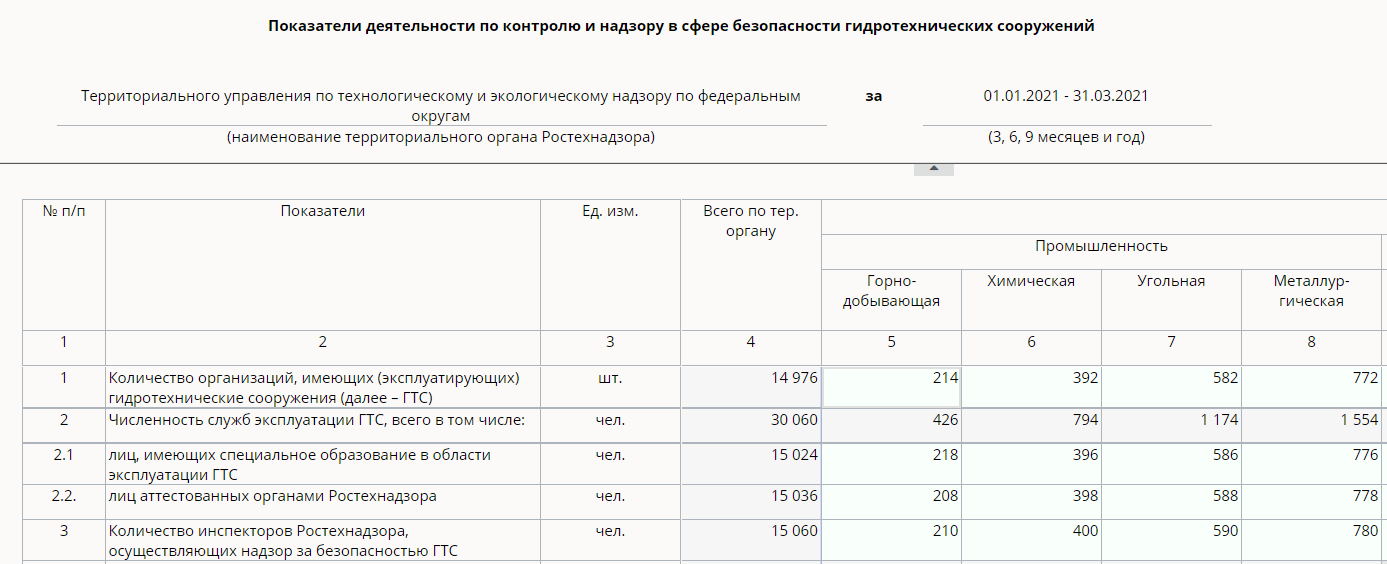


Рисунок 23 – Экранная форма

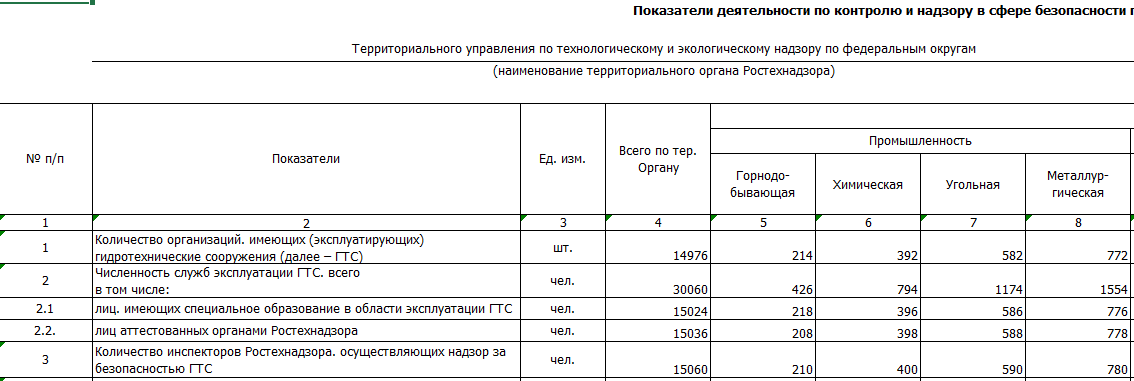


Рисунок 24 – Результат для указанного фрагмента

**Примечание** – Обратите внимание, что в отличие от экранных форм в шаблоне ПФ символ '#' используется эксклюзивно для секций (см. ниже), все параметры указываются через символ '$'.

### СтруктурныеПараметрыОтчета. Секции

Объект типа «СтруктурныеПараметрыОтчета»содержит данные динамических таблиц формы и секций, формируемых в макросах метаструктуры формы. Данные в объекте хранятся в виде XML-документа.

Класс СтруктурныеПараметрыОтчета:

|  |
| --- |
| public class СтруктурныеПараметрыОтчета  {  public XmlDocument СписокПараметров { get; }    public Секция ДобавитьСекцию(string имяСекции);  public void ДобавитьСекцию(Секция секция);  public Секция ВыбратьСекцию(string имяСекции);  public void УдалитьСекцию(string имяСекции);  public bool СодержитСекцию(string имяСекции);    public void Сохранить(string имяФайла); //сохранение xml файла по указанному пути  } |

Секции в печатных формах представляют собой динамически добавляемую часть excel-документа, заполняемую либо макросом, либо самой Системой.

Секция в шаблоне печатной формы задается с помощью символа '#' двумя способами:

1. вертикальная секция – для её задания используется **первая** **строка** шаблона ПФ, в первую ячейку первой строки вносится значение **"###"** (это маркер вертикальной секции, если его не поставить, то будет обрабатываться как горизонтальная), в ячейку после символа **#** вносится название секции, конец секции обязательно начинается со строки **#конец**,обычно после этого в ячейке указывается название обрамляемой секции, но работать будет и без этого:

Таблица 28 – Схема задания вертикальной секции

| ### | #testSection |  | #конец testSection |
| --- | --- | --- | --- |
|  |  | Динамический контент |  |

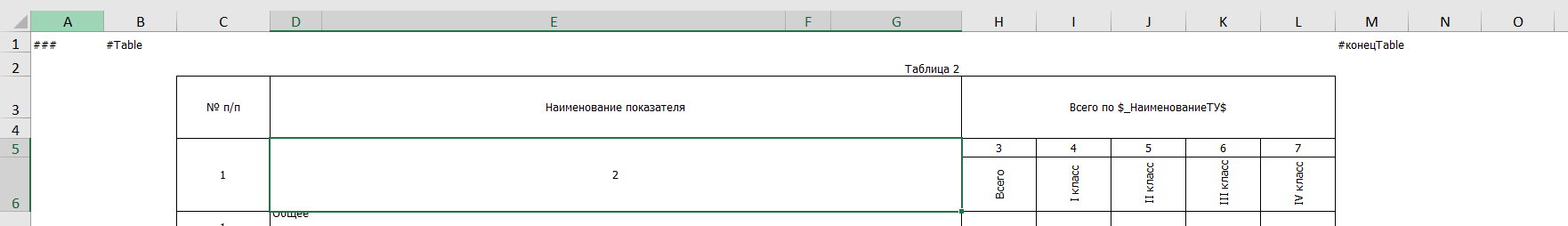


Рисунок 25 – Пример шаблона с вертикальной секцией

1. горизонтальная секция – для её задания используется **первый столбец** шаблона ПФ, название и окончание секции добавляется так же, как и для вертикальной, но по столбцам.

Таблица 29 – Схема задания горизонтальной секции

| #testSection |  |
| --- | --- |
|  | Динамический контент |
| #конец testSection |  |

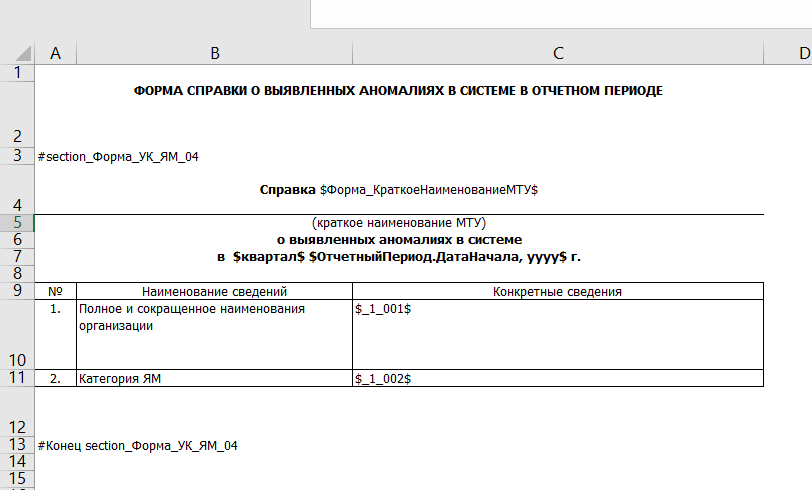


Рисунок 26 – Пример шаблона с горизонтальной секцией

#### Использование СтруктурныеПараметрыОтчета для управления содержанием секций печатных форм

Как и сами «СтруктурныеПараметрыОтчета» представляют собой xml-документ, так и «Секция»является подобным xml-документом, поэтому оба класса имеют похожую структуру.

Класс «Секция»:

|  |
| --- |
| public class Секция : ИнтерфейсПараметровОтчета  {  public Секция();  public string ИмяСекции { get; }    public object this[string имяПараметра] { get; set; }  public object this[string имяПараметра, int номерСтроки] { get; set; }    public void ДобавитьСтроку();    public Секция ВыбратьСекцию(string имяСекции);  public void ВыбратьСтроку(int номерСтроки);  public Секция ДобавитьСекцию(string имяСекции);  public bool СодержитСекцию(string имяСекции);    public int КоличествоПеременныхВСтроке();    public Dictionary<string, object> ПолучитьЗначенияСтроки();    public object[] ПолучитьМассивЗначенийСтроки();    public int КоличествоСтрок();  } |

Физически структурные параметры и секции имеют, к примеру, такой вид   
(Таблица 30):

Таблица 30 – Структурные параметры и секции

| СтруктурныеПараметрыОтчета | Секция |
| --- | --- |
| |  | | --- | | <СтруктурныеПараметры>  <субъекты>  <строка>  <\_субъектырф>Белгородская область</\_субъектырф>  </строка>  </субъекты>  <section\_10000003682>  <строка>  <\_3\_001>1</\_3\_001><\_3\_002>0</\_3\_002><\_3\_003>X</\_3\_003><\_3\_004>0</\_3\_004><\_3\_005>0</\_3\_005><\_3\_006>0</\_3\_006><\_3\_007>0</\_3\_007>  </строка>  </section\_10000003682>  <section\_10000004048>  <строка>  <\_3\_001>1</\_3\_001><\_3\_002>0</\_3\_002><\_3\_003>X</\_3\_003><\_3\_004>0</\_3\_004><\_3\_005>0</\_3\_005><\_3\_006>0</\_3\_006><\_3\_007>0</\_3\_007><\_3\_008>0</\_3\_008>  </строка>  </section\_10000004048>  </СтруктурныеПараметры> | | |  | | --- | | <section\_10000004048>  <строка>  <\_3\_001>1</\_3\_001><\_3\_002>0</\_3\_002><\_3\_003>X</\_3\_003><\_3\_004>0</\_3\_004>  <\_3\_005>0</\_3\_005><\_3\_006>0</\_3\_006><\_3\_007>0</\_3\_007><\_3\_008>0</\_3\_008>  </строка>  </section\_10000004048> | |

То есть отдельная секция является частью общего xml-документа «СтруктурныеПараметры».

Прежде чем начать работу с секцией, её необходимо проинициализировать в «СтруктурныеПараметрыОтчета», при этом необязательно, чтобы шаблон ПФ содержал разметку для секции, а заполнение данных в шаблон происходило уже после выполнения макроса ПФ в метаструктуре (при его наличии). Если ПФ не содержит такой разметки или же содержит объявления иных, чем в структурных параметрах, секций, то данные из «СтруктурныеПараметрыОтчета» не отразятся в итоговой печатной форме.

Что может содержать секция:

* Параметры – Аналогично по принципу работы из «ПростыеПараметрыОтчета»*,* объявления параметров в шаблоне внутри секции позволяет заменять их значениями из «СтруктурныеПараметрыОтчета»*.* Для этого необходимо в шаблоне внутри динамического контента в секции объявить параметр через символ $. Например:

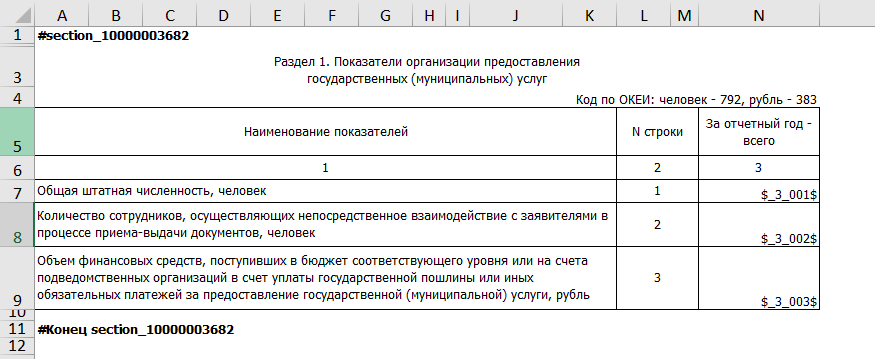


Рисунок 27 – Секции с параметрами в шаблоне ПФ

Пример заполнения секции:

|  |
| --- |
| var section = параметрыОтчета.СтруктурныеПараметрыОтчета.ДобавитьСекцию("section\_10000003682");  section.ДобавитьСтроку();  section["\_3\_001"] = "this";  section["\_3\_002"] = "is testValue";  section["\_3\_003"] = 123; |

Для каждой новой строки секции (содержания между началом и концом секции) необходимо вызывать метод у соответствующего объекта типа **Секция** - *ДобавитьСтроку*. Далее запишите значение через итератор, но название параметра пишется уже без символа $.

Если «СтруктурныеПараметры» не содержат объявления параметров из шаблона ПФ, но в «ПростыеПараметры» есть такие, то будут использоваться последние.

* Другие секции – механизм секции позволяет строить иерархические структуры в печатных формах. Как было указано ранее, раз секция – это xml-документ, то значит одну секцию можно встроить в другую с сохранением последовательности.

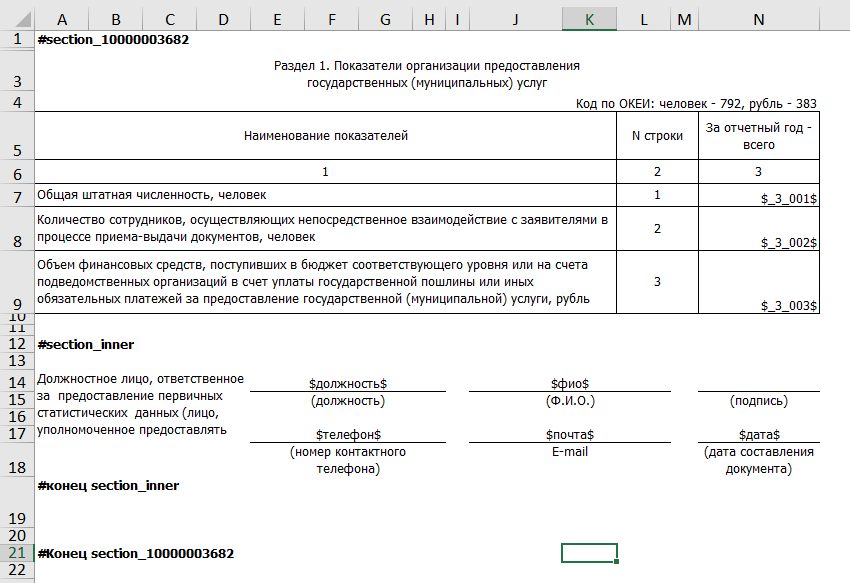


Рисунок 28 – Секция внутри другой в шаблоне ПФ

Заполнение в коде:

|  |
| --- |
| var outer\_section = параметрыОтчета.СтруктурныеПараметрыОтчета.ДобавитьСекцию("section\_10000003682");  outer\_section.ДобавитьСтроку();    var section\_inner = outer\_section.ДобавитьСекцию("section\_inner");  section\_inner.ДобавитьСтроку();    section\_inner["должность"] = "this is";  section\_inner["фио"] = "inner section text";    outer\_section["\_3\_001"] = "this";  outer\_section["\_3\_002"] = "is testValue";  outer\_section["\_3\_003"] = 123; |

## Атрибуты

### Добавление ПФ через Дизайнер

Добавить печатную форму можно в пункте «Меню/ Печатные Формы».

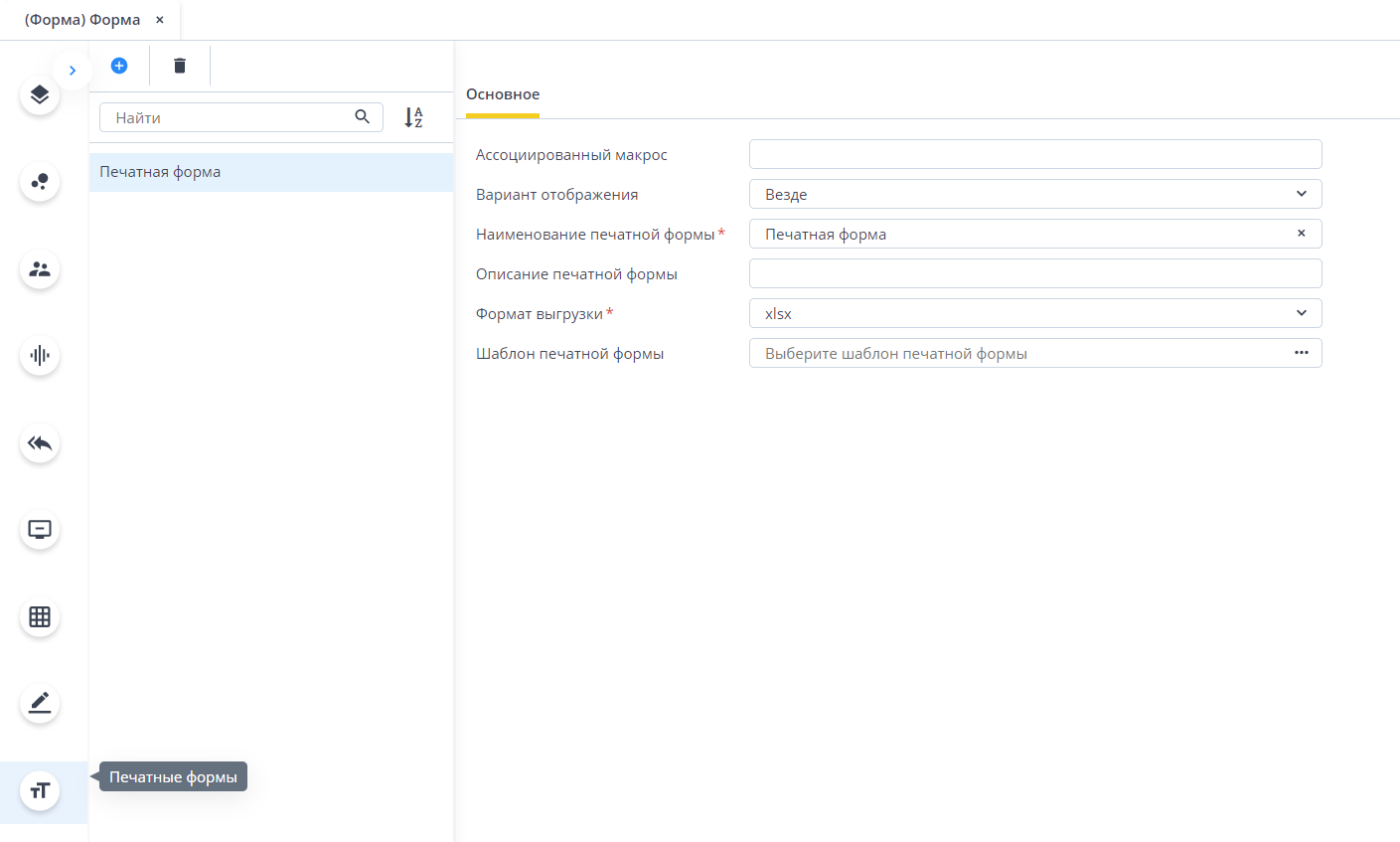


Рисунок 29 – Добавление ПФ через Дизайнер

При создании можно задать атрибуты:

* Ассоциированный макрос – название метода, используемое создаваемой печатной формой;
* Вариант отображения:
  + Везде;
  + Только для центральных офисов;
  + Только для офисов;
  + Только для центральных офисов и офисов;
  + Только для абонентов;
  + Только для центральных офисов и абонентов;
  + Только для офисов и абонентов;
  + Только для пассивных абонентов;
  + Только для центральных офисов и пассивных абонентов;
  + Только для офисов и пассивных абонентов;
  + Только для абонентов и пассивных абонентов;
  + Только для центральных офисов, абонентов и пассивных абонентов;
  + Только для офисов, абонентов и пассивных абонентов.
* Наименование печатной формы – отображаемой наименование ПФ в меню формы;
* Описание печатной формы;
* Формат выгрузки – доступны форматы:
  + .pdf;
  + .xls;
  + .xlsx;
  + .doc;
  + .docx;
  + .xlsm.
* Шаблон печатной формы – выбор шаблона печатной формы. В открывшемся окне «Загрузка печатной формы» нажмите на кнопку «Выбрать файл» и загрузите файл формата .xlsx/.xlsm/.docx, затем нажмитекнопку «Загрузить».

Дизайнер запишет все изменения в файл в папке формы (ПФ).Форма.xml, где Форма – название формы, то же самое можно сделать самим, сделав правки прямо в файле. Пример файла (ПФ).Форма.xml:

|  |
| --- |
| <ПечатныеФормы>  <ПечатнаяФорма Наименование="Печать формы (стандартная)" ФайлШаблона="(ПФ).Форма.Печатная\_форма\_(стандартная).xlsx" АссоциированныйМакрос="DefaultPrintForm" Описание="" ВариантОтображения="Везде" OutputFormat="xlsx" />  <ПечатнаяФорма Наименование="Печать формы (отдельно)" ФайлШаблона="(ПФ).Форма.Печатная\_форма\_(отдельно).xlsx" АссоциированныйМакрос="PrintForm" Описание="" ВариантОтображения="Только для центральных офисов" OutputFormat="xlsx" />  </ПечатныеФормы> |

### Навешанные атрибуты на метод

Таблица 31 – АтрибутПечатнойФормы

| Поле | Тип | Предназначение |
| --- | --- | --- |
| НаименованиеПечатнойФормы | string | Наименование печатной формы |
| НаименованиеШаблонаПечатнойФормы | string | Наименование шаблона печатной формы |
| ПолныйПутьКШаблону | string | Полное наименование шаблона печатной формы |
| ПорядокСортировки | int | Порядок сортировки |
| FormMenuActionVisibility | FormActionVisibilityType | Вариант отображения печатной формы |
| Kind | byte | Тип атрибута, добавлено для обеспечения совместимости с доработками по взаимодействию с атрибутами, определенными в коде Api |

Навешивание атрибута на метод выглядит следующим образом:

|  |
| --- |
| [АтрибутПечатнойФормы("Печать формы (стандартная)", "(ПФ).Форма.Печатная\_форма\_(стандартная).xlsx")]  public void МакросПФ(ref Отчеты.ПараметрыОтчета параметрыОтчета, ОтчетнаяФормаДанных отчетнаяФорма)  {  } |

Предусмотрена возможность использовать и другие параметры. Существуют перегрузки:

|  |
| --- |
| public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы);  public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы, FormActionVisibilityType printReportVisibility);  public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы, int пордядокСортировки);  public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы, int пордядокСортировки, FormActionVisibilityType printReportVisibility);  public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы, string полныйПутьКШаблону, int пордядокСортировки);  public АтрибутПечатнойФормы(string наименованиеПечатнойФормы, string наименованиеШаблонаПечатнойФормы, string полныйПутьКШаблону, int пордядокСортировки, FormActionVisibilityType printReportVisibility); |

# Работа с данными об ЭП

Информация об ЭП в хранимых данных:

|  |
| --- |
| ХранимыеДанныеФормы хранимыеДанныеФормы = ХранилищеДанныхФорм.ПолучитьДанныеФормы(Форма.Идентификатор);  if(хранимыеДанныеФормы.HasSign)  {  Dictionary<string, string>[] свойства = хранимыеДанныеФормы.СвойстваЭЦП();  } |

В данном случае будут получены следующие поля (Рисунок 30).

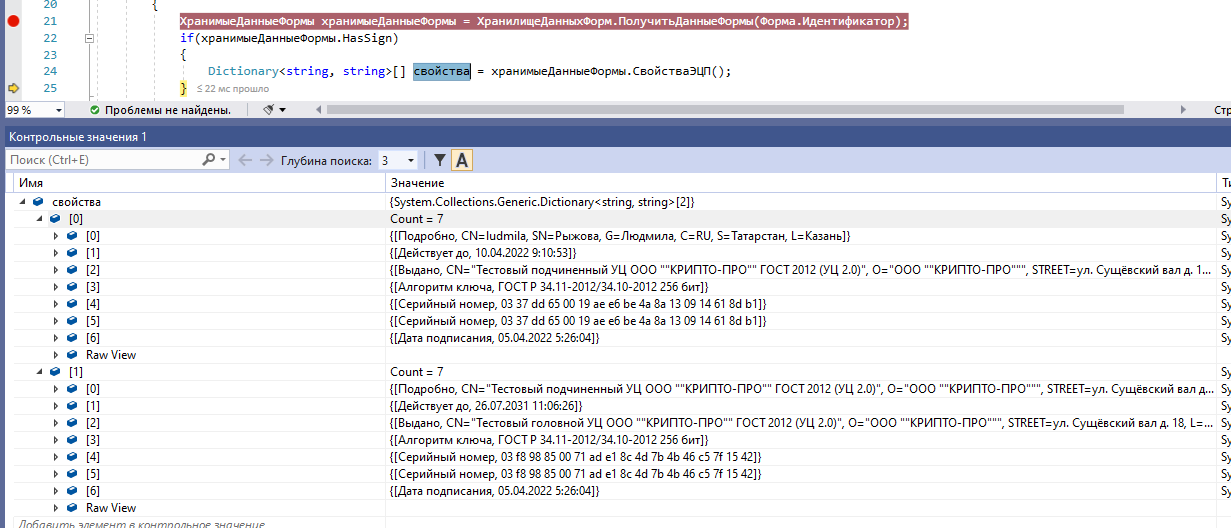


Рисунок 30 – Поля об ЭП в хранимых данных

Методы и свойства в классе «ХранимыеДанныеФормы» для работы с подписью (Таблица 32).

Таблица 32 – Методы и свойства в классе «ХранимыеДанныеФормы» для работы с подписью

| Название | Тип | Описание |
| --- | --- | --- |
| HasSign | bool | Признак наличия ЭЦП |
| ReadCertificates() | IEnumerable<Dictionary<string, string>> | Прочитать сертификаты из БД с проверкой поля на пустоту |
| ПодписатьЭЦП(bool СоПодпись) | bool | Метод сохраняет подпись ЭЦП в текущий объект данных форм |
| ПроверитьСертификат() | List<X509ChainStatus> | Метод проверяет сертификат ЭЦП в ХО ХранимыеДанныеФормы |
| ПроверитьСертификат(byte[] подпись) | List<X509ChainStatus> | Метод проверяет сертификат ЭЦП |
| ПроверитьЭЦП() | bool | Метод проверяет корректность установленных подписей формы |
| СвойстваЭЦП() | Dictionary<string, string>[] | Получает свойства всех подписей формы. Возвращает  Массив словарей, в котором каждый элемент - описание одной ЭЦП |
| СвойстваЭЦП(Guid arcId) | Dictionary<string, string>[] | Получает свойства всех подписей формы по идентификатору архивной записи |
| СформироватьПодпись(bool СоПодпись) | bool | Метод формирует ЭЦП для текущей формы из сертификатов, установленных на компьютер текущего пользователя |
| СформироватьПодпись(bool СоПодпись, X509Certificate2 сертификат) | bool | Метод формирует ЭЦП для текущей формы по переданному объекту сертификата |
| УдалитьЭЦП(bool addLogRecord = true) | bool | Метод удаляет подпись ЭЦП из текущего объекта данных форм |

Также информацию о подписи можно получить через отчетную форму данных. При этом полученный результат будет отличаться от примера выше:

|  |
| --- |
| ОтчетнаяФормаДанных отчетнаяФорма = ДополнительныеМетодыПолученияДанных.ПолучитьОтчетнуюФорму("КодФормы",  Форма.Идентификатор.Учреждение, Форма.Идентификатор.КомпонентОтчетногоПериода, Форма.Идентификатор.ЭлементЦепочки);    IEnumerable<Dictionary<string, string>> свойства2 = отчетнаяФорма.GetSignData(); |

В данном случае будут получены следующие поля (Рисунок 31):

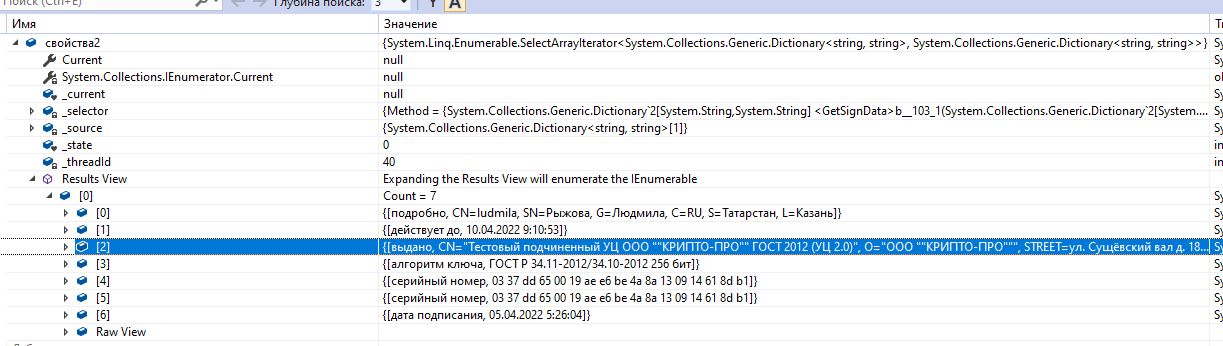


Рисунок 31 – Поля, полученные через отчетную форму данных

Получить объект подписи можно через IDataStore, класс с подписью DigitalSignature:

|  |
| --- |
| IDataStore \_dataStore = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>();  List<DigitalSignature> listDigitalSignature = \_dataStore.GetQuery<DigitalSignature>()  .Where(sign => sign.StoredFormDataИдентификатор == хранимыеДанныеФормы.Идентификатор)  .ToList(); |

Таблица 33 – Свойства в классе DigitalSignature

| Название | Тип | Описание |
| --- | --- | --- |
| Alghoritm | string | Алгоритм, используемый для создания подписи |
| IssuerName | string | Данные о сертификационном центре, выдавшем ЭП |
| SerialNumber | string | Серийный номер сертификата |
| SignatureDate | DateTime | Дата и время подписания отчетной формы |
| SubjectName | string | Кому выдан сертификат |
| StoredFormData | ХранимыеДанныеФормы | Данные отчетной формы |
| StoredFormDataИдентификатор | Guid | Идентификатор данных отчетной формы |
| Type | SignatureType | Тип подписи |
| ValidUntil | DateTime | Срок действия сертификата |

SignatureType – перечисление, допустимые значения: Sign (подпись) и CoSign (соподпись).

# Работа с субтаблицами (сводная, ПФ)

## ПФ для субтаблиц

Работа с ПФ субтаблиц начинается с создания шаблона в формате .xlsx, при этом желательно, чтобы название секции не совпадало с кодом таблицы, во избежание автоматического заполнения.

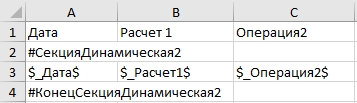


Рисунок 32 – Шаблон

Выгрузка данных субтаблиц в ПФ осуществляется при помощи макроса. Для того чтобы макрос отработал, необходимо имя макроса указать в xml-файле печатной формы в поле АссоциированныйМакрос.

Содержимое файла (ПФ).Форма.ПФ.xlsx:

|  |
| --- |
| <ПечатныеФормы>  <ПечатнаяФорма Наименование="PechatForm" ФайлШаблона="(ПФ).1.PechatForm.xlsx" АссоциированныйМакрос="МакросПФ" Описание="" ВариантОтображения="Везде" OutputFormat="xlsx" />  </ПечатныеФормы> |

В коде у макроса должны быть указаны параметры: ref ПараметрыОтчета параметрыОтчета, ОтчетнаяФормаДанных отчетнаяФорма.

Работа с ПФ осуществляется через секции. Поэтому для начала необходимо объявить секцию, далее циклом перебрать все таблицы и проверить, является ли таблица субтаблицей при помощи свойства IsSubTable. Если является, то перебрать все строки субтаблицы, заполняя ПФ. Для заполнения ПФ необходимо добавить строку в секцию, после чего, обращаясь по коду столбца к столбцу ПФ, заполнять ПФ значениями из субтаблицы.

Метод «МакросПФ»:

|  |
| --- |
| public void МакросПФ(ref ПараметрыОтчета параметрыОтчета, ОтчетнаяФормаДанных отчетнаяФорма)  {  Секция секция = параметрыОтчета.СтруктурныеПараметрыОтчета.ДобавитьСекцию("СекцияДинамическая2");  foreach (var table in Форма.Таблицы.Values)  {  if (table.IsSubTable)  {  for (int j = 0; j < table.ПолучитьКоличествоСтрок(); j++)  {  секция.ДобавитьСтроку();  секция["\_Дата"] = table.МатрицаЗначений[j, "Дата"].Значение;  секция["\_Расчет1"] = table.МатрицаЗначений[j, "Расчет1"].Значение;  секция["\_Операция2"] = table.МатрицаЗначений[j, "Операция2"].Значение;  }  }  }  } |

## Сводная для субтаблиц

В версии Системы 5.1 реализована стандартная сводная для субтаблиц в Системе. Однако при необходимости можно реализовать сборку сводной макросом.

Так как субтаблица открывается только по нажатию кнопки, при сборке сводной необходимо создать субтаблицу на основе исходной субтаблицы и заполнить ее значениями.

Для создания таблицы используется метод CreateModalTable класса БазоваяОтчетнаяФормаДанных, который принимает на вход следующие параметры:

* Код родительской таблицы;
* Код строки родительской таблицы;
* Код столбца родительской таблицы;
* Код субтаблицы.

При этом во избежание повторного создания одной и той же субтаблицы необходимо проверить, не была ли она создана ранее.

После создания субтаблицы она заполняется значениями из исходных субтаблиц. При этом ранее полученные значения субтаблиц суммируются с вновь полученной субтаблицей.

Метод создания субтаблицы:

|  |
| --- |
| private void MoveModalTableData(ЯчейкаДанных cell, string rowCode, ДанныеФормы sourceForm, ДанныеФормы receiverForm, СтрокаДанных receiverRow, bool newString)  {  var history = new List<HistoryCell>();  var sourceTableCode = cell.Значение.ЗначениеСтрокой;  var columnCode = cell.КодЯчейки;  var templateTableCode = sourceTableCode.Split('\_').FirstOrDefault();  var sourceTable = sourceForm[sourceTableCode];    if (sourceTable != null)  {  ТаблицаДанных receiver;    if (newString)  {  receiver = СводнаяФорма.CreateModalTable("ВидЛицензирования", rowCode, cell.КодЯчейки, templateTableCode);  }  else  {  var modalTableCode = receiverRow[columnCode].ЗначениеСтрокой;  receiver = receiverForm[modalTableCode];  }    if (receiver != null)  {  foreach (var row in sourceTable.МатрицаЗначений.Строки.Values)  {  foreach (var col in row.ТаблицаДанных.СтолбцыМетаструктуры.Values)  {  var colCode = col.Код;  var historyCell = history.FirstOrDefault(h => h.RowCode == row.КодСтроки && h.ColumnCode == colCode);  var value = row[colCode].ЗначениеЧисло;    receiver[row.КодСтроки, col.Код].ЗначениеЧисло += value;    if (historyCell != null)  {  historyCell.Value += value;  }  else  {  history.Add(  new HistoryCell  {  ColumnCode = colCode, RowCode = row.КодСтроки,  Value = value  });  }  }  }    AddNewRecordHistory(receiver, history);  }  }  } |

После суммирования значений необходимо записать в историю данные субтаблицы и ее значения для последующего сложения.

Метод добавление записи в историю:

|  |
| --- |
| private void AddNewRecordHistory(ТаблицаДанных receiverTable, List<HistoryCell> history)  {  foreach (var receiverRow in receiverTable.МатрицаЗначений.Строки.Values)  {  var historyRows = history.Where(h => h.RowCode == receiverRow.КодСтроки).ToList();    var recordHistory = new ЗаписьИсторииСборкиИтоговыхОтчетов();    recordHistory.КодТаблицыИсточника = "";  recordHistory.КодТаблицыПриемника = receiverTable.КодТаблицы;  recordHistory.КодСтрокиИсточника = receiverRow.КодСтроки;  recordHistory.КодСтрокиПриемника = receiverRow.КодСтроки;  recordHistory.НаименованиеУчрежденияИсточника = ИсторияСборки.НаименованиеУчрежденияИсточника;  recordHistory.НаименованиеУчрежденияПриемника = ИсторияСборки.НаименованиеУчрежденияПриемника;  recordHistory.ИдентификаторДанныхИсточника = ИсторияСборки.ИдентификаторДанныхИсточника;  recordHistory.ИдентификаторДанныхПриемника = ИсторияСборки.ИдентификаторДанныхПриемника;  recordHistory.НаименованиеФормыИсточника = ИсторияСборки.НаименованиеФормыИсточника;  recordHistory.НаименованиеФормыПриемника = ИсторияСборки.НаименованиеФормыПриемника;    foreach (var historyRow in historyRows)  {  var columns = $"{historyRow.ColumnCode}{ИсторияСборкиИтоговыхОтчетов.РазделительПолейНовый}{historyRow.ColumnCode}";    if (!recordHistory.Значения.ContainsKey(columns))  {  recordHistory[columns] = historyRow.Value;  }  }    \_historyRecordList.Add(recordHistory);  }  } |

Подробную информацию про работу с субтаблицами можно найти в п. 14.3.

## Модальные субтаблицы

В класс ТаблицаДанных добавлены 2 свойства:

|  |
| --- |
| /// <summary> Координаты ячейки родительской таблицы </summary>  public TableCellCoordinates ParentCell { get; }  /// <summary> Таблица является субтаблицей (модальной) </summary>  public bool IsSubTable { get; } |

Свойства представляют собой структуру с координатами ячейки, из которой была открыта таблица, в случае если эта таблица является модальной субтаблицей.

|  |
| --- |
| namespace BARS.Svody.DataForm.ДанныеФормы  {  /// <summary> Координаты ячейки таблицы </summary>  public readonly struct TableCellCoordinates  {  /// <summary>Код таблицы</summary>  public string TableCode { get; }    /// <summary>Код строки</summary>  public string RowCode { get; }    /// <summary>Код столбца</summary>  public string ColumnCode { get; }    public TableCellCoordinates(string tableCode, string rowCode, string columnCode)  {  TableCode = tableCode;  RowCode = rowCode;  ColumnCode = columnCode;  }    /// <summary> Свойства не заполнены </summary>  /// <returns>true - если структура была инициализирована через конструктор без параметров</returns>  internal bool IsEmpty()  {  return TableCode == null &&  RowCode == null &&  ColumnCode == null;  }  }  } |

В прикладном коде, работающем с модальными субтаблицами, необходимо проверять, является ли субтаблица модальной с помощью свойства IsSubtable, а затем можно работать со свойством ParentCell.

|  |
| --- |
| var table = Форма["Table1"];  if (table.IsSubTable)  {  var parentTable = Форма[table.ParentCell.TableCode];  var parentRow = parentTable[table.ParentCell.RowCode]  var parentCell = parentTable[table.ParentCell.RowCode, table.ParentCell.ColumnCode];    // и т.д.  } |

# Увязки

В Системе можно формировать различного рода увязки. Сама по себе увязка это сравнение данных текущей формы по какому-либо правилу.

Существует 3 типа увязок:

* Внутриформенные увязки – это увязки, которые проверяются в рамках одной формы;
* Межформенные увязки – это увязки, которые формируются при проверке текущей формы с любыми другими по заданной логике;
* Внутривкладочные увязки – это внутриформенные увязки в которых все проверяется повкладочно (рассматриваются доступные вкладки – таблицы).

Сформировать увязки для проверки можно через Дизайнер и через макрос. В Дизайнере добавляются стандартные и функциональные увязки. О реализации новых функций функциональных увязок написано в п. 28.18 настоящей инструкции.

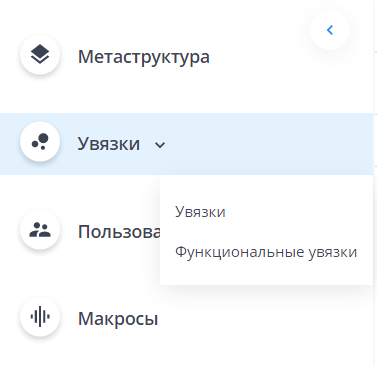


Рисунок 33 – Переход в раздел «Увязки» в Дизайнере

Также есть возможность сформировать увязки в приложении в консоли администрирования, пункт «Пользовательские увязки».

Существует несколько пользовательских параметров, при которых можно обработать события формирования увязок, добавить свои параметры.

## Внутриформенные увязки

Для того чтобы реализовать проверку внутриформенных увязок через макрос, необходимо переопределить базовый обработчик формы.

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияВнутреннихУвязок()  {  List<ОбъектУвязки> результат = new List<ОбъектУвязки>();  return результат;  } |

Макрос возвращает Список Объектов увязок.

Таблица 34 – Объект Увязок

| Поле | Тип | Предназначение |
| --- | --- | --- |
| ТипОперацииУвязки | Enum | Перечисление возможных типов операций, возникающих при проверке увязок (равенство, больше/меньше, не равно) |
| ТипОшибкиПроверкиУвязки | Enum | Перечисление возможных типов ошибок, возникающих при проверке увязок (Ошибка заполнения формы, предупреждение и т.д.) |
| Наименование | string | Задает наименование увязки |
| Сообщение | string | Задает сообщение об ошибке |
| ПериодыДействияУвязок | List<string> | Задает свойство, описывающее в каких периодах необходимо проверять увязку (месячный, годовой, полугодовой...). Если пустое, то проверяет на всех типах |
| УвязкаКорректна | bool | Указывается, является ли увязка корректной, создано для исключений. Если true - то увязка не отобразится в итоге. |
| НеПроверятьНезаполненыеДанные | bool | Если true - игнорируются нулевые значения, если false - иначе. |
| Расхождение | decimal | Если данные некорректны, то следовательно есть расхождения, это поле хранит непосредственно само расхождение. |
| РасшифровкаЗначений | string | Описание, в какой форме, каких таблицах, строках и ячейках была допущена ошибка при проверке. |

Пример:

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияВнутреннихУвязок()  {  var результат = new List<ОбъектУвязки>();  var formCode = Форма.Идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания;    var firstValue = Форма[кодТаблицы][кодСтроки1, кодСтолбца1].ЗначениеЧисло;  var secondValue = Форма[кодТаблицы][кодСтроки2, кодСтолбца2].ЗначениеЧисло;  if (firstValue < secondValue)  {  var объектУвязки = new ОбъектУвязки(  "Наименование увязки",  "Сообщение увязки",  ТипОшибкиПроверкиУвязки.ОшибкаЗаполненияФормы,  ТипОперацииУвязки.БольшеИлиРавно,  firstValue - secondValue,  "\n\tФорма:" + formCode +  ";Таблица:" + кодТаблицы +  ";Строка:" + кодСтроки1 +  ";Столбец:" + кодСтолбца1 +  ";Значение:" + value +  "\n\tФорма:" + formCode +  ";Таблица:" + кодТаблицы +  ";Строка:" + кодСтроки2 +  ";Столбец:" + кодСтолбца2 +  ";Значение:" + secondValue,  false);  результат.Add(объектУвязки);  }  return результат;  } |

Можно не указывать все вышеперечисленные поля для формирования увязки, существуют перегрузки.

Возможные перегрузки ОбъектаУвязки:

|  |
| --- |
| ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, string Сообщение, bool УвязкаКорректна);  ОбъектУвязки (string Наименование, string Сообщение, ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, object Расхождение, IEnumerable<TranscriptDto> transcriptions);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений);  ОбъектУвязки (string Наименование, string Сообщение, ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, ТипОперацииУвязки operationType, object Расхождение, IEnumerable<TranscriptDto> transcriptions);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, string Сообщение, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, string Сообщение, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок);  ОбъектУвязки (string Наименование, string Сообщение, ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, ТипОперацииУвязки ТипОперации, object Расхождение, string РасшифровкаЗначений, bool увязкаКорректна);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок, bool неПроверятьНезаполнДанные);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, string Сообщение, ТипОперацииУвязки ТипОперации, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок, bool неПроверятьНезаполнДанные);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, ТипОперацииУвязки ТипОперации, ТипОперацииУвязки ТипОперацииПроверкиНеобходимостиПроверки, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок, bool неПроверятьНезаполнДанные);  ОбъектУвязки (ТипОшибкиПроверкиУвязки ТипОшибкиПроверки, string Наименование, string Сообщение, ТипОперацииУвязки ТипОперации, ТипОперацииУвязки ТипОперацииПроверкиНеобходимостиПроверки, Dictionary<string, object> СписокПеременных, string РасшифровкаЗначений, string ПериодыУвязок, bool неПроверятьНезаполнДанные); |

После того как формируются все Объекты увязок, формируется окно с результатом (Рисунок 34).

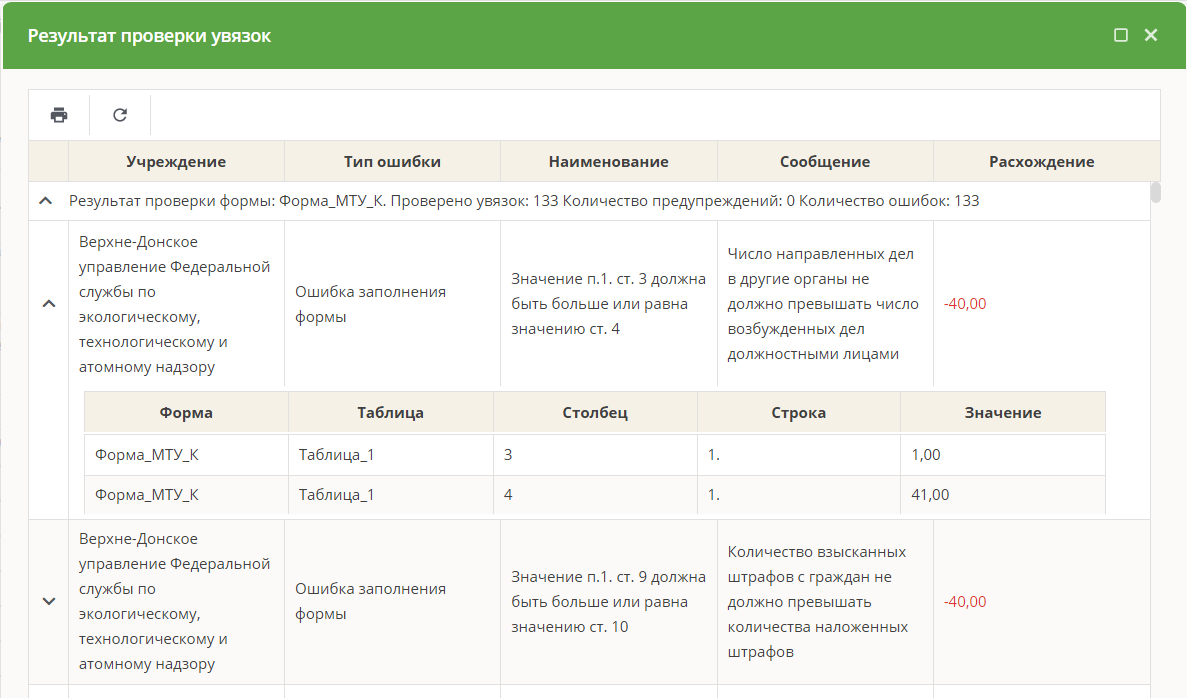


Рисунок 34 – Пример результата проверки увязок

Вложенная таблица с колонками: Форма,Таблица, Столбец, Строка и Значение – это и есть Расшифровка значений.

## Межформенные увязки

Для того чтобы реализовать проверку межформенных увязок через макрос, необходимо переопределить базовый обработчик формы:

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияМежформенныхУвязок()  {  List<ОбъектУвязки> результат = new List<ОбъектУвязки>();  return результат;  } |

Отличие от внутриформенных только в том, что проверяются значения между двумя формами. Отличие формирования Объекты увязок ВФУ в том, что для формирования расшифровки значений МФУ используется TranscriptDto для корректной работы перехода к форме.

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияМежформенныхУвязок()  {  var result= new List<ОбъектУвязки>();  var currentValue = Форма[КодТаблицы][кодСтроки, кодСтолбца].ЗначениеЧисло;  var pastValue = pastForm.Данные[КодТаблицы].МатрицаЗначений[кодСтроки, кодСтолбца].ЗначениеЧисло;  if (pastValue > currentValue)  {  result.Add(new ОбъектУвязки("Наименование увязки",  "Сообщение увязки",  ТипОшибкиПроверкиУвязки.ОшибкаЗаполненияФормы,  ТипОперацииУвязки.БольшеИлиРавно,  currentValue - pastValue,  new TranscriptDto[]  {  new TranscriptDto(Форма.Идентификатор)  {  FormCode = Форма.МетаструктураФормы.Код,  FormText = Форма.МетаструктураФормы.Наименование,  ChainElementType = Форма.Идентификатор.ЭлементЦепочки,  ComponentId = Форма.Идентификатор.КомпонентОтчетногоПериода.ИдентификаторОбъекта,  DepartmentId = Форма.Идентификатор.Учреждение.ИдентификаторОбъекта,  RowCode = кодСтроки,  ColumnCode = кодСтолбца,  TableCode = КодТаблицы,  Value = currentValue  },  new TranscriptDto(Форма.Идентификатор)  {  FormCode = pastForm.Данные.МетаструктураФормы.Код,  ChainElementType = pastForm.ЭлементЦепочки,  FormText = pastForm.Данные.МетаструктураФормы.Наименование,  ComponentId = pastForm.Данные.Идентификатор.КомпонентОтчетногоПериода.ИдентификаторОбъекта,  DepartmentId = pastForm.Данные.Идентификатор.Учреждение.ИдентификаторОбъекта,  RowCode = кодСтроки,  ColumnCode = кодСтолбца,  TableCode = КодТаблицы,  Value = pastValue  }  }));  }  return result;  } |

Пример результата проверки увязок представлен на рисунке ниже (Рисунок 35).

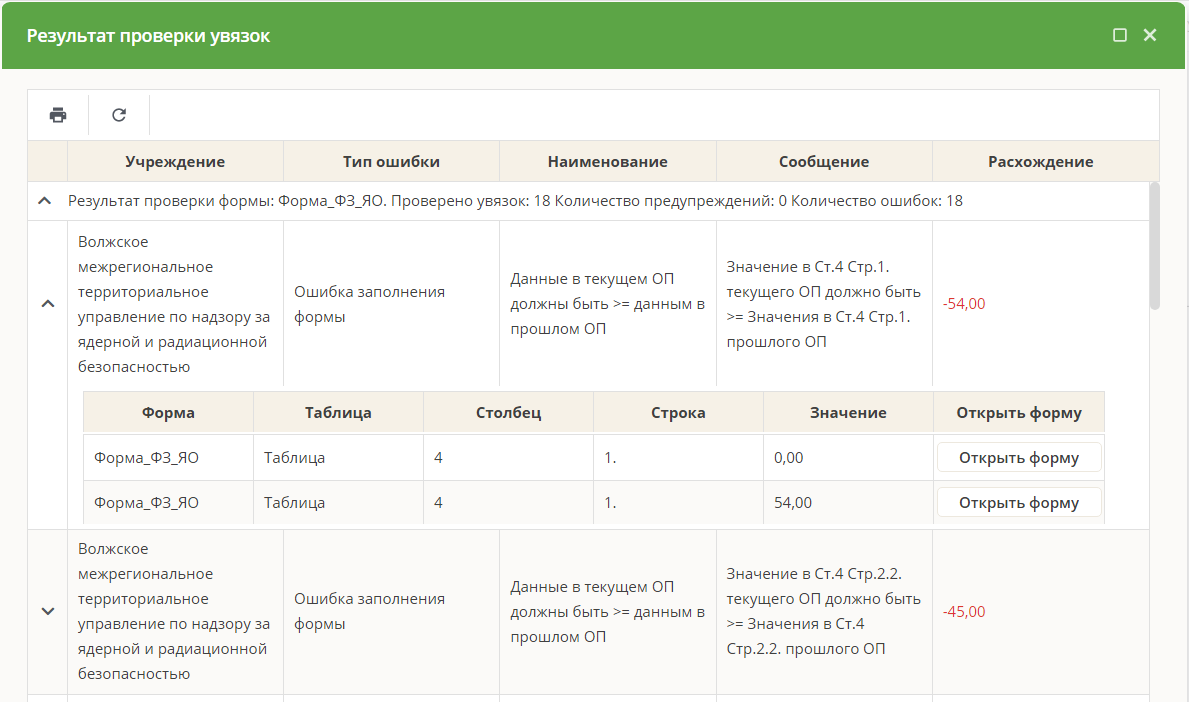


Рисунок 35 – Пример результата проверки увязок

## Внутривкладочные увязки

Для создания обработки внутривкладочных увязок, необходимо переопределить такую же обработку «ОбработатьСобытиеФормированияВнутреннихУвязок»но с аргументами:

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияВнутреннихУвязок(АргументыФормированияВнутриформенныхУвязок аргументы)  {  List<ОбъектУвязки> результат = new List<ОбъектУвязки>();  return результат;  } |

Аргументы: ВыбранныеЗакладки(List<string>).

Список доступных вкладок (кодов таблиц), по которым можно делать сверку.

Создание объектов и вывод осуществляется аналогично внутриформенным увязкам.

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияВнутреннихУвязок(АргументыФормированияВнутриформенныхУвязок аргументы)  {  var результат = new List<ОбъектУвязки>();  var formCode = Форма.Идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания;    if (аргументы.ВыбранныеЗакладки.Contains("КодПроверяемойТаблицы"))  {  var firstValue = Форма[кодТаблицы][кодСтроки1, кодСтолбца1].ЗначениеЧисло;  var secondValue = Форма[кодТаблицы][кодСтроки2, кодСтолбца2].ЗначениеЧисло;  if (firstValue < secondValue)  {  var объектУвязки = new ОбъектУвязки(  "Наименование увязки",  "Сообщение увязки",  ТипОшибкиПроверкиУвязки.ОшибкаЗаполненияФормы,  ТипОперацииУвязки.БольшеИлиРавно,  firstValue - secondValue,  "\n\tФорма:" + formCode +  ";Таблица:" + кодТаблицы +  ";Строка:" + кодСтроки1 +  ";Столбец:" + кодСтолбца1 +  ";Значение:" + value +  "\n\tФорма:" + formCode +  ";Таблица:" + кодТаблицы +  ";Строка:" + кодСтроки2 +  ";Столбец:" + кодСтолбца2 +  ";Значение:" + secondValue,  false);  результат.Add(объектУвязки);  }  }  return результат;  } |

## Формирование Пользовательских параметров

Существует несколько пользовательских параметров, при которых можно обработать события формирования увязок, добавить свои параметры. Подробнее ознакомиться с формированием пользовательских параметров можно в п. 7 настоящей инструкции.

|  |
| --- |
| protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string операция)  {  if (операция.Contains("ОперацияПроверкиВнутреннихУвязок")) // Срабатывает при нажатии на меню => внутриформенные увязки  {  параметры.ДобавитьПараметрФлажок("СУчетомСборкиСводной", "С учетом сборки сводной", false);  }    if (операция.Contains("ОперацияПроверкиМежформенныхУвязок")) //Срабатывает при нажатии на меню => Межформенные увязки  {  if (Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ТипОтчетногоПериода == ТипОтчетногоПериода.Годовой  || Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ТипОтчетногоПериода == ТипОтчетногоПериода.Квартальный)  {  параметры.ДобавитьПараметрВыборСтроковогоЗначенияИзСписка("КомпонентаОтчетногоПериода", "Компонент отчетного периода", списокКомпонентов););  }  }  } |

# Переопределение источника записей справочника

Для переопределения источника записей справочника:

1. с помощью Дизайнера в поле «Обработка формирования источника записей» необходимого столбца установите «флажок» (Рисунок 36):

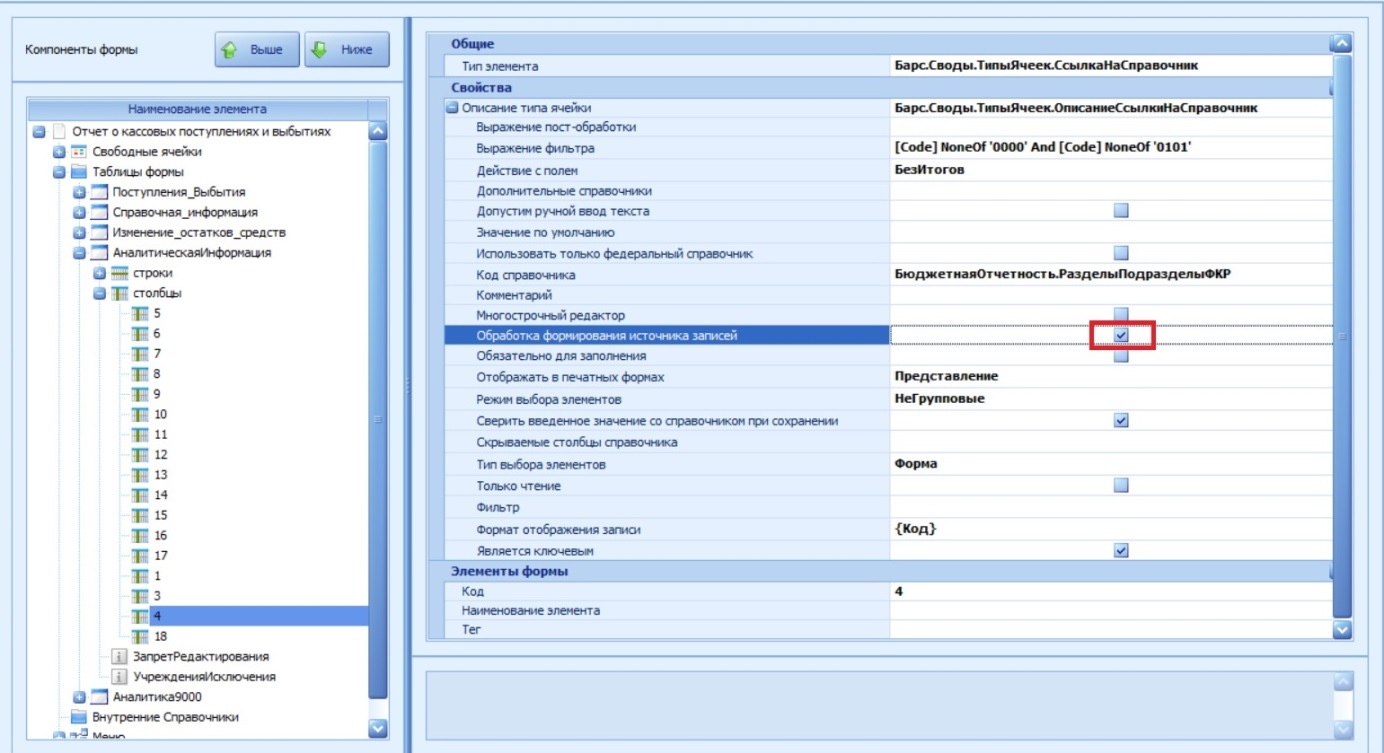


Рисунок 36 – Переопределение источника записей справочника

1. переопределите метод ПолучитьИсточникЗаписейДляВыбораИзСправочника():

|  |
| --- |
| public override object ПолучитьИсточникЗаписейДляВыбораИзСправочника(АргументыСобытияПолученияИсточникаЗаписейДляВыбораИзСправочника аргументы)  {  //Метод вызывается при нажатии на любую ячейку с типом Справочник, поэтому необходимо добавить условие для требуемого столбца  //замена справочника-источника  if (аргументы.КодСтолбца == "КодСтолбцаВКоторомНаходитсяСсылкаНаСправочник" && аргументы.ТаблицаДанных.КодТаблицы == "КодТаблицы")  {  var выборИзСправочника = new ВыборИзУниверсальногоСправочника  {  ОписаниеСсылки = new ОписаниеСсылкиНаСправочник()  {  КодСправочника = "Группа.КодСправочника",  РежимВыбора = РежимыВыбораЭлементовСправочника.ОпределяетсяСправочником,  ТипВыбора = ТипВыбораЭлементовСправочника.ОпределяетсяСправочником,  },  Данные = Форма  };    var новыйСправочник = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник("Группа.КодСправочника");    выборИзСправочника.СписокЗаписей = новыйСправочник .Записи;    return выборИзСправочника;  }    //Замена записей справочника  if (аргументы.КодСтолбца == "КодСтолбцаВКоторомНаходитсяСсылкаНаСправочник2" && аргументы.ТаблицаДанных.КодТаблицы == "КодТаблицы")  {  var выборИзСправочника = new ВыборИзУниверсальногоСправочника  {  ОписаниеСсылки = аргументы.ОписаниеСсылки as ОписаниеСсылкиНаСправочник,  Данные = Форма  };    выборИзСправочника.СписокЗаписей = new List<IЗаписьСправочника>(); //список с необходимымы записями справочника    return выборИзСправочника;  }      //Формирование справочника с помощью DynamicDictionary  if (аргументы.КодСтолбца == "КодСтолбцаВКоторомНаходитсяСсылкаНаСправочник3" && аргументы.ТаблицаДанных.КодТаблицы == "КодТаблицы")  {  var sourceRecords = new List<string>();  var dictionaryEntries = sourceRecords.Select(x => new ЗаписьСправочника() { Код = x, Наименование = x }).ToList<IЗаписьСправочника>();    var dynamicDictionary = new DynamicDictionary();  dynamicDictionary.СформироватьИзСписка(dictionaryEntries);    return dynamicDictionary;  }    return base.ПолучитьИсточникЗаписейДляВыбораИзСправочника(аргументы);  } |

# Запись данных в ОФ

## Через экранную форму

Откройте отчетную форму и введите новые или измените существующие значения (Рисунок 37).

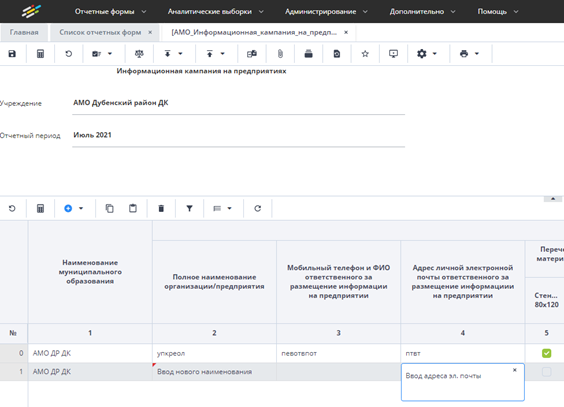


Рисунок 37 – Отчетная форма

## Во время вызова события

Пример ОбработатьСобытиеДоСохраненияФормы:

|  |
| --- |
| public override АргументыСохраненияФормы ОбработатьСобытиеДоСохраненияФормы(АргументыСохраненияФормы аргументы)  {  // Запись данных в ячейки отчетной формы  return аргументы;  } |

Подробнее см. п. 19 настоящей инструкции.

## Через импорт

Пример ВыполнитьИмпорт:

|  |
| --- |
| public List<string> ExecuteImport()  {  var errors = new List<string>();    try  {  // Запись данных в ячейки отчетной формы  }  catch (Exception ex)  {  errors.Add($"Произошла ошибка: {ex.Message}");  }    return errors;  } |

## Через обработку

Пример:

|  |
| --- |
| public void ЗаполнитьЯчейки()  {  // Запись данных в ячейки отчетной формы  } |

Подробнее см. п. 19 настоящей инструкции.

## Через сбор сводной

Таблицы отчетной формы заполняются с помощью вызова сбора сводной.

Подробнее см. п. 4 настоящей инструкции.

## Пример метода для заполнения свободных ячеек и таблиц

|  |
| --- |
| private void FillFormData()  {  //Заполнение свободной ячейки формы  var formFreeCell = Форма.СвободныеЯчейки["КодСвободнойЯчейки"];  formFreeCell.Значение = "заполняемое значение";    //Заполнение ячейки в таблице  var table = Форма["КодТаблицы"].МатрицаЗначений;  table["КодСтроки", "КодСтолбца"].Значение = "заполняемое значение";    //Заполнение свободной ячейки таблицы  var talbeFreeCell = Форма["КодТаблицы"].СвободныеЯчейки["КодСвободнойЯчейки"];  talbeFreeCell.Значение = "заполняемое значение";  }    //Пример как заполнить все ячейки во всех таблицах формы  private void FillAllTables()  {  var tablesDictionary = Форма.МетаструктураФормы.Таблицы;    foreach (var (tableCode, tableMeta) in tablesDictionary)  {  var tableColumns = tableMeta.Столбцы.Keys;  var tableRows = tableMeta.Строки.Keys;    FillTable(tableCode, tableRows, tableColumns, "заполняемое значение");  }  }    private void FillTable(string tableCode, IEnumerable<string> tableRows, IEnumerable<string> tableColumns, string value)  {  var table = Форма[tableCode].МатрицаЗначений;    foreach (var rowCode in tableRows)  {  foreach (var columnCode in tableColumns)  {  table[rowCode, columnCode].Значение = value;  }  }  }    //Пример как заполнить все свободные ячейки формы  private void FillAllFreeCells()  {  var freeCells = Форма.МетаструктураФормы.СвободныеЯчейки.Keys;    foreach (var freeCellCode in freeCells)  {  Форма.СвободныеЯчейки[freeCellCode].Значение = "заполняемое значение";  }  }    //Пример как заполнить все свободные ячейки во всех таблицах  private void FillAllTableFreeCells()  {  var tablesDictionary = Форма.МетаструктураФормы.Таблицы;    foreach (var (tableCode, tableMeta) in tablesDictionary)  {  var freeCells = tableMeta.СвободныеЯчейки.Keys;    FillTableFreeCells(tableCode, freeCells, "заполняемое значение");  }  }    private void FillTableFreeCells(string tableCode, IEnumerable<string> freeCells, string value)  {  var talbeFreeCells = Форма[tableCode].СвободныеЯчейки;    foreach (var freeCell in freeCells)  {  talbeFreeCells[freeCell].Значение = value;  }  } |

# Получение другой ОФ

Для получения другой ОФ можно воспользоваться статическим методом ДополнительныеМетодыПолученияДанных.ПолучитьОтчетнуюФорму(), который имеет несколько перегрузок:

* ПолучитьОтчетнуюФорму(string КодФормы, IУчреждение Учреждение, IКомпонентОтчетногоПериода КомпонентПериода, ТипЭлементаЦепочки ЭлементЦепочки);
* ПолучитьОтчетнуюФорму(string КодФормы, IКомпонентОтчетногоПериода КомпонентПериода, ЭлементЦепочки ЭлементЦепочки);
* ПолучитьОтчетнуюФорму(string КодФормы, IУчреждение Учреждение, string КодКомпонентаПериода, ТипЭлементаЦепочки ЭлементЦепочки);
* ПолучитьОтчетнуюФорму(string КодФормы, string КодКомпонентаПериода, ЭлементЦепочки ЭлементЦепочки).

Получение ОФ:

|  |
| --- |
| private ОтчетнаяФормаДанных ПервыйСпособПолученияФормы()  {  return ДополнительныеМетодыПолученияДанных.ПолучитьОтчетнуюФорму(  "КодФормы", //Код формы, которую необходимо получить  Форма.Идентификатор.Учреждение, //Учреждение формы, которую необходимо получить  Форма.Идентификатор.КомпонентОтчетногоПериода, //Компонент отчетного периода формы, которую необходимо получить  Форма.Идентификатор.ЭлементЦепочки); //Тип элемента цепочки формы, которую необходимо получить  }    private ОтчетнаяФормаДанных ВторойСпособоПолученияФормы()  {  var идентификатор = new ИдентификаторДанныхФормы();  идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания = "КодФормы"; //Код формы, которую необходимо получить  идентификатор.КомпонентОтчетногоПериода = Форма.Идентификатор.КомпонентОтчетногоПериода; //Компонент отчетного периода формы, которую необходимо получить  идентификатор.Учреждение = Форма.Идентификатор.Учреждение; //Учреждение формы, которую необходимо получить  идентификатор.ЭлементЦепочки = ТипЭлементаЦепочки.Офис; //Тип элемента цепочки формы, которую необходимо получить    var отчетнаяФорма = new ОтчетнаяФормаДанных(идентификатор);  отчетнаяФорма.Загрузить(new ПараметрыЗагрузкиОтчетнойФормыДанных());    return отчетнаяФорма;  } |

Для получения «Учреждение» и «КомпонентПериода» можно использовать статические методы класса ДополнительныеМетодыПолученияДанных или DataStore:

* ДополнительныеМетодыПолученияДанных.ПолучитьКомпонентПериодаПоКоду(string КодКомпонента);
* ДополнительныеМетодыПолученияДанных.ПолучитьУчреждениеПоКоду(string КодУчреждения).

Способы получения «Учреждение» и «Компонент отчетного периода»:

|  |
| --- |
| //С использование ДополнительныеМетодыПолученияДанных  var компонентОтчетногоПериода = ДополнительныеМетодыПолученияДанных.ПолучитьКомпонентПериодаПоКоду(string КодКомпонента);  var учреждение = ДополнительныеМетодыПолученияДанных.ПолучитьУчреждениеПоКоду(string КодУчреждения);    //С использованием DataStore  var dataStore = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>();    var компонентОтчетногоПериода = dataStore.GetQuery<КомпонентОтчетногоПериода>()  .Where(x => x.Код == "КодКомпонентаОП")  .FillFields()  .FirstOrDefault();    var учреждение = dataStore.GetQuery<Учреждение>()  .Where(x => x.Код == "КодУчреждения")  .FillFields()  .FirstOrDefault(); |

# Обрабатываемые события при работе с формой

## Написание событий макросом

Для того чтобы использовать возможности базового обработчика формы, необходимо от него унаследоваться:

|  |
| --- |
| public Ваш\_Модуль : БазовыйОбработчикФорм  {    } |

## Обработки

### ОбработатьСобытиеОткрытияФормы

|  |
| --- |
| public override void ОбработатьСобытиеОткрытияФормы(АргументыОткрытияФормы аргументы)  {  } |

Доступны «Аргументы Открытия Формы».

С помощью данных аргументов можно задать РежимОткрытияЭкраннойФормы(enum) или РежимСборкиСводнойЭкраннойФормы(enum).

**Пример:**

аргументы.РежимОткрытияЭкраннойФормы = РежимОткрытияЭкраннойФормы.Просмотр(Редактирование);

аргументы.РежимСборкиСводнойЭкраннойФормы = РежимСборкиСводнойЭкраннойФормы.НеСобирать(Собирать);

### ОбработатьСобытиеДоСохраненияФормы

|  |
| --- |
| public override АргументыСохраненияФормы ОбработатьСобытиеДоСохраненияФормы(АргументыСохраненияФормы аргументы)  {  if (аргументы.SavingSummaryForm)  {  //sth  }  аргументы.ФлажокОтменыСохранения = true;  return аргументы;  } |

Обработка сработает перед сохранением. На этом этапе можно менять значения ячеек или удалять строки (т.е. менять данные формы).

Доступны АргументыСохраненияФормы:

* SavingSummaryForm(bool) – доступен только для чтения и служит для проверки необходимости сохранения сводной формы;
* ФлажокОтменыСохранения(bool) – позволяет отменять сохранения при несоответствии каких-либо условий.

### ОбработатьСобытиеЗакрытияСубтаблицы

|  |
| --- |
| public override АргументыЗакрытияСубтаблицы ОбработатьСобытиеЗакрытияСубтаблицы (АргументыЗакрытияСубтаблицы аргументы)  {  if (sth)  {  аргументы.ФлажокОтменыЗакрытия = true;  }  if (аргументы.ЗакрытиеССохранением)  {  //do sth  }  аргументы.ТаблицаДанных...  return аргументы;  } |

Обработка сработает при закрытии субтаблицы.

Доступные аргументы:

* ФлажокОтменыСохранения(bool) – позволяет отменять сохранения при несоответствии каких-либо условий;
* ЗакрытиеССохранением(bool) – доступен только для чтения. Проверяет, нажата ли кнопка «Закрыть» или «Сохранить» при выходе из субтаблицы;
* ТаблицаДанных(ТаблицаДанных) – это сама субтаблица. С помощью нее можно изменять данные и производить различные проверки для других аргументов.

### ОбработатьСобытиеЗакрытияФормы

|  |
| --- |
| public override АргументыЗакрытияФормы ОбработатьСобытиеЗакрытияФормы (АргументыЗакрытияФормы аргументы)  {  if (sth)  {  аргументы.ФлажокОтменыЗакрытия = true;  }  if (аргументы.ЗакрытиеССохранением)  {  //do sth  }  return аргументы;  } |

Обработка сработает при закрытии формы.

Доступные аргументы:

* ФлажокОтменыСохранения(bool) – позволяет отменять сохранения при несоответствии каких-либо условий;
* ЗакрытиеССохранением(bool) – доступен только для чтения. Проверяет, нажата ли кнопка «Закрыть» или «Сохранить» при выходе из Формы.

### ОбработатьСобытиеПередЗапускомМакроса

|  |
| --- |
| public override АргументыПередЗапускомМакроса ОбработатьСобытиеПередЗапускомМакроса (АргументыПередЗапускомМакроса аргументы)  {  if (аргументы.ИмяМакроса == "Макрос Который Не нужно запускать при определенных условиях")  {  аргументы.ФлажокОтменыЗапускаМакроса = true;  }  return аргументы;  } |

Обработка сработает перед запуском любого макроса.

Доступные аргументы:

* ИмяМакроса(string) – имя макроса, который запускается в данный момент;
* ФлажокОтменыЗапускаМакроса(bool) – позволяет отменять запуск макроса при несоответствии каких-либо условий.

### ОбработатьСобытиеПередИмпортомДанных

|  |
| --- |
| public override АргументыПередИмортомДанных ОбработатьСобытиеПередИмпортомДанных (АргументыПередИмортомДанных аргументы)  {  if (аргументы.НаименованиеИсточникаИмпортаДанных == "Ненужный источник")  {  аргументы.ФлажокОтменыИмпортаДанных = true;  }  return аргументы;  } |

Обработка сработает перед любым импортом данных («Меню/ Импорт/ ...»).

Доступные аргументы:

* НаименованиеИсточникаИмпортаДанных(string) – источник данных импорта. С помощью него можно узнать, какой тип импорта был запущен;
* ФлажокОтменыИмпортаДанных.(bool) – позволяет отменять импорт данных при несоответствии каких-либо условий.

### ОбработатьСобытиеПередОткрытиемТаблицы

|  |
| --- |
| public override void ОбработатьСобытиеПередОткрытиемТаблицы(АргументыПередОткрытиемТаблицы аргументы)  {  if (аргументы.ЗаголовокТаблицы == "currentTableName")  {  string параметры = аргументы.ПараметрыОбработки;  аргументы.ТаблицаДанных...  }  } |

Обработка срабатывает при открытии выбраной таблицы.

Доступны аргументы:

* ЗаголовокТаблицы (string) – заголовок текущей таблицы;
* ПараметрыОбработки(string) – параметры обработки данной таблицы;
* ТаблицаДанных(ТаблицаДанных) – открываемая таблица.

### ОбработатьСобытиеПередПроверкойУвязок

|  |
| --- |
| public override АргументыПередПроверкойУвязок ОбработатьСобытиеПередПроверкойУвязок(АргументыПередПроверкойУвязок аргументы)  {  if (аргументы.ТипУвязок == "Не подходящий тип")  {  аргументы.ФлажокОтменыОперации = true;  }  аргументы.СписокУвязок...  return аргументы;  } |

Обработка срабатывает перед проверкой сформированных увязок.

Доступные аргументы:

* ТипУвязок(string) – тип увязок, который будет проверяться;
* ФлажокОтменыОперации(bool) – позволяет отменять проверку увязок при несоответствии каких-либо условий.

### ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы

|  |
| --- |
| public override void ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы(АргументыПередУдалениемСтрокиДинамическойТаблицы аргументы)  {    } |

ДинамическаяТаблица(ТаблицаДанных) – это сама динамическая таблица, откуда удаляется строка. С помощью нее можно изменять данные и производить различные проверки для других аргументов.

ОтменитьУдаление(bool) – позволяет отменять удаление строки при несоответствии каких-либо условий.

СообщенияОперации(IList)

Строка(СтрокаДанных) – сама удаляемая строка. Используется, например, для переноса данных в другие таблицы перед удалением.

### ОбработатьСобытиеПередЭкспортомДанных

|  |
| --- |
| public override АргументыПередЭкспортомДанных ОбработатьСобытиеПередЭкспортомДанных(АргументыПередЭкспортомДанных аргументы)  {  if (аргументы.НаименованиеИсточникаЭкспортаДанных == "Ненужный источник")  {  аргументы.ФлажокОтменыЭкспортаДанных= true;  }  return аргументы;  } |

Обработка сработает перед любым экспортом данных («Меню/ Экспорт/ ...»).

Доступные аргументы:

* НаименованиеИсточникаЭкспортаДанных(string) – источник данных экспорта. С помощью него можно узнать, какой тип экспорта был запущен;
* ФлажокОтменыЭкспортаДанных.(bool) – позволяет отменять импорт данных при несоответствии каких-либо условий.

### ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы

|  |
| --- |
| public override void ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы (АргументыДобавленияСтрокиДинамическойТаблицы аргументы)  {    } |

Срабатывает после добавления строки в динамическую таблицу.

Доступные аргументы:

* ДинамическаяТаблица(ТаблицаДанных) – это динамическая таблица, в которую была добавлена строка. С помощью нее можно изменять данные и производить различные проверки для других аргументов;
* Строка(СтрокаДанных) – это добавленная строка. Можно сразу заполнять какие-либо ячейки заранее в зависимости от задач.

### ОбработатьСобытиеПослеЗагрузкиАрхива

|  |
| --- |
| public override void ОбработатьСобытиеПослеЗагрузкиАрхива (АргументыПослеЗагрузкиАрхива аргументы)  {    } |

Доступные аргументы:

* МенеджерИстории(IМенеджерИсторииСборкиИтоговыхОтчетов);
* СписокАрхивов(List<IАрхивХранимыхДанныхФорм>).

### ОбработатьСобытиеПослеЗапускаМакроса

|  |
| --- |
| public override void ОбработатьСобытиеПослеЗапускаМакроса(АргументыПослеЗапускаМакроса аргументы)  {    } |

Срабатывает после запуска макроса.

Доступные аргументы:

* ИмяМакроса(string) – можно использовать для логгирования или ведения статистики, отслеживать, в каком порядке макросы запускаются.

### ОбработатьСобытиеПослеИзмененияЗначенияЯчейки

|  |
| --- |
| public override void ОбработатьСобытиеПослеИзмененияЗначенияЯчейки (АргументыСобытияПослеИзмененияЗначенияЯчейки аргументы)  {    } |

Срабатывает после того, как в ячейку внесли изменение.

Доступные аргументы:

* КодЯчейки(string) – код ячейки, в которую внесли изменение;
* ТаблицаДанных(Таблица Данных) – таблица данных, в ячейку которой внесли изменение.

### ОбработатьСобытиеПослеИмпортаДанных

|  |
| --- |
| public override void ОбработатьСобытиеПослеИмпортаДанных (АргументыПослеИмпортаДанных аргументы)  {    } |

Срабатывает после импорта данных. Обычно используется для того, чтобы переписать некоторые значения импортируемого файла, или чтобы проверить, правильно ли сели данные в форму.

Доступные аргументы:

* НаименованиеИсточникаИмпортаДанных(string) – полное наименование пути импорта данных.

### ОбработатьСобытиеПослеПроверкиУвязок

|  |
| --- |
| public override АргументыПослеПроверкиУвязок ОбработатьСобытиеПослеПроверкиУвязок(АргументыПослеПроверкиУвязок аргументы)  {  return аргументы;  } |

Срабатывает после любой проверки увязок (внутреформенных, межформенных).

Доступные аргументы:

* КоличествоПроверенныхУвязок(int) – количество проверенных увязок;
* СписокОшибокПроверки(List<ОписаниеОшибкиПроверкиУвязки>) – можно посмотреть, какие типы ошибок были найдены при проверке;
* ТипУвязок(string) – тип проверенных увязок.

### ОбработатьСобытиеПослеСохраненияСубтаблицы

|  |
| --- |
| public override АргументыПослеСохраненияСубтаблицы ОбработатьСобытиеПослеСохраненияСубтаблицы(АргументыПослеСохраненияСубтаблицы аргументы)  {  return аргументы;  } |

Срабатывает после сохранения субтаблиц (если внести изменения в этой обработке, то они не будут сохранены). Если необходимо поменять данные перед сохранением, то необходимо переопределить обработчик «ОбработатьСобытиеСохраненияСубтаблицы».

Доступные аргументы:

* ДанныеСубтаблицы(ТаблицаДанных) – сохраненная субтаблица;
* ИдентификаторСубтаблицы(string) – идентификатор сохраненной субтаблицы.

### ОбработатьСобытиеПослеУстановкиЗначенияЯчейкиСправочника

|  |
| --- |
| public override void ОбработатьСобытиеПослеСохраненияСубтаблицы(АргументыПослеУстановкиЗначенияЯчейки аргументы)  {    } |

Сработает после того, как в ячейку внесли (выбрали) значение из справочника и нажали на кнопку «Ок».

Доступные аргументы:

* КодСтолбца(string) – код столбца, где было выбрано значение из справочника;
* КодСтроки(string) – код строки, где было выбрано значение из справочника;
* ПересчитатьАвтоблоки(bool) – если установлено значение «true», то после установки значения справочника будут пересчитаны автоблоки;
* ТаблицаДанных(ТаблицаДанных) – таблица данных, где было выбрано значение из справочника.

### ОбработатьСобытиеПослеЭкспортаДанных

|  |
| --- |
| public override void ОбработатьСобытиеПослеЭкспортаДанных(АргументыПослеЭкспортаДанных аргументы)  {    } |

Срабатывает после непосредственного экспорта данных. В аргументах можно только получить «НаименованиеИсточникаЭкспортаДанных(string)».

### ОбработатьСобытиеПриОткрытииТаблицы

|  |
| --- |
| public override void ОбработатьСобытиеПриОткрытииТаблицы(АргументыПриОткрытииТаблицы аргументы)  {  string param = аргументы.ПараметрыОбработки;  аргументы.ТаблицаДанных.МатрицаЗначений...  } |

Срабатывает перед тем, как таблица откроется для просмотра.

Доступные аргументы:

* ПараметрыОбработки(string);
* ТаблицаДанных(ТаблицаДанных) – открываемая таблица.

### ОбработатьСобытиеСохраненияСубтаблицы

|  |
| --- |
| public override АргументыСохраненияСубтаблицы ОбработатьСобытиеСохраненияСубтаблицы(АргументыСохраненияСубтаблицы аргументы)  {    return аргументы;  } |

Срабатывает во время сохранения субтаблицы.

Аргументы:

* ДанныеСубтаблицы(ТаблицаДанных) – сохраняемая субтаблица;
* ФлажокОтменыСохранения(bool) – позволяет отменять сохранения при несоответствии каких-либо условий.
* ИдентификаторСубтаблицы(string) – идентификатор сохраняемой субтаблицы.

### ОбработатьСобытиеСохраненияФормы

|  |
| --- |
| public override АргументыСохраненияСубтаблицы ОбработатьСобытиеСохраненияФормы(АргументыСохраненияСубтаблицы аргументы)  {    return аргументы;  } |

Срабатывает при нажатии на сохранение формы. Обработка содержит два аргумента:

* SavingSummaryForm(bool) – проверяет, сохраняется ли сводная форма;
* ФлажокОтменыСохранения(bool) – позволяет отменять сохранения при несоответствии каких-либо условий.

### ОбработатьСобытиеФормированияВнутреннихУвязок

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияВнутреннихУвязок(АргументыФормированияВнутриформенныхУвязок аргументы)  {  List<ОбъектУвязки> результат = new List<ОбъектУвязки>();  return результат;  } |

Срабатывает при выборе пункта «Меню/ Увязки/ Внутриформенные увязки». Есть две перегрузки этого метода: с аргументами и без них. Задача обработки – проверить соответствие значений в форме по определенной логике, и для каждой несостыковки создать «ОбъектУвязки» и добавить его в List<ОбъектУвязки>.

ДоступныеАргументы:

* ВыбранныеЗакладки(List<string>) – доступ только для чтения. Идентификаторы вкладок, по которым проверяются увязки.

### ОбработатьСобытиеФормированияМежформенныхУвязок

|  |
| --- |
| public override List<ОбъектУвязки> ОбработатьСобытиеФормированияМежформенныхУвязок(АргументыФормированияВнутриформенныхУвязок аргументы)  {  List<ОбъектУвязки> результат = new List<ОбъектУвязки>();  return результат;  } |

Срабатывает при выборе «Меню/ Увязки/ Межформенные увязки». Задача обработки – проверить соответствие значений в форме с другой формой по определенной логике, и для каждой несостыковки создать «ОбъектУвязки» и добавить его в List<ОбъектУвязки>.

Данный метод не содержит аргументов.

### ОбработатьНеверноЗаполненныеЯчейкиДоЗакрытияСубтаблицы

|  |
| --- |
| public override АргументыОбработкиНеВерноЗаполненныхЯчеек ОбработатьНеверноЗаполненныеЯчейкиДоЗакрытияСубтаблицы(АргументыОбработкиНеВерноЗаполненныхЯчеек аргументы)  {  if (аргументы.НайденыНеВерноЗаполненныеЯчейки)  {  аргументы.Таблица.МатрицаЗначений.УдалитьСтроки();//Например  }  return аргументы;  } |

Срабатывает при закрытии таблицы. Можно изменять значения при нахождении неправильно заполненых ячеек. Содержит следующие аргументы:

* НайденыНеВерноЗаполненныеЯчейки(bool) – проверяет наличие в таблице неправильно заполненных ячеек;
* Таблица(ТаблицаДанных) – закрываемая таблица данных.

### ПолучитьИсточникЗаписейДляВыбораИзСправочника

Событие перед формированием записей справочника.

|  |
| --- |
| public override object ПолучитьИсточникЗаписейДляВыбораИзСправочника(АргументыСобытияПолученияИсточникаЗаписейДляВыбораИзСправочника аргументы)  {  var текущийСправочник = new ВыборИзУниверсальногоСправочника();  if (аргументы.ТаблицаДанных.КодТаблицы == "Динамическая" &&  аргументы.КодСтолбца == "4")  {  if (Форма.Идентификатор.ЭлементЦепочки == ТипЭлементаЦепочки.ЦентральныйОфис)  {  текущийСправочник.СписокЗаписей = СправочникПрофессий.Записи.Where(запись => запись.Код == "Автомтранс" ||  (запись.РодительскаяЗапись != null && запись.РодительскаяЗапись.Код == "Автомтранс")).ToList();  }  else  {  текущийСправочник.СписокЗаписей = СправочникПрофессий.Записи;  }  }  return текущийСправочник;  }  private IУниверсальныйСправочник справочникПрофессий;  public IУниверсальныйСправочник СправочникПрофессий  {  get  {  if (справочникПрофессий == null)  {  var идентификаторСправочника = new ИдентификаторУниверсальногоСправочника()  {  ГруппаСправочника = "СтатотчетностьМО",  КодСправочника = "Профессии",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  };  справочникПрофессий = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(идентификаторСправочника);  }  return справочникПрофессий;  }  } |

Событие срабатывает при нажатии на ячейку с типом «справочник». Для того чтобы данное событие сработало, необходимо в Дизайнере поставить «флажок» в поле «Обработка формирования источника записей» (Рисунок 38).

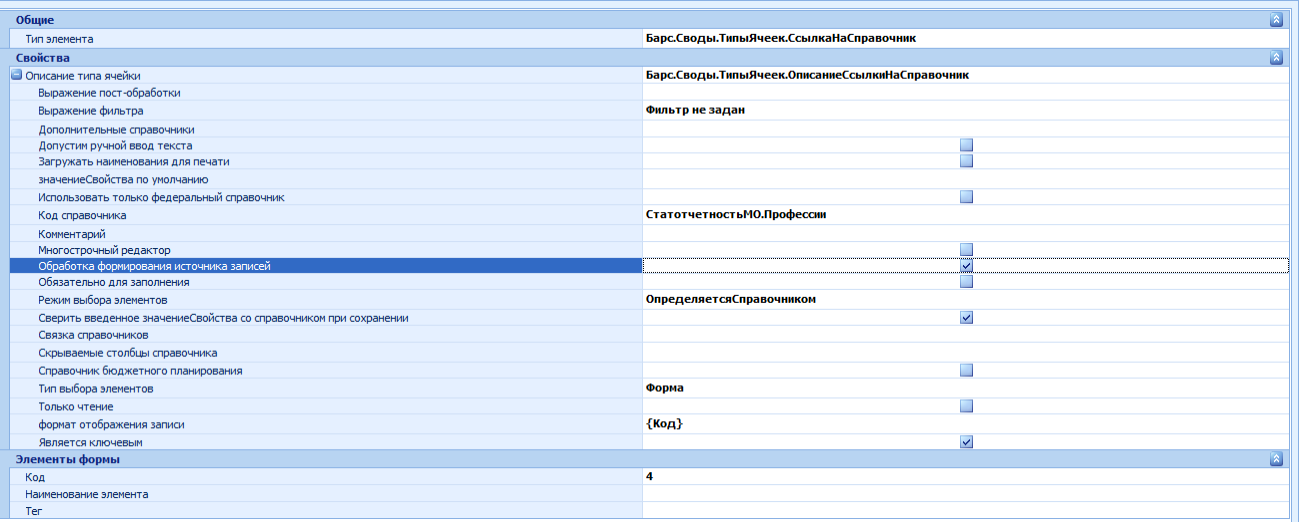


Рисунок 38 – Поле «Обработка формирования источника записей»

### СформироватьСписокПользовательскихПараметров

В данном событии можно определить пользовательские параметры, которые пользователь сможет выбрать перед отрабатыванием какого-либо события (проверка межформенных / внутриформенных увязок, обработки, печатные формы и т.д.).

|  |
| --- |
| protected override СписокПользовательскихПараметров СформироватьСписокПользовательскихПараметров(string Операция)  {  var параметры = new СписокПользовательскихПараметров();  if (Операция == "ОперацияПроверкиМежформенныхУвязок")  {  var отчетныеПериоды = new Выборка<ОтчетныйПериод>();  отчетныеПериоды.Запрос.ДобавитьУсловиеОтбора("ДатаНачала", new DateTime(2017, 1, 1));  отчетныеПериоды.Запрос.ДобавитьУсловиеОтбора("Код", "Тест 105");  отчетныеПериоды.Загрузить();  var списокКомпонентов = отчетныеПериоды[0].Компоненты.Where(компонент => компонент.ПакетФорм.ОтчетныеФормы.Any(форма => форма.ИдентификаторОтчетнойФормы == "105\_О\_достижении\_целевых\_показателей")).Select(оп => оп.Код).ToList();  параметры.ДобавитьПараметрВыборСтроковогоЗначенияИзСписка("Компонент105", "Выберите компонент с формой 105", списокКомпонентов);  }  return параметры;  } |

### Макрос для печатной формы

|  |
| --- |
| public void МакросПечатнойФормы(ref ПараметрыОтчета параметры, ОтчетнаяФормаДанных форма)  {  Секция секцияДинамическая = параметры.СтруктурныеПараметрыОтчета.ДобавитьСекцию("СекцияДинамическая");  int номер = 0;  foreach (СтрокаДанных строка in Форма.Таблицы["Динамическая"].МатрицаЗначений.Строки.Values)  {  секцияДинамическая.ДобавитьСтроку();  foreach (string графа in Форма.Таблицы["Динамическая"].СтолбцыМетаструктуры.Keys)  {  секцияДинамическая[string.Format("\_{0}", графа)] = строка[графа].Значение;  }  секцияДинамическая["\_num"] = номер;  номер++;  }  } |

Данный метод сработает при формировании печатной формы. Название метода может быть любым, метод обязательно должен быть public. Метод должен принимать два параметра: ПараметрыОтчета и ОтчетнаяФормаДанных. Параметры отчета находятся в пространстве имен: Барс.Отчеты, поэтому необходимо добавить using Барс.Отчеты.

# Создание обработки. Атрибуты

Обработка – это различные манипуляции с данными форм, например, перенос данных из других форм, сверка данных, заполнение форм по характеристикам формы прошлого отчетного периода и т.д. Найти в форме обработку можно в меню панели инструментов (Рисунок 39).

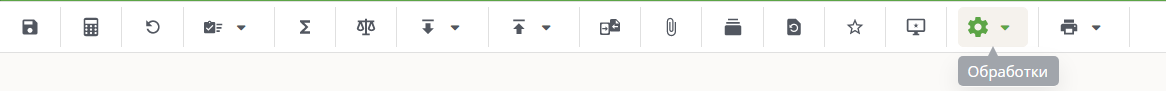


Рисунок 39 – Панель инструментов

Сама обработка реализуется макросом. Посмотреть используемый обработкой метод можно:

* в файле метаструктуры формы (НазваниеФормы.xml), элемент «Меню»;
* в добавленных в классе Модуля Формы атрибутах к методу обработки.

В обработках часто используются пользовательские параметры. Подробнее ознакомиться с формированием пользовательских параметров можно в п. 7 настоящей инструкции.

## Добавление обработки через Дизайнер

Добавить обработку можно в пункт меню Метаструктуры (Рисунок 40).

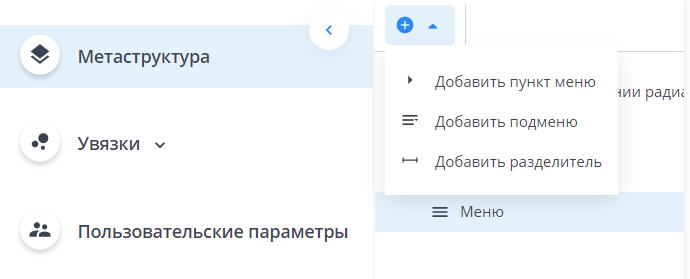


Рисунок 40 – Метаструктура

Пример создания пунктов меню и их отображения в форме представлен на рисунке ниже (Рисунок 41).

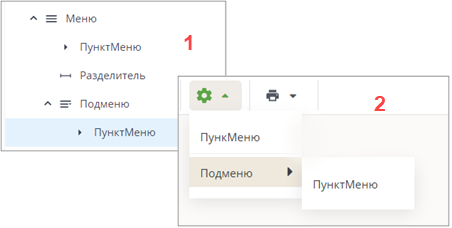


Рисунок 41 – Создание пунктов меню и их отображение в форме

При создании нужно задать:

* Код – название обработки;
* Ассоциированный макрос – название метода, используемое создаваемой обработкой;
* Также можно выбрать, будет ли возможен запуск обработки в режиме «Только чтение», и будет ли он открывать отчетную форму. По умолчанию стоит значение «false».

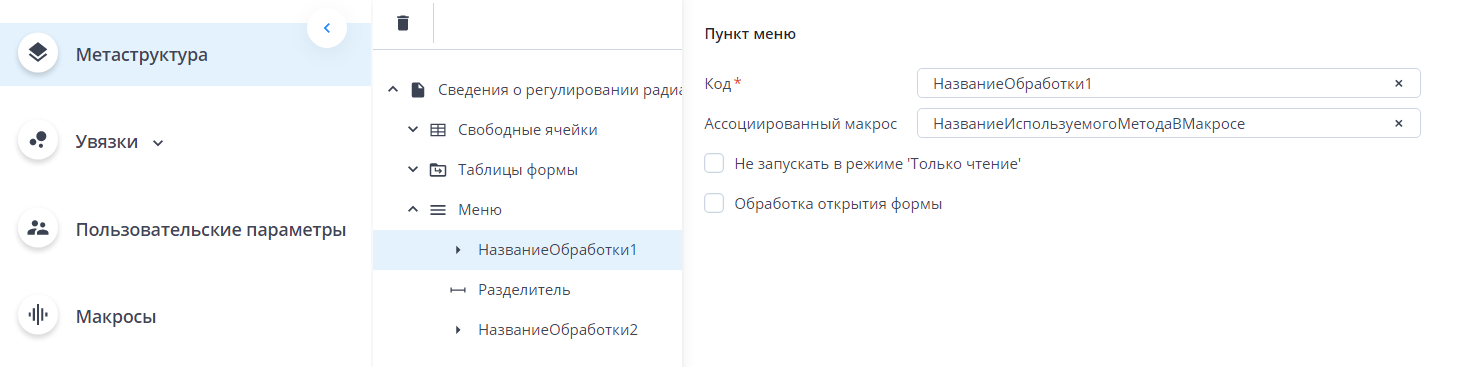


Рисунок 42 – Создание обработки

Дизайнер запишет все изменения в метаструктуру формуы, то же самое можно сделать самим, внеся правки в файле метаструктуры (подробнее описано ниже).

Затем необходимо создать метод в классе модуля формы, например, метод, который будет предупреждать, что сейчас пользователь находится не на ЦО и не на Офисе.

|  |
| --- |
| namespace Барс.Своды  {  public partial class МодульФормы\_НазваниеФормы : БазовыйОбработчикФормы  {  public void НазваниеМетода()  {  if (Форма.Идентификатор.ЭлементЦепочки != ТипЭлементаЦепочки.ЦентральныйОфис && Форма.Идентификатор.ЭлементЦепочки != ТипЭлементаЦепочки.Офис)  {  var словарьСтолбцовСверки = new List<СтолбецСверкиДанныхФормы>();  словарьСтолбцовСверки.Add(new СтолбецСверкиДанныхФормы("Предупреждение", "Предупреждение"));    var текстСообщения = "Это не ЦО и не Офисе";    var словарьСтрокСверки = new List<object[]>();  словарьСтрокСверки.Add(new object[] { текстСообщения });    РезультатВыполненияОперации = new РезультатыВыполненияСверкиДанных(словарьСтолбцовСверки, словарьСтрокСверки);  }  }  }  } |

## Добавление напрямую в файл метаструктуры

Необходимо сделать то же, что сделал Дизайнер, прописав нужные атрибуты в Меню. Как и в Дизайнере есть 3 элемента:

* ПунктМеню;
* Подменю;
* Разделитель.

Получится следующее:

|  |
| --- |
| <Меню>  <ПунктМеню Идентификатор="ПункМеню1" Заголовок="ПункМеню" Расположение="ТолькоМеню" Иконка="" Макрос="МетодОбработки1" OpenFormButton="False" />  <Разделитель />  <Подменю Идентификатор="Подменю" Заголовок="Подменю" Расположение="ТолькоМеню" Иконка="">  <ПунктМеню Идентификатор="ПунктМеню2" Заголовок="ПунктМеню" Расположение="ТолькоМеню" Иконка="" Макрос="МетодОбработки2" OpenFormButton="False" НеЗапускатьВРежимеТолькоЧтение="1" />  </Подменю>  </Меню> |

Можно использовать следующие атрибуты (Таблица 35):

Таблица 35 – Атрибуты

| Поле | Предназначение |
| --- | --- |
| Идентификатор | Идентификатор элемента |
| Заголовок | Название обработки, отображаемое в форме |
| Расположение | Расположение, бывает 3 видов:  1. Только в меню - "ТолькоМеню" 2. И в меню, и на панели инструментов - "МенюИПанельИнструментов" 3. Только на панели инструментов - "ТолькоПанельИнструментов" |
| Макрос | Наименование метода обработки |
| OpenFormButton | = "True"  - пункт меню открывает отчетную форму |
| НеЗапускатьВРежимеТолькоЧтение | = "1" - кнопка запуска обработки не кликабельна в режиме "Только чтение", в остальных случаях можем не задавать |

Метод обработки прописывается так же, как и в Дизайнере.

# Пересчеты

Пересчет – это возможность при необходимости дополнить автоблоки ЭФ макросом. В форме доступны два события для пересчета:

* Событие до пересчета – срабатывает до пересчета автоблоков ЭФ:

|  |
| --- |
| public override void ОбработатьСобытиеДоПересчетаАвтоблоков(АргументыПослеУстановкиЗначенияЯчейки аргументы)  {  if (аргументы.ТаблицаДанных.КодТаблицы == "Статическая")  {  аргументы.ТаблицаДанных.МатрицаЗначений["строка1", "столбец2"].Значение = аргументы.ТаблицаДанных.МатрицаЗначений["строка2", "столбец2"].Значение;  }    base.ОбработатьСобытиеДоПересчетаАвтоблоков(аргументы);  } |

* Событие после пересчета – срабатывает после пересчетов автоблоков ЭФ:

|  |
| --- |
| public override void ОбработатьСобытиеПослеПересчетаАвтоблоков(АргументыПослеУстановкиЗначенияЯчейки аргументы)  {  if (аргументы.ТаблицаДанных.КодТаблицы == "Статическая")  {  аргументы.ТаблицаДанных.МатрицаЗначений["строка1", "столбец2"].Значение = аргументы.ТаблицаДанных.МатрицаЗначений["строка2", "столбец2"].Значение;  }    base.ОбработатьСобытиеПослеПересчетаАвтоблоков(аргументы);  } |

Таблица 36 – Свойства и значения класса АргументыПослеУстановкиЗначенияЯчейки для пересчетов:

| Название | Тип | Описание и возможные значения |
| --- | --- | --- |
| ТаблицаДанных | ТаблицаДанных | Приходит таблица, для которой необходимо пересчитать данные. В некоторых случаях может прийти null |
| КодСтроки | string | Для пересчетов всегда пустая строка |
| КодСтолбца | string | Для пересчетов всегда пустая строка |
| ОбновитьЗначения | bool | Не используется для пересчетов |
| ПересчитатьАвтоблоки | bool | Указывает необходимость запустить пересчет автоблоков ЭФ еще раз |

События пересчетов срабатывают в следующих случаях:

* при открытии формы – для той таблицы, которая будет открыта первой. Т.е. если первая таблица, которую видит пользователь – "Шаблон", то в ТаблицаДанных будет указана таблица «Шаблон»;
* при сохранении формы – для всех таблиц, кроме субтаблиц, реализованных через Дизайнер. Т.е. событие будет вызвано несколько раз, меняя значение в аргументах в поле ТаблицаДанных;
* при открытии таблицы – для таблицы, которая открывается, в том числе для субтаблиц, реализованных через Дизайнер;
* при добавлении строки – для таблицы, в которой добавили строку;
* при сохранении таблицы – для таблицы, которую сохранили;
* при нажатии на кнопку «Пересчитать данные» – для таблиц, в которых были изменены данные.

# Вывод результаты файлом/окном

После выполнения операции есть возможность вывести результат файлом или окном сверки данных. Для этого используйте свойства:

* РезультатВыполненияОперации – для вывода окна;
* РезультатВыполненияОперацииФайл – для вывода файла.

## Вывод результата окном

Используйте для вывода предупреждений или результата сверки данных при операции. РезультатВыполненияОперации присвойте РезультатыВыполненияСверкиДанных.

### РезультатыВыполненияСверкиДанных

Инициализация нового объекта класса

|  |
| --- |
| public РезультатыВыполненияСверкиДанных()  public РезультатыВыполненияСверкиДанных(List<СтолбецСверкиДанныхФормы> произвольныйСписокСтолбцовСверки, List<object[]> произвольныйСписокСтрокСверки) |

#### СтолбецСверкиДанныхФормы

Таблица 37 – СтолбецСверкиДанныхФормы

| Свойство | Тип | Значение |
| --- | --- | --- |
| ИмяСтолбца | string | Имя столбца |
| ЗаголовокСтолбца | string | Заголовок столбца, отображение в таблице сверки |
| ФорматОтображения | string | Формат отображения записи |
| ТипСтолбцаСверкиДанных | ТипСтолбцаСверкиДанных | Тип столбца: None, Numeric, Custom, DateTime |

Инициализация:

|  |
| --- |
| public СтолбецСверкиДанныхФормы(string имяСтолбца, string заголовокСтолбца)  public СтолбецСверкиДанныхФормы(string имяСтолбца, string заголовокСтолбца, string форматОтображения, ТипСтолбцаСверкиДанных типСтолбцаСверкиДанных) |

#### СтрокаОтчетаСверкиДанных

Таблица 38 – СтрокаОтчетаСверкиДанных

| Свойство | Тип | Значение |
| --- | --- | --- |
| Форма | string | Наименование  формы, которой соответствует текущая строка сверки. |
| Вкладка | string | Наименование вкладки формы, которой соответствует текущая строка сверки |
| Субтаблица | string | Идентификатор таблицы, которой соответствует текущая строка сверки. |
| Столбец | string | Код столбца, которому соответствует текущая строка сверки |
| Строка | string | Код строки, которой соответствует текущая строка сверки |
| НаименованиеСтроки | string | Наименование строки, которой соответствует текущая строка сверки |
| НаименованиеСтолбца | string | Наименование столбца, которому соответствует текущая строка сверки. |
| Условие | string | Обычно хранит строку сообщения корректного условия |
| ТипОшибки | string | Расшифровка  ошибки |
| Значение | decimal | Значение ячейки или выражения ячеек, не прошедших проверку на корректность введенных данных |
| СохранениеРазрешено | bool | true, если ячейку с не корректными данными можно сохранить; в противном случае - false |
| RowCode | string | Код строки, которой соответствует текущая строка сверки. |
| ColumnCode | string | Код столбца, которой соответствует текущая строка сверки. |
| IsDeleted | bool | true, если проверенная строка была удалена; в противном случае - false |
| Id | string | Идентификатор строки данных. |

Пример через СтрокаОтчетаСверкиДанных:

|  |
| --- |
| private void СформироватьСтрокуОтчета(ЭлементЦепочки элементЦепочки)  {  var РезультатОтчетаСверки = new РезультатыВыполненияСверкиДанных();    var строкаОтчета = new СтрокаОтчетаСверкиДанных();  строкаОтчета.Субтаблица = "Субтаблица";  строкаОтчета.ТипОшибки = "Форма по учреждению не заполнена";  строкаОтчета.Форма = "Форма";  строкаОтчета.Условие = "Таблица должна быть заполнена";  строкаОтчета.Строка = "01";  РезультатОтчетаСверки.СтрокиСверки.Add(строкаОтчета);    if (РезультатОтчетаСверки.СтрокиСверки.Count > 0)  {  РезультатВыполненияОперации = РезультатОтчетаСверки;  }  } |

Пример, если столбцы формируете сами:

|  |
| --- |
| public void Count2Column()  {  var currentRow = args.ДанныеСубтаблицы.МатрицаЗначений.Строки[args.КодСтроки];  var codeColumn = "2";  var subDataTableCode = currentRow[codeColumn].ЗначениеСтрокой;    if (string.IsNullOrEmpty(subDataTableCode))  {  var resultColumns = new List<СтолбецСверкиДанныхФормы>  {  new СтолбецСверкиДанныхФормы("ошибка", "Предупреждение")  };  var result = new List<object[]>  {  new object[]{"Данные не заполнены, обработка отменена." }  };  РезультатВыполненияОперации = new РезультатыВыполненияСверкиДанных(resultColumns, result)  return;  }  } |

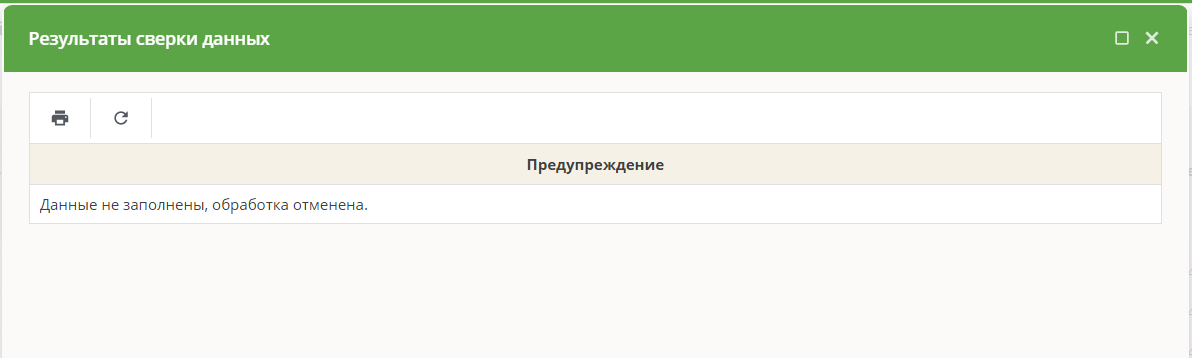


Рисунок 43 – Окно «Результаты сверки данных»

Также можно вывести на печать, нажав на кнопку .

## Вывод результата файлом

Используйте класс РезультатВыполненияОперацииФайл:

Таблица 39 – Класс РезультатВыполненияОперацииФайл

| Свойство | Тип | Значение |
| --- | --- | --- |
| потокФайла | Stream | поток файла для результата операции |
| имяФайла | string | имя файла результата операции |

Пример использования, где после завершения метод выгружает заполняемый в процессе обработки файл, либо выводит окно с ошибкой:

|  |
| --- |
| public void МетодОбработки()  {  var fileName = "Шаблон.xlsx";    var path = Path.GetTempPath();  var templatePath = Path.Combine(Path.GetDirectoryName(Форма.МетаструктураФормы.ФайлМетаописания), fileName);  var tempFilePath = Path.Combine(path, $"{Guid.NewGuid()}{Path.GetExtension(fileName)}");    File.Copy(templatePath, tempFilePath, true);  File.SetAttributes(tempFilePath, FileAttributes.Normal);    using var openXmlWorkbook = КонтейнерЗависимостей.Текущий.Resolve<IOpenXmlWorkbook>();  openXmlWorkbook.Open(tempFilePath);  var activeSheet = openXmlWorkbook.ActiveSheet;  //Здесь должно быть заполнение файла    openXmlWorkbook.Save();  var stream = new MemoryStream();  try  {  using (var fs = File.OpenRead(tempFilePath))  {  fs.CopyTo(stream);  }    stream.Seek(0, SeekOrigin.Begin);  РезультатВыполненияОперацииФайл = new РезультатВыполненияОперацииФайл  {  ПотокФайла = stream,  ИмяФайла = fileName  };  }  catch  {  var resultColumns = new List<СтолбецСверкиДанныхФормы>  {  new СтолбецСверкиДанныхФормы("Ошибка", "Ошибка")  };  var result = new List<object[]>  {  new object[]{"Ошибка выгрузки файла" }  };  РезультатВыполненияОперации = new РезультатыВыполненияСверкиДанных(resultColumns, result)  }  } |

# Основные методы работы с данными формы

## ЗагрузитьМетаструктуру

|  |
| --- |
| public void ЗагрузитьМетаструктуру(МетаструктураФормы метаструктура)  {  } |

Загружает схему, описывающую структуру заполняемых данных формы. Параметр «метаструктура» – это метаструктура формы, описывающая структуру данных.

### Параметр типа МетаструктураФормы

Таблица 40 – Параметр типа МетаструктураФормы

| Свойства | Тип | Описание |
| --- | --- | --- |
| ВерсияФорматаМетаструктуры | decimal | Возвращает версию формата метаструктуры |
| Авторство | string | Возвращает автора формы. Обычно отображается наименование учреждения, в котором разрабатывается форма |
| ВерсияМетаописания | string | Возвращает версию формы |
| Группа | string | Возвращает наименование группы формы |
| СсылкаНаМетодическийСправочник | string | Возвращает код доступа к странице методического справочника |
| СсылкаНаВнешнююСправку | string | Возвращает ссылку на внешнюю справку |
| ДатаНачалаДействия | DateTime | Возвращает дату начала периода действия формы |
| ДатаОкончанияДействия | DateTime | Возвращает дату окончания периода действия формы |
| Код | string | Возвращает обязательный реквизит Код метаструктуры формы |
| Наименование | string | Возвращает наименование формы. Введенное наименование отображается в названии окна готовой формы, а также является названием узла в области компонентов |
| Тег | string | Возвращает тег |
| Идентификатор | ИдентификаторМетаописанияФормы | Возвращает и задает обязательный реквизит Идентификатор метаструктуры формы |
| СвободныеЯчейки | Dictionary<string, ЯчейкаМетаструктуры> | Возвращает словарь свободных ячеек в виде набора объектов ЯчейкаМетаструктуры |
| Таблицы | Dictionary<string, ТаблицаМетаструктуры> | Возвращает словарь объектов ТаблицаМетаструктуры, описывающих структуру данных формы |
| Меню | МенюМетаструктуры | Возвращает меню метаструктуры формы |
| Справочники | Dictionary<string, СправочникМетаструктуры> | Возвращает словарь объектов СправочникМетаструктуры |
| ДатаПоследнегоИзменения | DateTime | Возвращает дату последнего изменения метаструктуры формы |
| НомерВерсии | int | Возвращает номер версии метаструктуры формы |
| Хост | string | Возвращает хост метаструктуры формы |
| ВерсияФормы | string | Возвращает версию формы. Версия присваивается на основании даты последнего изменения и номера сборки и содержит следующие данные, указанные через точку: год.месяц.день.час.мин.сек.номер сборки |
| ФайлМетаописания | string | Возвращает путь к файлу метаописания |
| РасположениеШапки | РасположениеШапки (enum) | Возвращает расположение шапки |
| CodeGenerationConfig | CodeGenerationConfig | Содержит параметры используемые при генерации кода cs-файлов формы |

## Загрузить

|  |
| --- |
| public bool Загрузить(Stream потокДанных)  {  if (error)  {  throw new ArgumentNullException("Сообщение об ошибке");  }    return ЗагрузитьИзXPathDocument(new XPathDocument(потокДанных));  } |

Загружает данные из потока. Поток должен содержать данные xml-формата. Параметр «потокДанных» – это поток данных, из которого необходимо загрузить данные. Метод возвращает значение «true», если выгрузка прошла без ошибок. ArgumentNullException вызывается, когда поток данных пустой.

### Параметр типа Stream

Таблица 41 – Параметр типа Stream

| Свойства | Тип | Описание |
| --- | --- | --- |
| Length | long | При переопределении в производном классе получает длину потока в байтах. |
| Position | long | При переопределении в производном классе получает или задает позицию в текущем потоке. |
| CanRead | bool | При переопределении в производном классе возвращает значение, показывающее, поддерживает ли текущий поток возможность чтения. |
| CanSeek | bool | При переопределении в производном классе возвращает значение, которое показывает, поддерживается ли в текущем потоке возможность поиска. |
| CanTimeout | bool | Возвращает значение, которое показывает, может ли для данного потока истечь время ожидания. |
| CanWrite | bool | При переопределении в производном классе возвращает значение, которое показывает, поддерживает ли текущий поток возможность записи. |
| ReadTimeout | int | Возвращает или задает значение в миллисекундах, определяющее период времени, отведенного потоку на выполнение операции чтения. |
| WriteTimeout | int | Возвращает или задает значение в миллисекундах, определяющее период времени, отведенного потоку на выполнение операции записи. |

## ЗагрузитьИзФайла

|  |
| --- |
| public bool ЗагрузитьИзФайла(string ИмяФайлаОтчетнойФормы)  {  return ЗагрузитьИзФайла(ИмяФайлаОтчетнойФормы, false, false);  }    public bool ЗагрузитьИзФайла(string ИмяФайлаОтчетнойФормы, bool ТолькоСвободныеЯчейки)  {  return ЗагрузитьИзФайла(ИмяФайлаОтчетнойФормы, ТолькоСвободныеЯчейки, false);  }    public bool ЗагрузитьИзФайлаСОтключеннойПроверкойСуществованияЗаписейВСправочнике(string ИмяФайлаОтчетнойФормы)  {  return ЗагрузитьИзФайла(ИмяФайлаОтчетнойФормы, false, true);  } |

Загружает данные из xml-файла. Есть возможность указать загрузку только свободных ячеек. Параметр «ИмяФайлаОтчетнойФормы» – имя файла, из которого будет производиться загрузка данных. Параметр «ТолькоСвободныеЯчейки» – необходимо указать значение «true», если требуется загружать только свободные ячейки, и не указывать в параметрах или указать значение «false», если требуются данные для всех ячеек, включая свободные. Параметр «ignoreRecordNotFound» – в методе ЗагрузитьИзФайлаСОтключеннойПроверкойСуществованияЗаписейВСправочнике указывается значение «true», если необходимо загрузить данные для всех ячеек, включая свободные, без проверки на наличие в справочнике. Метод возвращает значение «true», если выгрузка прошла без ошибок. «Exception» вызывается, если передано пустое имя файла, или не удалось загрузить данные из указанного файла.

## Load

|  |
| --- |
| public void Load(byte[] fileData)  {  if (errors)  {  throw new ИсключениеБарсСводов("Сообщение об ошибке");  }  } |

Прочитывает данные формы из массива байтов. Параметр «fileData» – байты файла формы. «ИсключениеБарсСводов» вызывается, если массив байтов null или пустой.

## ЗагрузитьИзXPathDocument

|  |
| --- |
| public bool ЗагрузитьИзXPathDocument(XPathDocument ДокументОтчетнойФормы)  {  return ЗагрузитьИзXPathDocument(ДокументОтчетнойФормы, false);  }    public bool ЗагрузитьИзXPathDocument(XPathDocument ДокументОтчетнойФормы, bool ТолькоСвободныеЯчейки)  {  return ЗагрузитьИзXPathDocument(ДокументОтчетнойФормы, ТолькоСвободныеЯчейки, false);  } |

Загружает данные из XPathDocument. Есть возможность указать загрузку только свободных ячеек. Параметр «ДокументОтчетнойФормы» – объект, из которого будут загружаться данные. Параметр «ТолькоСвободныеЯчейки» – необходимо указать значение «true», если требуется загружать только свободные ячейки. Метод возвращает значение «true», если выгрузка прошла без ошибок. «Exception» вызывается, если переданный документ имеет ошибки в xml-структуре.

### Параметр типа XPathDocument

Предоставляет быстрое, хранимое в памяти и доступное только для чтения представление XML-документа с помощью модели данных XPath.

## СчитатьТаблицы

|  |
| --- |
| public void СчитатьТаблицы(XPathNavigator xPathNavig)  {  } |

Загружает данные таблицы из объекта класса XPathNavigator, указанного в параметре. Параметр «xPathNavig» – объект, из которого будут загружаться таблицы данных. «Exception» вызывается, если переданный объект имеет ошибки в xml-структуре.

### Параметр типа XPathNavigator

Представляет модель курсора для навигации по данным XML и их редактирования.

## СериализоватьВXML

|  |
| --- |
| public XmlDocument СериализоватьВXML(bool СохранятьНезаполненыеЯчейки)  {  var document = new XmlDocument();  return document;  } |

Сериализует все текущие данные формы в объект класса XMLDocument. Есть возможность указать необходимость сериализации незаполняемых ячейки. Параметр «СохранятьНезаполненыеЯчейки» – укажите значение «true» при необходимости сохранять незаполняемые ячейки.

### Возвращаемый тип XMLDocument

Представляет XML-документ. Этот класс можно использовать для загрузки, проверки, изменения, добавления и размещения XML-кода в документе.

## GetFromDataXml

|  |
| --- |
| public XmlDocument GetFromDataXml()  {  var document = new XmlDocument();  return document;  } |

Позволяет получить документ из данных XML. Возвращает XmlDocument.

## Сохранить

|  |
| --- |
| public bool Сохранить(Stream потокДанных)  {  if (потокДанных == null)  {  throw new ArgumentNullException("Сообщение об ошибке");  }    return true;  } |

Сериализует и сохраняет данные формы в указанный поток. Параметр «потокДанных» – поток данных, в который необходимо сохранить данные. Метод возвращает значение «true», если сохранение прошло без ошибок. «ArgumentNullException» вызывается, если переданный поток не инициализирован.

## СохранитьВФайл

|  |
| --- |
| public bool СохранитьВФайл(string ИмяФайла)  {  return СохранитьВФайл(ИмяФайла, false);  }    public bool СохранитьВФайл(string ИмяФайла, bool СохранятьНезаполненыеЯчейки)  {  if (error)  {  throw new Exception("Сообщение об ошибке");  }    return true;  } |

Сохраняет данные в виде xml-представления в указанный файл. Есть возможность указать необходимость сохранения незаполняемых ячейки. Параметр «ИмяФайла» – имя файла, в который будет производиться сохранение. Параметр «СохранятьНезаполненыеЯчейки» – укажите значение «true», если требуется сохранять незаполняемые ячейки, значение «false» или не указывать значние – если не требуется. Метод возвращает значение «true», если сохранение прошло без ошибок.

## Save

|  |
| --- |
| public bool Save()  {  return SaveTo(ПровайдерФайловФормы.ПолучитьПутьКФайлуДанных(Идентификатор));  } |

Сохраняет данные в файл, расположенный в папке КэшДанныхФорм.

## SaveTo

|  |
| --- |
| public bool SaveTo(string fileName, bool writeEmptyCellValues = false)  {  if (error)  {  throw new Exception("Сообщение об ошибке");  }    return true;  } |

Сохраняет в указанный файл. Параметр «fileName» – путь к файлу данных. Параметр «writeEmptyCellValues» – параметр сохранения значений незаполненных ячеек.

## GetDataFileStream

|  |
| --- |
| public MemoryStream GetDataFileStream()  {  var ms = new MemoryStream();  return ms;  } |

Сериализует данные в поток байт в кодировке UTF-8. Метод возвращает поток байтов MemoryStream с данными формы.

### Возвращаемый тип MemoryStream

Наследуется от Stream, свойства которого описаны выше.

Таблица 42 – MemoryStream

| Свойства | Тип | Описание |
| --- | --- | --- |
| Capacity | int | Возвращает или задает число байтов, выделенных для этого потока. |

## MaterializeDictionaryCellValues

|  |
| --- |
| public void MaterializeDictionaryCellValues()  {  } |

Заполняет ссылки на записи справочника в ячейках.

## ЗаполнитьЗначенияСправочников

|  |
| --- |
| public void ЗаполнитьЗначенияСправочников()  {  }    public void ЗаполнитьЗначенияСправочников(ТаблицаДанных таблица)  {  } |

Загружает значения справочников, используемых в данной форме. Есть возможность указать таблицу данной формы. Заполняются ячейки из матриц значений таблиц данной формы, свободные ячейки таблиц и свободные ячейки данной формы. Если в параметрах указана таблица, то заполняются как ячейки из матрицы значений заданной таблицы, так и свободные ячейки указанной таблицы.

### Параметр типа ТаблицаДанных

Таблица 43 – Параметр типа ТаблицаДанных

| Свойства | Тип | Описание |
| --- | --- | --- |
| ДанныеФормы | ДанныеФормы | Возвращает данные формы, к которым относится данная таблица |
| ТаблицаМетаструктуры | ТаблицаМетаструктуры | Возвращает структуру и свойства таблицы |
| СтолбцыМетаструктуры | Dictionary<string, СтолбецМетаструктуры> | Возвращает структуру и свойства столбцов таблицы |
| Идентификатор | string | Возвращает и задает обязательный реквизит Идентификатор таблицы данных |
| КодТаблицы | string | Возвращает и задает обязательный реквизит Код таблицы данных |
| МатрицаЗначений | МатрицаЗначений | Возвращает и задает данные таблицы |
| СвободныеЯчейки | НаборЯчеекДанных | Возвращает и задает свободные ячейки таблицы |
| HasTableReferences | bool | Флаг, возвращает true, если таблица содержит ячейку с типом TableReference, false - если не содержит |
| ParentCell | TableCellCoordinates (структура содержит TableCode, RowCode, ColumnCode) | Координаты ячейки родительской таблицы |
| IsSubTable | bool | Возвращает true, если таблица является субтаблицей (модальной), false - если нет |

## ЗаполнитьСвободныеЯчейки

|  |
| --- |
| public void ЗаполнитьСвободныеЯчейки()  {  } |

Загружает значения справочников, используемых в свободных ячейках данной формы. Заполняются только свободные ячейки данной формы.

## ЗаполнитьЗначенияСправочниковСтрокДанных

|  |
| --- |
| public void ЗаполнитьЗначенияСправочниковСтрокДанных(ТаблицаДанных таблица, IEnumerable<СтрокаДанных> строкиДанных)  {  } |

Загружает значения справочников, используемых в ячейках указанной таблицы данной формы, только для указанных строк. Заполняются ячейки только из матрицы значений заданной таблицы и только в указанных строках. Параметр «таблица» – таблица данных, для которой необходимо заполнить значения справочников. Параметр строкиДанных – коллекция строк данных, для которых требуется заполнить значения справочников.

### Параметр типа СтрокаДанных

Таблица 44 – Параметр типа СтрокаДанных

| Свойства | Тип | Описание |
| --- | --- | --- |
| КодСтроки | string | Возвращает и задает обязательное свойство Код строки данных |
| Ячейки | НаборЯчеекДанных | Возвращает и задает набор ячеек данных, которые относятся к данной строке |
| ТаблицаДанных | ТаблицаДанных | Возвращает и задает таблицу данных, к которой относится данная строка |

## ЗаполнитьЗначенияСправочниковСвободныхЯчеек

|  |
| --- |
| public void ЗаполнитьЗначенияСправочниковСвободныхЯчеек(ТаблицаДанных таблица, IEnumerable<ЯчейкаДанных> свободныеЯчейки)  {  } |

Загружает значения справочников, используемых в указанных свободных ячейках. Заполняются все свободные ячейки из заданной коллекции. Параметр «таблица» – таблица данных, для которой необходимо заполнить значения справочников. Параметр «свободныеЯчейки» – коллекция свободных ячеек, для которых требуется заполнить значения справочников.

### Параметр типа ЯчейкаДанных

Таблица 45 – Параметр типа ЯчейкаДанных

| Свойства | Тип | Описание |
| --- | --- | --- |
| Значение | ТипЯчейки | Возвращает и задает значение ячейки данных |
| КодЯчейки | string | Возвращает обязательный реквизит Код ячейки |
| НаименованиеЭлемента | string | Возвращает необязательный реквизит Наименование элемента |
| ЯчейкаЗаполненаВерно | bool | Возвращает свойство, показывающее, верно ли заполнена ячейка |

## ОчиститьДанные

|  |
| --- |
| public void ОчиститьДанные()  {  }    public void ОчиститьДанные(bool очищатьСвободныеЯчейки)  {  } |

Очищает все данные формы. Удаляются строки динамических таблиц. Для ячеек матрицы значений статических таблиц и свободных ячеек задаются значения по умолчанию. Есть возможность указать, нужно ли очищать свободные ячейки формы. Параметр «очищатьСвободныеЯчейки» – укажите значение «true», если требуется очистить свободные ячейки формы, значение «false» или не указывать – если не требуется.

## СодержитТаблицу

|  |
| --- |
| public bool СодержитТаблицу(string таблица)  {  if (string.IsNullOrEmpty(таблица) || (Таблицы == null) || (Таблицы.Count <= 0))  {  return false;  }    return Таблицы.ContainsKey(таблица);  } |

Проверяет, содержит ли коллекция «Таблицы» таблицу данных с заданным идентификатором. Параметр «таблица» – таблица данных, для которой необходимо заполнить значения справочников. Если передан пустой идентификатор или коллекция «Таблицы» пуста, то метод вернет значение «false». Метод возвращает значение «true», если указанная таблица найдена.

## НайтиЯчейку

|  |
| --- |
| public ТипЯчейки НайтиЯчейку(string кодЯчейки)  {  if (ячейкаТаблицы != null)  {  return ячейкаТаблицы;  }    return null;  } |

Возвращает объект «Барс.Своды.ТипыЯчеек.ТипЯчейки» с указанным в параметрах кодом. Поиск осуществляется в первую очередь среди свободных ячеек таблиц и формы. Параметр «кодЯчейки» – код ячейки, которую требуется найти. Метод возвращает «null», если ячейка не найдена.

### Тип возвращаемого объекта ТипЯчейки

Таблица 46 – Тип возвращаемого объекта ТипЯчейки

| Свойства | Тип | Описание |
| --- | --- | --- |
| Значение | object | Возвращает и задает значение |
| ТолькоЧтение | bool | Флаг возвращает true, если ячейка предназначена только для чтения, false - если нет |
| Hidden | bool | Флаг отображения ячейки в соответствии с ограничениями |
| ЯвляетсяЧислом | bool | Флаг возвращает true, если тип ячейки является числом, false - если нет |
| ЗначениеЧисло | decimal | Возвращает и задает числовое значение ячейки |
| ЯвляетсяЦелымЧислом | bool | Флаг возвращает true, если тип ячейки является целым числом, false - если нет |
| ЗначениеЦелоеЧисло | int | Возвращает и задает целочисленное значение ячейки |
| ЯвляетсяДатаВремя | bool | Флаг возвращает true, если тип ячейки является типом DateTime, false - если нет |
| ЗначениеДатаВремя | DateTime | Возвращает и задает DateTime значение ячейки |
| ЯвляетсяЛогикой | bool | Флаг возвращает true, если тип ячейки является логическим типом, false - если нет |
| ЗначениеЛогика | bool | Возвращает и задает логическое значение ячейки |
| Статус | СтатусЯчейки | Возвращает и задает верность значения ячейки |
| НеВидимая | bool | Флаг возвращает true, если ячейка невидима, false - в обратном случае |
| ЗначениеСтрокой | string | Возвращает значение строкой |
| Описание | IОписаниеТипаЯчейки | Возвращает и задает описание через ОбработчикаТиповЯчеек |
| ОписаниеСтрокой | string | Возвращает описание строкой |

## СкопироватьДанные

|  |
| --- |
| public void СкопироватьДанные(ДанныеФормы исходныеДанные)  {  } |

Копирует данные, переданные в параметрах метода, в текущий объект ДанныеФормы.

### Параметр типа ДанныеФормы

Таблица 47 – Параметр типа ДанныеФормы

| Свойства | Тип | Описание |
| --- | --- | --- |
| МенеджерСправочников | IМенеджерСправочников | Возвращает и задает менеджер справочников, который будет использоваться для данной формы |
| Идентификатор | ИдентификаторДанныхФормы | Возвращает и задает обязательный реквизит Идентификатор данных формы |
| WarningList | List<string> | Список предупреждений об изменениях в метаструктуре |
| МетаструктураФормы | МетаструктураФормы | Возвращает структуру и свойства данных формы |
| СвободныеЯчейки | НаборЯчеекДанных | Возвращает и задает данные и свойства свободных ячеек формы |
| Таблицы | Dictionary<string, ТаблицаДанных> | Возвращает и задает коллекцию данных и свойств таблиц формы |
| Автоблоки | СловарьАвтоблоков | Возвращает словарь автоблоков данной формы |
| ДанныеЗагружены | bool | Возвращает значение, указывающее загружены ли данные в объект ДанныеФормы |
| ЗагружатьСправочники | bool | Возвращает и задает условие, указывающее загружать или нет справочники, используемые при заполнении формы при загрузке объекта ДанныеФормы из файла |
| ValuesChanged | bool | Признак того, что данные были изменены в одной из таблиц формы с момента последнего сохранения |
| HeadChanged | bool | Отдельный признак того, что была отредактирована шапка формы |

## ПерегенерироватьСтрокиДинамическихТаблиц

|  |
| --- |
| public void ПерегенерироватьСтрокиДинамическихТаблиц()  {  } |

Генерирует коды строк динамических таблиц по порядку (0, 1, 2, 3...). Метод влияет только на динамические таблицы.

## ПроверитьУникальностьСтрокДинамическихТаблиц

|  |
| --- |
| public List<СтрокаОтчетаСверкиДанных> ПроверитьУникальностьСтрокДинамическихТаблиц()  {  var списокОшибок = new List<СтрокаОтчетаСверкиДанных>();    return списокОшибок;  } |

Проверяет все динамические таблицы на повторяющиеся строки. Идентичными считаются строки, содержащие одинаковые значения в ключевых ячейках. Если в таблице нет ни одного ключевого столбца, то все строки считаются идентичными. Информация о повторяющихся строках заполняется в список с типом данных «Барс.Своды.СтрокаОтчетаСверкиДанных». Метод возвращает список записей с информацией о повторяющихся строках динамических таблиц данной формы.

### Возвращаемый тип СтрокаОтчетаСверкиДанных

Таблица 48 – Возвращаемый тип СтрокаОтчетаСверкиДанных

| Свойства | Тип | Описание |
| --- | --- | --- |
| Форма | string | Возвращает и задает наименование формы, которой соответствует текущая строка сверки |
| Вкладка | string | Возвращает и задает наименование вкладки формы, которой соответствует текущая строка сверки |
| Субтаблица | string | Возвращает и задает идентификатор таблицы, которой соответствует текущая строка сверки |
| Столбец | string | Возвращает и задает код столбца, которому соответствует текущая строка сверки |
| Строка | string | Возвращает и задает код строки, которой соответствует текущая строка сверки |
| НаименованиеСтроки | string | Возвращает и задает наименование строки, которой соответствует текущая строка сверки |
| НаименованиеСтолбца | string | Возвращает и задает наименование столбца, которому соответствует текущая строка сверки |
| Условие | string | Возвращает и задает строку условия проверки |
| ТипОшибки | string | Возвращает и задает строку расшифровки ошибки |
| Значение | decimal | Возвращает и задает значение ячейки или выражения ячеек, не прошедших проверку на корректность введенных данных |
| СохранениеРазрешено | bool | Возвращает и задает свойство, показывающее, можно ли сохранить данные ячейки, если она не прошла проверку корректности заполнения |
| RowCode | string | Возвращает код строки, которой соответствует текущая строка сверки |
| IsDeleted | bool | Признак удаления проверенной строки |
| Id | string | Идентификатор строки данных |

## ПересчитатьАвтоблоки

|  |
| --- |
| public void ПересчитатьАвтоблоки()  {  } |

Обновляет значения всех ячеек всех таблиц, в которые занесены автоблоки. Метод возвращает значение «true», если коллекция «Автоблоки» не пуста, «false» – если пуста.

## УстановитьНеверноЗаполненыеЯчейки

|  |
| --- |
| public static void УстановитьНеверноЗаполненыеЯчейки(ДанныеФормы данныеФормы, List<СтрокаОтчетаСверкиДанных> cписокОшибокЗаполнения)  {  }    public static void УстановитьНеверноЗаполненыеЯчейки(ТаблицаДанных таблицаДанных, List<СтрокаОтчетаСверкиДанных> cписокОшибокЗаполнения)  {  } |

Заполняет свойство «Барс.Своды.ТипыЯчеек.ТипЯчейки.Статус» для ячеек, указанных в переданном списке ошибок заполнения, значениями, указанными в данном списке. Есть возможность указать в параметрах либо данные формы, либо таблицу данных. Параметр «данныеФормы» – данные формы, для ячеек которой будут обновляться свойства. Параметр «таблицаДанных» – таблица данных, для ячеек которой будут обновляться свойства. Параметр «списокОшибокЗаполнения» – список ошибок, по данным которого будут обновляться свойства ячеек.

## ЗаполнитьЗначенияПоУмолчанию

|  |
| --- |
| public void ЗаполнитьЗначенияПоУмолчанию()  {  } |

Заполнение всех ячеек данных нулевыми значениями по умолчанию в зависимости от типа ячейки. Заполняются как ячейки матриц значений таблиц и свободные ячейки таблиц, так и свободные ячейки формы.

## SetTableChanged

|  |
| --- |
| public void SetTableChanged(ТаблицаДанных table)  {  }    public void SetTableChanged(ТаблицаДанных table, bool value)  {  } |

Устанавливает «флажок» измененных данных в таблице. Метод необходим для того, чтобы после окончания загрузки данных в таблицу при открытии формы пометить её неизменённой. Есть возможность указать пометку таблицы «true» или не указывать, если изменённая, «false», если таблицу считать неизменённой. Параметр «table» – таблица данных. Параметр «value» – значение «флажка».

## SetTableCommited

|  |
| --- |
| public void SetTableCommited(ТаблицаДанных table, bool value)  {  } |

Устанавливает «флажок», означающий, что результат редактирования динамической таблицы был сохранён в данных формы, либо отменён. Параметр «table» – таблица данных. Параметр «value» – значение «флажка».

## SetAllTableComited

|  |
| --- |
| public void SetAllTableComited(bool value)  {  } |

Устанавливает «флажок» всем таблицам, означающий, что результат редактирования динамических таблиц был сохранён в данных формы, либо отменён. Параметр «value» – значение «флажка».

## GetTableChanged

|  |
| --- |
| public bool GetTableChanged(string tableCode)  {  return \_tableChangesTracker.ContainsKey(tableCode) && \_tableChangesTracker[tableCode].HasChanges;  } |

Получает «флажок» наличия изменённых данных в таблице по коду. Параметр «tableCode» – код таблицы. Метод возвращает значение «true», если данные были изменены с момента последнего сохранения, иначе – значение «false».

## GetTableCommited

|  |
| --- |
| public bool GetTableCommited(string tableCode)  {  return \_tableChangesTracker.ContainsKey(tableCode) && \_tableChangesTracker[tableCode].SnapshotCommited;  } |

Получает «флажок» наличия незафиксированных изменений в таблице по коду. Параметр «tableCode» – код таблицы. Метод возвращает значение «true», если данные были изменены с момента последнего сохранения, иначе – значение «false».

## ДополнительныеМетодыПолученияДанных.ПолучитьДанныеОтчетнойФормы

Существует несколько перегрузок метода в зависимости от входных параметров:

|  |
| --- |
| public static ДанныеФормы ПолучитьДанныеОтчетнойФормы(string КодФормы, IУчреждение Учреждение, IКомпонентОтчетногоПериода КомпонентПериода, ТипЭлементаЦепочки ЭлементЦепочки)  {  return форма.Данные;  }    public static ДанныеФормы ПолучитьДанныеОтчетнойФормы(string КодФормы, IУчреждение Учреждение, string КодКомпонентаПериода, ТипЭлементаЦепочки ЭлементЦепочки)  {  return форма.Данные;  }    public static ДанныеФормы ПолучитьДанныеОтчетнойФормы(string КодФормы, IКомпонентОтчетногоПериода КомпонентПериода, ЭлементЦепочки ЭлементЦепочки)  {  return форма.Данные;  }    public static ДанныеФормы ПолучитьДанныеОтчетнойФормы(string КодФормы, string КодКомпонентаПериода, ЭлементЦепочки ЭлементЦепочки)  {  return форма.Данные;  } |

# Работа с вложениями

Для работы с вложениями необходимо получить экземпляр класса МенеджерВложенийХранимыхДанныхФормы.

Вариант 1: создайте объект класса через конструктор:

|  |
| --- |
| **public** МенеджерВложенийХранимыхДанныхФормы(ИдентификаторДанныхФормы идентификаторДанныхФормы) :**this**(ХранилищеДанныхФорм.ПолучитьДанныеФормы(идентификаторДанныхФормы))  {  }    **public** МенеджерВложенийХранимыхДанныхФормы(ХранимыеДанныеФормы хранимыеДанные)  {  **this**.хранимыеДанные = хранимыеДанные;  } |

Вариант 2: получите с помощью метода у экземпляра ОтчетнаяФормаДанных:

|  |
| --- |
| **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных1 = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();  **var** менеджерВложенийХранимыхДанных2 = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормыWeb(); |

Различия ПолучитьМенеджерВложенийХранимыхДанныхФормыWeb() и ПолучитьМенеджерВложенийХранимыхДанныхФормы():

* в ПолучитьМенеджерВложенийХранимыхДанныхФормы() используется конструктор public МенеджерВложенийХранимыхДанныхФормы(ИдентификаторДанныхФормы идентификаторДанныхФормы);
* в ПолучитьМенеджерВложенийХранимыхДанныхФормыWeb() используется конструктор public МенеджерВложенийХранимыхДанныхФормы(ХранимыеДанныеФормы хранимыеДанные).

Возможности:

* получить вложения:

|  |
| --- |
| **private** List<IВложениеХранимыхДанныхФормы> ПолучитьВложения()  {  **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();    **return** менеджерВложенийХранимыхДанных.ПолучитьВложенияФормы();  } |

* сохранить вложение в файл:

|  |
| --- |
| **private** **void** СохранитьВложениеВФайл()  {  **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();    **foreach** (**var** вложение **in** менеджерВложенийХранимыхДанных.ПолучитьВложенияФормы())  {  **using** (**var** memoryStream = менеджерВложенийХранимыхДанных.ПолучитьФайлВложения(вложение))  {  **var** fileStream = File.Create(вложение.Наименование);  memoryStream.WriteTo(fileStream);  fileStream.Close();  }  }  } |

* добавить вложение:

|  |
| --- |
| **private** **void** ДобавитьВложение()  {  **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();  **using** (**var** file = File.Create("Путь до файла"))  {  менеджерВложенийХранимыхДанных.ДобавитьВложение("Наименование", file);  }  } |

* удалить вложение:

|  |
| --- |
| **private** **void** УдалитьВложение()  {  **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();    **foreach** (**var** вложение **in** менеджерВложенийХранимыхДанных.ПолучитьВложенияФормы())  {  менеджерВложенийХранимыхДанных.Удалить(вложение);  }  } |

* свойства вложений:

|  |
| --- |
| **private** **void** НаименованиеВложения()  {  **var** отчетнаяФормаДанных = **new** ОтчетнаяФормаДанных(Форма);  **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();    **foreach** (**var** вложение **in** менеджерВложенийХранимыхДанных.ПолучитьВложенияФормы())  {  **var** наименованиеВложения = вложение.Наименование;  **var** идентификатор = вложение.ИдентификаторОбъекта;  **var** датаСозданияОбъекта = вложение.ДатаСозданияОбъекта;  **var** авторОбъекта = вложение.ПользовательАвторОбъекта;  }  } |

## Настройка вложений

В настройках приложения можно указать, где хранить вложения формы   
(Рисунок 44).

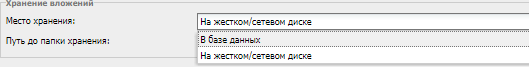


Рисунок 44 – Настройки приложения

Если указано «на жестком/сетевом диске», то необходимо прописать путь до папки хранения.

Настройка разрешения прикрепления вложений для форм со статусом выше чем «Черновик» (Рисунок 45).

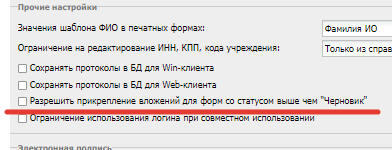


Рисунок 45 – Настройки приложения

**Примечание** – Для того чтобы взаимодействовать с вложениями, необходимо **наличие хранимых данных формы** и **статус формы == Черновик** при выключенной настройке или **статус формы != Пусто** при включенной настройке.

Для того чтобы проверить, можно ли взаимодействовать с вложениями формы, предусмотрено свойство «МожноРедактировать».

|  |
| --- |
| **var** менеджерВложенийХранимыхДанных = отчетнаяФормаДанных.ПолучитьМенеджерВложенийХранимыхДанныхФормы();  **if** (менеджерВложенийХранимыхДанных.МожноРедактировать)  {  //Делаем свои дела  } |

# Работа с архивами данных

Чтобы сохранить данные формы в архив, нажмите на соответствующую иконку в пункте меню (Рисунок 46)



Рисунок 46 – Панель инструментов

Затем сохраните заполненные данные (Рисунок 47), задав наименование   
(Рисунок 48).

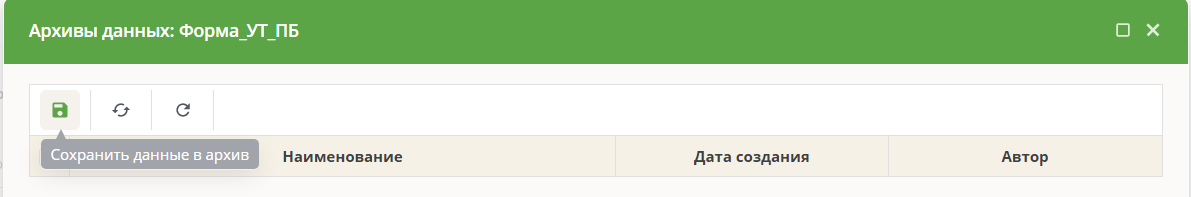


Рисунок 47 – Сохранение данных

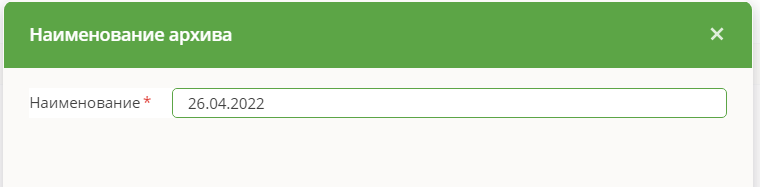


Рисунок 48 – Ввод наименования

Также можно выгрузить данные из сохраненных архивов (Рисунок 49).

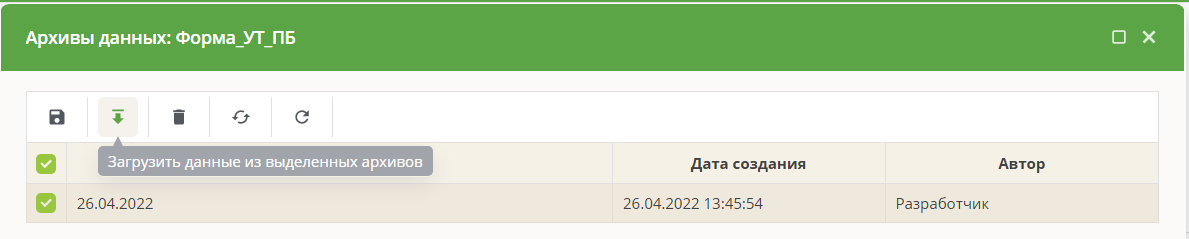


Рисунок 49 – Выгрузка из сохраненных архивов

## Пример получения архива данных формы

Метод получает данные формы на основе архива формы, где архив должен содержать слово «Свод»:

|  |
| --- |
| public ДанныеФормы ПолучитьДанныеФормыИзАрхива()  {  var хранимыеДанные = ХранилищеДанныхФорм.ПолучитьДанныеФормы(Форма.Идентификатор);    if (хранимыеДанные == null || хранимыеДанные.АрхивыДанныхФорм.Count == 0)  {  return null;  }    var датаСоздания = DateTime.MinValue;  var архивХранимыхДанныхФормы = new АрхивХранимыхДанныхФорм();    foreach (var архивХранимыхДанных in хранимыеДанные.АрхивыДанныхФорм)  {  if (архивХранимыхДанных != null && архивХранимыхДанных.Наименование.ToUpper().Contains("СВОД"))  {  if (датаСоздания < архивХранимыхДанных.ДатаПоследнегоИзмененияОбъекта)  {  датаСоздания = архивХранимыхДанных.ДатаПоследнегоИзмененияОбъекта;  архивХранимыхДанныхФормы = архивХранимыхДанных;  }  }  }    System.Xml.XPath.XPathDocument документ;    using (MemoryStream поток = Файл.СчитатьПотокИзБД(архивХранимыхДанныхФормы, "ФайлАрхивДанных", true))  {  документ = new System.Xml.XPath.XPathDocument(поток);  }    var данныеФормы = new ДанныеФормы();  данныеФормы.Идентификатор = Форма.Идентификатор;  данныеФормы.ЗагрузитьМетаструктуру(Форма.МетаструктураФормы);  данныеФормы.ЗагрузитьИзXPathDocument(документ);    return данныеФормы;  } |

Рисунок 50 – Настройки приложения

# Работа с динамическими таблицами, добавление строк, заполнение данных по умолчанию

В основном работа со статическими и динамическими таблицами одинакова. Разница в том, что в динамической таблице можно добавлять и удалять строки.

Проверить, является ли текущая таблица динамической можно через свойство: «таблицаДанных.ТаблицаМетаструктуры.Динамическая».

## Добавление строк

В случае если строки добавляются через кнопку «Добавить», то создается новая строка, в которую можете добавить свою логику, переопределив метод «ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы».

|  |
| --- |
| public override void ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы(АргументыДобавленияСтрокиДинамическойТаблицы аргументы)  {  if (аргументы.ДинамическаяТаблица.IsSubTable)  {  var parentTable = Форма[аргументы.ДинамическаяТаблица.ParentCell.TableCode];  var операция = parentTable[аргументы.ДинамическаяТаблица.ParentCell.RowCode, "Операция"].Значение;  аргументы.Строка["Операция2"].Значение = операция;  }    base.ОбработатьСобытиеПослеДобавленияСтрокиДинамическойТаблицы(аргументы);  } |

При удалении строки вызывается метод «ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы».

|  |
| --- |
| public override void ОбработатьСобытиеПередУдалениемСтрокиДинамическойТаблицы(АргументыПередУдалениемСтрокиДинамическойТаблицы args)  {  if (args.ДинамическаяТаблица.КодТаблицы == "Таблица2" || args.ДинамическаяТаблица.КодТаблицы.StartsWith("Таблица3"))  {  DeleteRowFromTable1(args.ДинамическаяТаблица.КодТаблицы, args.Строка);  }  } |

Если строки добавляются в макросе, то можете добавить строку в «МатрицаЗначений» с помощью одного из методов: «Добавить» или «ДобавитьВНачало». Если нужно создать пустую строку в самой таблице, то это можно сделать с помощью метода «CreateNewRow», однако в этом методе не заполняется автоматически код строки, поэтому необходимо вызвать метод «ПерегенерироватьСтроки».

|  |
| --- |
| //код строки будет равен номеру последней строки  var newRow1 = Форма["Таблица"].МатрицаЗначений.Добавить();    //код строки будет 0  var newRow2 = Форма["Таблица"].МатрицаЗначений.ДобавитьВНачало();    //не будет задан код строки, поэтому вызываем ПерегенерироватьСтроки  var newRow3 = Форма["Таблица"].CreateNewRow();  Форма["Таблица"].МатрицаЗначений.ПерегенерироватьСтроки(); |

Удалить строку макросом можно через метод «Удалить», который принимает код строки. После удаления одной строки обязательно вызовите метод «ПерегенерироватьСтроки», т.к. коды строк в динамической таблице необходимо обновить. Для удаления всех строк таблицы, есть метод «УдалитьСтроки».

|  |
| --- |
| Форма["Таблица"].МатрицаЗначений.Удалить("КодСтроки");  //После удаления строки по коду, обязательно вызываем ПерегенерироватьСтроки  Форма["Таблица"].МатрицаЗначений.ПерегенерироватьСтроки();    //Удаляем все строки всех динамических таблиц в форме  foreach (var dynamicTable in Форма.Таблицы.Values.Where(table => table.ТаблицаМетаструктуры.Динамическая))  {  dynamicTable.МатрицаЗначений.УдалитьСтроки();  } |

Если добавление строки происходит в другой отчетной форме, заблокируйте форму перед добавлением новых строк.

## Заполнение данных по умолчанию

При вызове метода «ЗаполнитьЗначениямиПоУмолчаниюМетаструктуры» в «МатрицаЗначений» происходит заполнение матрицы значениями по умолчанию из метаструктуры.

А при вызове метода «ЗаполнитьЗначенияПоУмолчанию» в таблице заполняются как ячейки матриц значений таблиц и свободные ячейки таблиц, так и свободные ячейки формы.

|  |
| --- |
| Форма["Таблица"].ЗаполнитьЗначенияПоУмолчанию();  Форма["Таблица"].МатрицаЗначений.ЗаполнитьЗначениямиПоУмолчаниюМетаструктуры(); |

# Открытие формы из формы

Для реализации открытия формы из формы выполните следующее:

* по кнопке в таблице:
  + добавьте ячейку типа «Барс.Своды.ТипыЯчеек.ТипКнопкаОткрытьФорму»;
  + выберите ассоциированный макрос соответствующий сигнатуре public OpenFormParameters MethodName(АргументыСобытияНажатияЯчейкиТипаКнопка args) – описание сигнатур представлено в п. 27.1;
  + в указанном макросе реализуйте функциональность для возврата структуры OpenFormParameters.

OpenFormParameters:

|  |
| --- |
| public readonly struct OpenFormParameters  {  public OpenFormParameters(ИдентификаторДанныхФормы identifier, РежимОткрытияЭкраннойФормы openMode, РежимСборкиСводнойЭкраннойФормы summaryMode)  {  Identifier = identifier;  OpenMode = openMode;  SummaryMode = summaryMode;  }    public OpenFormParameters(string openFormErrorMessage)  {  OpenFormErrorMessage = openFormErrorMessage;  Identifier = null;  OpenMode = default;  SummaryMode = default;  FormOpenAllowed = false;  }    /// <summary> Идентификатор данных формы </summary>  public ИдентификаторДанныхФормы Identifier { get; }    /// <summary> Режим открытия формы </summary>  public РежимОткрытияЭкраннойФормы OpenMode { get; }    /// <summary> Режим сборки сводной </summary>  public РежимСборкиСводнойЭкраннойФормы SummaryMode { get; }    /// <summary> Разрешено открытие формы </summary>  public bool FormOpenAllowed { get; }    /// <summary> Сообщение об ошибке открытия формы </summary>  public string OpenFormErrorMessage { get; }  } |

**Отмена открытия:**

Если макрос должен отменить открытие формы и вывести сообщение, то используйте конструктор OpenFormParameters(string openFormErrorMessage). Параметром передается сообщение, которое будет выведено.

Пример метода открытия формы:

|  |
| --- |
| public OpenFormParameters OpenForm(АргументыСобытияНажатияЯчейкиТипаКнопка args)  {  var form = ДополнительныеМетодыПолученияДанных.ПолучитьДанныеОтчетнойФормы(  Форма.Идентификатор.ИдентификаторМетаописанияФормы.ИдентификаторМетаописания,  Форма.Идентификатор.Учреждение,  Форма.Идентификатор.КомпонентОтчетногоПериода,  Форма.Идентификатор.ЭлементЦепочки);    if (form == null)  {  //Для отмены открытия формы передаем в конструктор сообщение об ошибке  return new OpenFormParameters("Форма не найдена");  }    return new OpenFormParameters(form.Идентификатор, РежимОткрытияЭкраннойФормы.Просмотр, РежимСборкиСводнойЭкраннойФормы.НеСобирать);  } |

Возможны два режима открытия формы: «Просмотр» и «Редактирование», а также два режима сборки сводной: «НеСобирать» и «Собирать».

* по кнопке в меню «Обработки»:
  + добавьте пункт меню;
  + установите «флажок» в поле «Обработка открытия формы»;
  + выберите из выпадающего списка ассоциированный макрос, соответствующий сигнатуре public OpenFormParameters MethodName() – описание сигнатур представлено в п. 27.1;
  + в указанном макросе реализуйте функциональность для возврата структуры OpenFormParameters.

## Сигнатуры макросов

При реализации форм через общую API сигнатура методов будет изменяться, т.к. использование общей API подразумевает отказ от ссылок на платформу из макросов форм.

Для поддержки такой функциональности реализован интерфейс IformOpenParametersProvider:

|  |
| --- |
| /// <summary> Провайдер параметров для открытия формы макросом </summary>  public interface IFormOpenParametersProvider  {  /// <summary> Получить параметры открытия формы </summary>  /// <param name="parentForm">Родительская форма</param>  /// <param name="macroName">Макрос формы</param>  /// <returns>Параметры открытия формы</returns>  OpenFormParameters GetParameters(ОтчетнаяФорма.БазоваяОтчетнаяФормаДанных parentForm, string macroName);    /// <summary> Получить параметры открытия формы </summary>  /// <param name="parentForm">Родительская форма</param>  /// <param name="macroName">Макрос формы</param>  /// <param name="args">Аргументы макроса формы</param>  /// <returns>Параметры открытия формы</returns>  OpenFormParameters GetParameters(ОтчетнаяФорма.БазоваяОтчетнаяФормаДанных parentForm, string macroName, object[] args);    /// <summary> Получить список сигнатур методов для пункта меню открытия отчетной формы </summary>  /// <param name="methodSignatures">Список сигнатур методов</param>  /// <returns>Список сигнатур методов для пункта меню открытия отчетной формы</returns>  IEnumerable<MethodSignature> GetMethodsForMenuItem(IEnumerable<MethodSignature> methodSignatures);    /// <summary> Получить список сигнатур методов для кнопки таблицы открытия отчетной формы </summary>  /// <param name="methodSignatures">Список сигнатур методов</param>  /// <returns>Список сигнатур методов для кнопки таблицы открытия отчетной формы</returns>  IEnumerable<MethodSignature> GetMethodsForCellButton(IEnumerable<MethodSignature> methodSignatures);  } |

И его реализация по умолчанию:

|  |
| --- |
| /// <summary> Провайдер параметров для открытия формы макросом </summary>  public sealed class FormOpenParametersProvider : IFormOpenParametersProvider  {  /// <summary> Получить параметры открытия формы </summary>  /// <param name="parentForm">Родительская форма</param>  /// <param name="macroName">Макрос формы</param>  /// <returns>Параметры открытия формы</returns>  public OpenFormParameters GetParameters(ОтчетнаяФорма.БазоваяОтчетнаяФормаДанных parentForm, string macroName)  {  return (OpenFormParameters)parentForm.ВыполнитьМетодМакроса(macroName);  }    /// <summary> Получить параметры открытия формы </summary>  /// <param name="parentForm">Родительская форма</param>  /// <param name="macroName">Макрос формы</param>  /// <param name="args">Аргументы макроса формы</param>  /// <returns>Параметры открытия формы</returns>  public OpenFormParameters GetParameters(ОтчетнаяФорма.БазоваяОтчетнаяФормаДанных parentForm, string macroName, object[] args)  {  return (OpenFormParameters)parentForm.ВыполнитьМетодМакроса(macroName, args);  }    /// <summary> Получить список сигнатур методов для пункта меню открытия отчетной формы </summary>  /// <param name="methodSignatures">Список сигнатур методов</param>  /// <returns>Список сигнатур методов для пункта меню открытия отчетной формы</returns>  public IEnumerable<MethodSignature> GetMethodsForMenuItem(IEnumerable<MethodSignature> methodSignatures)  {  return methodSignatures.Where(signature => signature.ReturnTypeName == nameof(OpenFormParameters) &&  signature.Parameters.Count == 0);  }    /// <summary> Получить список сигнатур методов для кнопки таблицы открытия отчетной формы </summary>  /// <param name="methodSignatures">Список сигнатур методов</param>  /// <returns>Список сигнатур методов для кнопки таблицы открытия отчетной формы</returns>  public IEnumerable<MethodSignature> GetMethodsForCellButton(IEnumerable<MethodSignature> methodSignatures)  {  return methodSignatures.Where(signature => signature.ReturnTypeName == nameof(OpenFormParameters) &&  signature.Parameters.Count == 1 &&  signature.Parameters[0].TypeName == nameof(АргументыСобытияНажатияЯчейкиТипаКнопка));  }  } |

Данный интерфейс предназначен для формирования списка макросов в дизайнере и для вызова и получения результатов макросов.

При реализации общей API необходимо реализовать данный интерфейс с учетом типов-фасадов которые будут использованы вместо типов платформы.

# Работа вне отчетной формы

## Кастомизация статусов

В Системе доступны следующие возможности кастомизации отображения статусов в списке отчетных форм:

* выбор цвета текста статуса;
* выбор цвета фона статуса.

Добавлен интерфейс «IStyledReportFormStatus»:

|  |
| --- |
| /// <summary> Интерфейс стилизованного статуса отчетной формы </summary>  public interface IStyledReportFormStatus : IReportFormStatus  {  /// <summary> Hex цвета фона отображаемого в ячейке грида </summary>  string BackgroundColorHex { get; set; }    /// <summary> Hex цвета текста отображаемого в ячейке грида </summary>  string TextColorHex { get; set; }  } |

Класс «ReportFormStatus» реализует интерфейс «IStyledReportFormStatus». Это позволяет изменять цвета статусов Системы по умолчанию («Пусто», «Черновик», «Заполнено», «Проверено», «Экспертиза», «Утверждено»).

Начиная с версии Системы 5.0 Системы, реализована возможность выбора пиктограммы пункта меню для смены статуса формы (Рисунок 51).



Рисунок 51 – Выбор пиктограммы

Добавлен интерфейс «IReportFormStatusWithIcon»:

|  |
| --- |
| /// <summary> Интерфейс статуса отчетной формы с иконкой пункта меню </summary>  public interface IReportFormStatusWithIcon  {  /// <summary> Css-класс иконки пункта меню </summary>  string IconCls { get; set; }  } |

Класс «ReportFormStatus» реализует интерфейс «IReportFormStatusWithIcon». Это позволяет изменять пиктограммы статусов Системы по умолчанию («Пусто», «Черновик», «Заполнено», «Проверено», «Экспертиза», «Утверждено»).

Объявите css-класс в прикладных файлах:

|  |
| --- |
| .custom-state-icon {  background-image: url("data:image/svg+xml,%3Csvg width='16' height='16' viewBox='0 0 16 16' fill='none' xmlns='http://www.w3.org/2000/svg'%3E%3Cpath fill-rule='evenodd' clip-rule='evenodd' d='M4.00004 3.33334H2.00004C1.63337 3.33334 1.33337 3.63334 1.33337 4.00001V12C1.33337 12.3667 1.63337 12.6667 2.00004 12.6667H4.00004C4.36671 12.6667 4.66671 12.3667 4.66671 12V4.00001C4.66671 3.63334 4.36671 3.33334 4.00004 3.33334ZM13.3334 3.33334H11.3334C10.9667 3.33334 10.6667 3.63334 10.6667 4.00001V12C10.6667 12.3667 10.9667 12.6667 11.3334 12.6667H13.3334C13.7 12.6667 14 12.3667 14 12V4.00001C14 3.63334 13.7 3.33334 13.3334 3.33334ZM6.66671 3.33334H8.66671C9.03337 3.33334 9.33337 3.63334 9.33337 4.00001V12C9.33337 12.3667 9.03337 12.6667 8.66671 12.6667H6.66671C6.30004 12.6667 6.00004 12.3667 6.00004 12V4.00001C6.00004 3.63334 6.30004 3.33334 6.66671 3.33334Z' fill='%2366717F'/%3E%3C/svg%3E%0A");  }    // стиль при наведении на пункт меню  .x-menu-item-active {  .custom-state-icon {  background-image: url("data:image/svg+xml,%3Csvg width='16' height='16' viewBox='0 0 16 16' fill='none' xmlns='http://www.w3.org/2000/svg'%3E%3Cpath fill-rule='evenodd' clip-rule='evenodd' d='M4.00004 3.33334H2.00004C1.63337 3.33334 1.33337 3.63334 1.33337 4.00001V12C1.33337 12.3667 1.63337 12.6667 2.00004 12.6667H4.00004C4.36671 12.6667 4.66671 12.3667 4.66671 12V4.00001C4.66671 3.63334 4.36671 3.33334 4.00004 3.33334ZM13.3334 3.33334H11.3334C10.9667 3.33334 10.6667 3.63334 10.6667 4.00001V12C10.6667 12.3667 10.9667 12.6667 11.3334 12.6667H13.3334C13.7 12.6667 14 12.3667 14 12V4.00001C14 3.63334 13.7 3.33334 13.3334 3.33334ZM6.66671 3.33334H8.66671C9.03337 3.33334 9.33337 3.63334 9.33337 4.00001V12C9.33337 12.3667 9.03337 12.6667 8.66671 12.6667H6.66671C6.30004 12.6667 6.00004 12.3667 6.00004 12V4.00001C6.00004 3.63334 6.30004 3.33334 6.66671 3.33334Z' fill='%23FFFFFF'/%3E%3C/svg%3E%0A");  }  } |

При необходимости кастомизации статусов создайте в прикладных API класс реализующий данные интерфейсы и используйте его в реализации контроллера статусной модели «IReportFormStatusController».

Пример кода:

|  |
| --- |
| /// <summary> Контроллер переходов отчетных форм между статусами </summary>  public sealed class CustomStatusController : IReportFormStatusController  {  /// <summary> Статусы отчетных форм </summary>  private readonly IList<IStyledReportFormStatus> \_statuses;    public CustomStatusController()  {  \_statuses = new List<IStyledReportFormStatus>  {  new ReportFormStatus(СостояниеДанныхФорм.Пусто)  {  BackgroundColorHex = "#555555"  },  new ReportFormStatus(СостояниеДанныхФорм.Черновик),  new ReportFormStatus(СостояниеДанныхФорм.Заполнено),  new ReportFormStatus(СостояниеДанныхФорм.Проверено),  new ReportFormStatus(СостояниеДанныхФорм.Экспертиза),  new ReportFormStatus(СостояниеДанныхФорм.Утверждено),  new CustomStatus  {  Name = "Показатели отсутствуют",  StatusNumber = 6,  BackgroundColorHex = "#FFFFFF",  TextColorHex = "#000000",  IconCls = "custom-state-icon"  }  };  }    /\* остальная реализация интерфейса IReportFormStatusController \*/  }    public sealed class CustomStatus : IStyledReportFormStatus, IReportFormStatusWithIcon  {  /// <summary> Отображаемое наименование </summary>  public string Name { get; set; }    /// <summary> Номер статуса </summary>  public int StatusNumber { get; set; }    /// <summary> Hex цвета фона отображаемого в ячейке грида </summary>  public string BackgroundColorHex { get; set; }    /// <summary> Hex цвета текста отображаемого в ячейке грида </summary>  public string TextColorHex { get; set; }    /// <summary> Css-класс иконки пункта меню </summary>  public string IconCls { get; set; }  } |

## Обработчик смены статусов. Изменения статуса отчетной формы

Статусы отчетных форм определяются согласно таблице ниже (Таблица 49).

Таблица 49 – Статусы отчетных форм

| **Статус** | **Численный эквивалент (StatusNumber)** | **Описание** |
| --- | --- | --- |
| Пусто | 0 | Данные отсутствуют |
| Черновик | 1 | Данные в процессе заполнения |
| Заполнено | 2 | Данные заполнены |
| Проверено | 3 | Данные проверены |
| Экспертиза | 4 | Данные проходят экспертизу |
| Утверждено | 5 | Данные утверждены |

Подробнее переходы статусов описаны в таблице ниже (Таблица 50).

Таблица 50 – Переходы статусов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Возможности перехода  из статуса в статус  и условия успешности перехода | | В | | | | | |
|  | | Пусто | Черновик | Заполнено | Проверено | Экспертиза | Утверждено |
| ИЗ | Пусто | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-2) | – | – | – | – |
| Черновик | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-3) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-4) | – | – | – |
| Заполнено | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-3) | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-5) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-6) | – | – |
| Проверено | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-3) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-7) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-8) | – |
| Экспертиза | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-3) | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-9) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-10) | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-11) |
| Утверждено | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-3) | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-12) | – | – | [+](file:///C:\Users\frolovicheva\Downloads\Telegram%20Desktop\Инструкция_Своды_Света.docx#scroll-bookmark-13) | – |

Обработчик смены статуса формы содержит несколько виртуальных методов, которые можно будет реализовать при наследовании.

### Методы

#### GetBeforeChangeStatusReportAsync

|  |
| --- |
| public virtual ValueTask<BeforeChangeFormStatusReport> GetBeforeChangeStatusReportAsync(ChangeFormStatusOperationArgs operationArgs)  {  return new ValueTask<BeforeChangeFormStatusReport>(new BeforeChangeFormStatusReport(false));  } |

Метод получает отчет операции изменения статуса форм, возвращает отчет операции изменения статуса форм и признак отмены операции.

Свойства типа входного параметра «ChangeFormStatusOperationArgs» (параметры выполнения операции смены статуса форм) представлены ниже (Таблица 51).

Таблица 51 – Свойства типа входного параметра «ChangeFormStatusOperationArgs»

| **Название свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| Forms | IEnumerable<FormStatusChangeDto> | Список DTO форм, у которых изменяется статус |
| NewStatus | IReportFormStatus | Новый статус |

Свойства типа «FormStatusChangeDto» (DTO формы, у которой изменяется статус) представлены ниже (Таблица 52).

Таблица 52 – Свойства типа «FormStatusChangeDto»

| **Название свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| FormIdentifier | ИдентификаторДанныхФормы | Идентификатор хранимых данных формы |
| CurrentStatus | IReportFormStatus | Текущий статус данных формы |

Свойства типа «IReportFormStatus» (интерфейс «DTO» статуса данных отчетной формы) представлены ниже (Таблица 53).

Таблица 53 – Свойства типа «IReportFormStatus»

|  |  |  |
| --- | --- | --- |
| Название свойства | Тип | Описание |
| Name | string | Наименование статуса данных отчетной формы |
| StatusNumber | int | Номер статуса данных отчетной формы |

Свойства типа атрибута выходного параметра «BeforeChangeFormStatusReport» (Результат получения отчета об операции изменения статуса форм) представлены ниже (Таблица 54).

Таблица 54 – Свойства типа атрибута выходного параметра «BeforeChangeFormStatusReport»

| **Название свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| Report | РезультатыВыполненияСверкиДанных | Отчет операции изменения статуса форм |
| CancelOperation | bool | Признак отмены операции изменения статуса форм |

#### ДоСменыСтатуса

Метод «ДоСменыСтатуса» – это метод, вызываемый до изменения статуса данных формы:

|  |
| --- |
| public virtual void ДоСменыСтатусаФормы(АргументыДоСменыСтатусаФормы аргументы)  {  } |

Свойства типа входного параметра «АргументыДоСменыСтатусаФормы» представлены в таблице ниже (Таблица 55).

Таблица 55 – Свойства типа входного параметра «АргументыДоСменыСтатусаФормы»

| **Название свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| ИдентификаторДанных | ИдентификаторДанныхФорм | Идентификатор данных форм |
| OldStatus | IReportFormStatus | Старый статус данных отчетной формы |
| NewStatus | IReportFormStatus | Новый статус данных отчетной формы |
| Пользователь | Пользователь | Пользователь, изменивший статус |
| Отменить | bool | Флаг отмены изменения |
| ThrowIfCancelled | bool | Генерировать исключение в случае отмены статуса |
| КомментарийОтмены | string | Комментарий в случае отмены |

#### ПослеСменыСтатуса

Метод «После СменыСтатуса» – это метод, вызываемый после изменения статуса данных формы:

|  |
| --- |
| public virtual void ПослеСменыСтатусаФормы(АргументыПослеСменыСтатусаФормы аргументы)  {  } |

Свойства типа входного параметра «АргументыПослеСменыСтатусаФормы» представлены ниже (Таблица 56).

Таблица 56 – Свойства типа входного параметра «АргументыПослеСменыСтатусаФормы»

|  |  |  |
| --- | --- | --- |
| **Название свойства** | **Тип** | **Описание** |
| ИдентификаторДанных | ИдентификаторДанныхФорм | Идентификатор данных форм |
| OldStatus | IReportFormStatus | Старый статус данных отчетной формы |
| NewStatus | IReportFormStatus | Новый статус данных отчетной формы |
| Пользователь | Пользователь | Пользователь, изменивший статус |

### ОбработчикСменыСтатусаФормыПоУмолчанию

ОбработчикСменыСтатусаФормыПоУмолчанию определен в классе «БазовыйПроцессСдачиОтчетности» и содержит определенный метод «ПослеСменыСтатуса». В данном методе содержится обработка, которая выполняется после того, как статус формы был изменен. Код метода содержит в себе уведомление пользователей. Оно выполняется через центр сообщений, а также дублируется по электронной почте тем операторам, которые связаны с отчетной формой и у которых в атрибутах заполнена графа «E-mail». Уведомление может отправляться при смене статуса данных формы:

* из «Заполнено» в «Проверено»;
* из «Проверено» в «Экспертиза»;
* из «Экспертиза» в «Утверждено»;
* из «Экспертиза» в «Черновик».

Если нет дополнительно определенного обработчика смены статуса формы, то при смене статуса формы будет создан ОбработчикСменыСтатусаФормыПоУмолчанию и будет выполнен метод «ПослеСменыСтатуса».

## Скрытие пунктов основного меню в зависимости от каких-то условий

Подробнее реализация кастомных js-файлов описана в п. 28.11.

Чтобы скрыть пункты меню, переопределите Svody.view.main.navigation.ToolbarController метод «afterRender»:

|  |
| --- |
| Ext.define('Svody.apiJs.overrides.ToolbarController', {    override: 'Svody.view.main.navigation.ToolbarController',    afterRender: function () {  this.callParent();    let view = this.getView();  let toolbarParameters = view.navigationParams;    let analytics = view.down('#analytics');    //если кнопка "Аналитические выборки" доступна, то скрываем ее  if (toolbarParameters.hasAccessToAnalytics) {  analytics.setVisible(false);  }  }  }  ) |

Для работы «override[[1]](#footnote-1)» укажите путь до папки с файлами в файле **src\BARS.Svody.Web.ExtJsClient\app.json** в тег «overrides» (Рисунок 52).

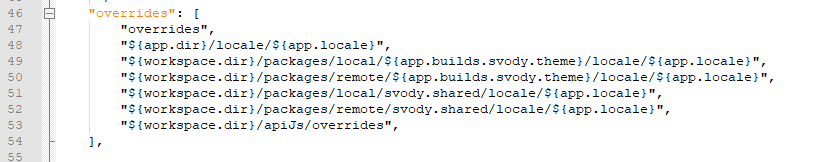


Рисунок 52 – Указание пути до папки с файлами

## Замена карточки учреждения в прикладных API

### Кастомизация инстанцирования формы Учреждения с помощью прикладных API проекта

Выполните следующие действия:

1. добавьте в прикладные API класс, реализующий интерфейс «IDepartmentFormService». Список методов и ожидаемая логика описаны в таблице ниже (Таблица 57);
2. добавьте в прикладные API класс, реализующий интерфейс «IWindsorInstaller» (если он еще не добавлен);
3. в класс, реализующий интерфейс «IWindsorInstaller», добавьте регистрацию необходимой реализации «IDepartmentFormService» в контейнере (с параметром «IsDefault()», lifestyle компонента обязательно Singleton!);

Такая регистрация полностью перекроет базовую реализацию в виде «DepartmentFormService» и Система будет обращаться к реализации в прикладных API.

Пример регистрации представлен ниже в таблице (Таблица 57).

Таблица 57 – Пример регистрации представлен ниже в таблице

| **Метод** | **Логика** |
| --- | --- |
| Описание интерфейса «IDepartmentFormService» | |
| BaseDepartmentForm CreateForm(DepartmentFormParamsDto formParamsDto); | Вернуть экземпляр наследника "BaseDepartmentForm" сконфигурированного в соответствии с настройками, переданными во входном параметре  Применение параметров ограничений возлагается на логику формы и усмотрение разработчика её реализующего.  В базовой реализации DTO параметров передаётся в качестве параметра конструктора класса «ФормаУчреждений». Конструктор, в свою очередь, вызывает метод «ApplyFormConfiguration()». Метод «ApplyFormConfiguration()» содержит непосредственную логику установки ограничений доступа к элементам пользовательского интерфейса (текстовым полям, кнопкам, спискам и т.д.) Формы Учреждений (объекта, наследуемого от «BaseDepartmentForm») |
| Описание типа DepartmentFormParamsDto | |
| public bool СodeFieldIsReadonly { get; set; } | Запрет редактирования поля Код |
| public bool NameFieldIsReadonly { get; set; } | Запрет редактирования поля Наименование |
| public bool FullNameFieldIsReadonly { get; set; } | Запрет редактирования поля Полное Наименование |
| public bool ShortNameFieldIsReadonly { get; set; } | Запрет редактирования поля Краткое Наименование |
| public bool InnFieldIsReadonly { get; set; } | Запрет редактирования поля ИНН |
| public bool KppFieldIsReadonly { get; set; } | Запрет редактирования поля КПП |
| public bool PfrFieldIsReadonly { get; set; } | Запрет редактирования поля ПФР |
| public bool OkonhFieldIsReadonly { get; set; } | Запрет редактирования поля ОКОНХ |
| public bool OkudFieldIsReadonly { get; set; } | Запрет редактирования поля ОКУД |
| public bool OkopfFieldIsReadonly { get; set; } | Запрет редактирования поля ОКОПФ |
| public bool OkinFieldIsReadonly { get; set; } | Запрет редактирования поля ОКИН |
| public bool OkvedFieldIsReadonly { get; set; } | Запрет редактирования поля ОКВЭД |
| public bool BranchNumberFieldIsReadonly { get; set; } | Запрет редактирования поля Филиал |
| public bool OkpoFieldIsReadonly { get; set; } | Запрет редактирования поля ОКПО |
| public bool OkatoFieldIsReadonly { get; set; } | Запрет редактирования поля ОКАТО |
| public bool OkoguFieldIsReadonly { get; set; } | Запрет редактирования поля ОКОГУ |
| public bool OkfsFieldIsReadonly { get; set; } | Запрет редактирования поля ОКФС |
| public bool OktmoFieldIsReadonly { get; set; } | Запрет редактирования поля ОКТМО |
| public bool BcChaptersTableIsReadonly { get; set; } | Запрет редактирования таблицы Глава По БК |
| public bool PersonnelTabIsReadonly { get; set; } | Запрет редактирования вкладки Персоналии |
| public bool AttributesTabIsReadonly { get; set; } | Запрет редактирования вкладки Атрибуты |
| public bool LiquidationInfoTabIsReadonly { get; set; } | Запрет редактирования Сведения О Ликвидации |
| public bool AdditionalAttributesTabIsReadonly { get; set; } | Запрет редактирования таблицы Дополнительные Атрибуты |
| public bool PlacementTabIsReadonly { get; set; } | Запрет редактирования вкладки Местонахождение |
| public bool UserProfileSelectorIsReadonly { get; set; } | Запрет выбора Профиль пользователя |
| public bool ParentDepartmentSelectorIsReadonly { get; set; } | Запрет выбора Родительское учреждение |

Регистрация кастомной реализации IDepartmentFormService:

|  |
| --- |
| public class ApplicationComponentsInstaller : IWindsorInstaller  {  public void Install(IWindsorContainer container, IConfigurationStore store)  {  container.Register(Component.For<IDepartmentFormService>().ImplementedBy<DepartmentFormService>().IsDefault().LifestyleSingleton());  }  } |

Пример базовой реализации сервиса DepartmentFormService:

|  |
| --- |
| /// <summary> Содержит логику создания / инициализации Формы Учреждения. </summary>  class DepartmentFormService : IDepartmentFormService  {  /// <summary> Создать окно Учреждения. </summary>  /// <param name="formParamsDto"> Параметры Формы Учреждения. </param>  public BaseDepartmentForm CreateForm(DepartmentFormParamsDto formParamsDto)  {  return new ФормаУчреждений(formParamsDto);  }  } |

### Для сводов на .net core и выше

### Пошаговая инструкция замены карточки учреждения

Выполните следующие действия:

1. создайте контроллер, унаследованный от BaseEditableObjectController<T, VM>, где Т – модель карточки учреждения, а VM – ViewModel карточки учреждения[[2]](#footnote-2);

Пример кастомного контроллера карточки учреждения:

|  |
| --- |
| [RIAAuthorize]  [AccessPermission("Учреждение.Учреждение.Просмотр")]  public class CustomController: BaseEditableObjectController<Department, DepartmentViewModel>  {  public async Task<ActionResult> DeleteChildren(Guid id, bool withParent) {...}    public ActionResult GetChildren(Guid? departmentId, List<string> columns, bool allChildNodes, bool checkNodes, ConsoleContext context) {...}    public ActionResult GetDepartments(List<Guid?> departmentsIds) {...}    public ActionResult FindChildren(Guid childrenId) {...}    public ActionResult Search(List<string> columns, string filter, int start, int limit) {...}    public ActionResult Move(Guid departmentId, Guid? parentDepartmentId) {...}    protected override List<Department> GetEditableObjectsList() {...}  } |

1. замените контроллер в классе представления на кастомный;

Замена контроллера у представления (view) формы учреждения на кастомный:

|  |
| --- |
| (function () {  let dep = RIA.Globals.EORegistry.getClass('Department');  //Заменяем платформенный контроллер контроллером из API  dep.directController = 'Custom';  })(); |

**Примечание** – Пункты, приведенные ниже, выполняйте, если необходимо изменить модель карточки учреждения.

1. определите новый класс модели карточки учреждения, унаследуйте отBaseEditableStoredObject<Учреждение, DepartmentViewModel> и создайте все обязательные методы и поля по аналогии с реализацией Системы. Пример платформенный реализации можно посмотреть в классе «Department»;

Кастомный класс карточки учреждения:

|  |
| --- |
| public class CustomDepartmentCardModel: BaseEditableStoredObject<Учреждение, DepartmentViewModel> |

1. укажите в кастомном контроллере новый класс модели карточки учреждения;

Кастомный контроллер с кастомной моделью карточки учреждения:

|  |
| --- |
| public class CustomController: BaseEditableObjectController<CustomDepartmentCardModel, DepartmentViewModel> {...} |

1. переопределите представление карточки учреждения для отображение кастомной модели. Для этого переопределите класс «DepartmentForm»;

Переопределение представления карточки учреждения:

|  |
| --- |
| Ext.ns('RIA');    //Переопределение стандартной карточки  Ext.override(RIA.Management.DepartmentForm, {...}) |

## Логирование

### Хранение логов

Логи сохраняются по пути:

**\websvody\src\BARS.Svody.Web.Host\bin\NetCoreDebug\netcoreapp3.1\.logs**

В папках формата гггг-мм-дд сохранятся файлы:

* debug.log;
* errors.log;
* info.log;
* trace.log;
* userActionsMonitor.log.

Отдельно в папке «Quartz» хранятся логи связанные с работой данного сервиса.

### Конфигурационный файл Nlog

Конфигурационный файл Nlog расположен по пути: **\websvody\src\BARS.Svody.Web.Host\NLog.config** (Рисунок 53).



Рисунок 53 – Конфигурационный файл Nlog

#### Секция <targets>

<target name="trace-logfile" xsi:type="File" fileName="${basedir}/.logs/${shortdate}/trace.log" encoding="utf-8" layout="${layout}" />

Значение атрибутов в секции <targets>:

* «name» – название файла необходимое для организации правил, по которым будет вестись запись в текущий файл;
* «fileName» – относительный путь файла, в который будет производиться запись логов;
* «layout» – шаблон заполнения лога.

#### Шаблон формирования записи лога

<variable name="layout" value="${longdate}|${level:uppercase=true}|${logger}|${message} ${exception:format=tostring,stacktrace:separator=\*}" />

#### Основные маркеры подстановки

Основные маркеры подстановки:

* ${basedir} – возвращает базовую директорию вашего приложения
* ${shortdate} / ${longdate};
* ${uppercase:...};
* ${message} – под данный маркер подставляется сообщение, указанное в аргументных скобках методов;
* ${logger} – название класса, от которого поступило сообщение

#### Секция <rules>

В секции <rules> формируются правила, в которых для каждого уровня логирования назначается файл, в который будет производиться запись.

Уровни логирования:

* «Fatal» – тотальный сбой работоспособности;
* «Error» – типичная ошибка;
* «Warn» – нештатная ситуация, потенциальная проблема;
* «Info» – общая информация о работе службы или сервиса;
* «Debug» – запись масштабных переходов состояний;
* «Trace» – пошаговые записи процесса.

### Пример

Логирование формы:

|  |
| --- |
| namespace Барс.Своды  {  using ApiBI.BO;  using АргументыСобытийОтчетнойФормы;    using NLog; //подключение пространства имен NLog    public partial class МодульФормы\_0503125ПБС  {  //Необходимо создать объект Logger и присвоить ему наименование, которое будет отражаться в лог-файлах  private Logger \_logger = LogManager.GetLogger("НаименованиеЛогаФормы0503125ПБС");    public override void ОбработатьСобытиеОткрытияФормы(АргументыОткрытияФормы Аргументы)  {  //Далее у объекта Logger вызывается метод (в зависимости от уровня лога) и в параметрах передается сообщение  //Логирование форм производится в Info. Запись будет производиться в info.log  \_logger.Info($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Info");    //Приведены для примера  \_logger.Fatal($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Fatal");  \_logger.Error($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Error");  \_logger.Warn($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Warn");  \_logger.Debug($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Debug");  \_logger.Trace($"Событие открытия формы {Форма.МетаструктураФормы.Код} запись в Trace");  }  }  } |

Результат логирования представлен на рисунке (Рисунок 54).

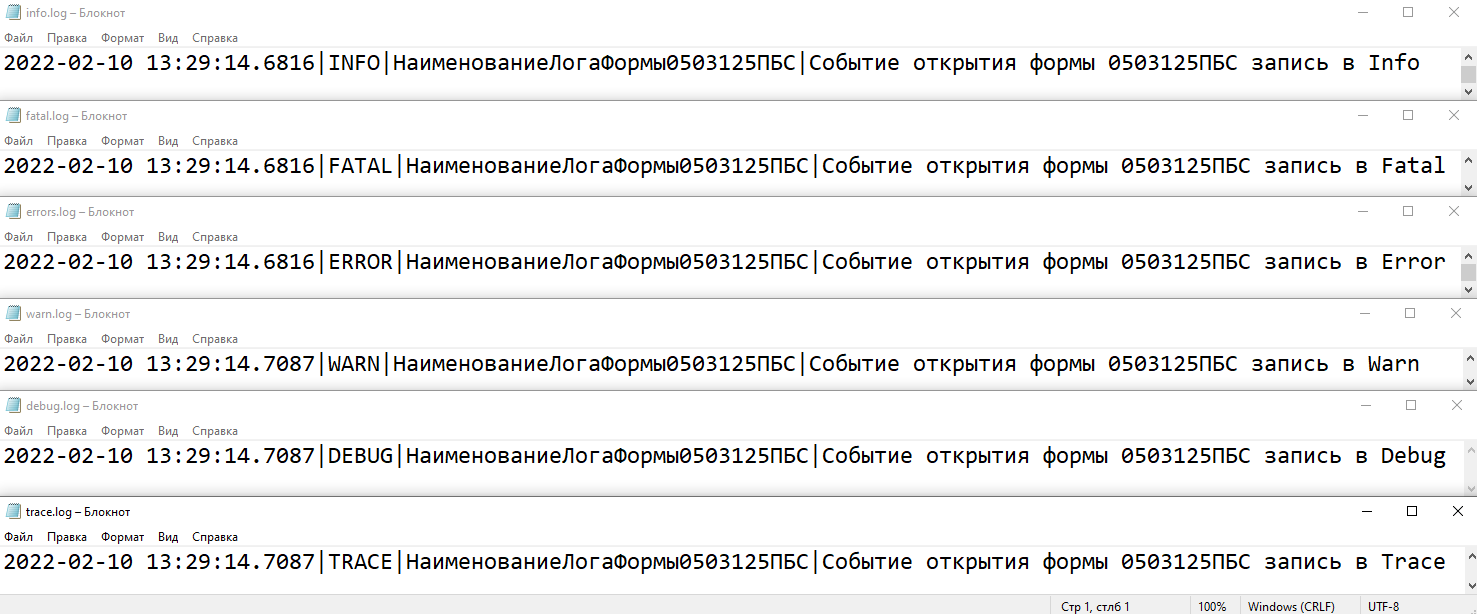


Рисунок 54 – Результат логирования

## Менеджер фоновых процессов Системы

Существующие API:

* IJobScheduler;
* BaseWorker,

оставлены без изменений, для новой реализации написаны адаптеры, позволяющие использовать новую реализацию для запуска старого кода фоновых задач.

**Примечание** – В данном разделе описаны адаптеры, позволяющие использовать новую реализацию для запуска старого кода фоновых задач.

### Новый интерфейс менеджера фоновых процессов

Интерфейс менеджера фоновых процессов:

|  |
| --- |
| namespace BARS.Svody.Job  {  using System;  using System.Collections.Generic;  using BARS.Svody.Job.Log;  using BARS.Svody.ProcessManager;  using Quartz;  /// <summary>Интерфейс менеджера процессов платформы</summary>  public interface IProcessManager  {  /// <summary>Добавить триггер Quartz с параметрами запуска процесса</summary>  /// <typeparam name="TJob">Тип процесса</typeparam>  /// <param name="processStartParams">Параметры запуска процесса</param>  /// <param name="listener">Обработчик событий триггера процесса</param>  /// <returns>Уникальный идентификатор процесса</returns>  Guid AddTriggerForJob<TJob>(ProcessStartParams processStartParams, ITriggerListener listener = null)  where TJob : ISvodyProcess;    /// <summary>Добавить триггер Quartz с параметрами запуска процесса</summary>  /// <param name="processType">Тип процесса</param>  /// <param name="processStartParams">Параметры запуска процесса</param>  /// <param name="listener">Обработчик событий триггера процесса</param>  /// <returns>Уникальный идентификатор процесса</returns>  Guid AddTriggerForJob(Type processType, ProcessStartParams processStartParams, ITriggerListener listener = null);    /// <summary>Получить список выполняющихся процессов</summary>  IEnumerable<JobViewModel> GetProcessesList();    /// <summary>Получить Dto с параметрами процесса</summary>  ProcessInfoDto GetProcessInfo(Guid processId);    /// <summary>Получить объект типа процесса</summary>  Type GetProcessInstanceType(Guid processId);    /// <summary>Получить лог выполнения процесса</summary>  IEnumerable<ILogRecord> GetProcessLog(Guid processId);    /// <summary>Получить результат выполнения процесса</summary>  TResult GetProcessResult<TResult>(Guid jobId);    /// <summary>Получить состояние выполнения процесса</summary>  JobViewModel GetProcessState(Guid triggerId);    /// <summary>Инициализировать экземпляр менеджера</summary>  void Init();    /// <summary>Запросить отмену всех процессов, у которых идентификатор владельца совпадает с заданным</summary>  /// <param name="ownerId">Идентификатор владельца</param>  void InterruptOwnerProcesses(Guid ownerId);    /// <summary>Запросить отмену выполнения процесса</summary>  void InterruptProcess(Guid processId);    /// <summary>Зарегистрировать тип фоновой операции</summary>  /// <typeparam name="TJob">Тип фоновой операции</typeparam>  void RegisterJobType<TJob>()  where TJob : ISvodyProcess;    /// <summary>Зарегистрировать тип фоновой операции</summary>  void RegisterJobType(Type jobType);    /// <summary>Остановить экземпляр менеджера</summary>  /// <param name="waitForJobsToComplete">Дождаться завершения выполняющихся процессов</param>  void ShutDown(bool waitForJobsToComplete = true);    С 4.2.15  /// <summary>Распланировать процесс по его идентификатору и типу</summary>  /// <param name="triggerInstanceId">Идентификатор процесса</param>  bool UnscheduleJob<TJob>(Guid triggerInstanceId);    С 4.2.15  /// <summary>Распланировать процесс по его идентификатору и типу</summary>  /// <param name="triggerInstanceId">Идентификатор процесса</param>  Task<bool> UnscheduleJobAsync<TJob>(Guid triggerInstanceId);  } |

В реализации новых классов фоновых процессов вместо интерфейса «IJobScheduler» используется интерфейс «IProcessManager».

Для удаления задачи из планировщика используются новые методы в интерфейсе «IProcessManager - UnscheduleJob и UnscheduleJobAsync».

### Базовый класс фонового процесса и его интерфейс

Базовый класс фонового процесса:

|  |
| --- |
| public abstract class BaseSvodyProcess : ISvodyProcess  {  ...  }    /// <summary>Интерфейс фонового процесса Quartz</summary>  public interface ISvodyProcess : IInterruptableJob  {  /// <summary>Уникальный идентификатор процесса</summary>  Guid Id { get; set; }  /// <summary>Состояние выполнения процесса</summary>  JobViewModel State { get; }  /// <summary>Результат выполнения процесса</summary>  object Result { get; set; }  /// <summary>Получить лог выполнения процесса</summary>  IEnumerable<ILogRecord> GetProcessLog();  /// <summary>Запросить отмену выполнения c ожиданием фактического выполнения отмены</summary>  void InterruptProcess(bool noWait = false);  } |

При реализации класса фонового процесса выполните его наследование от «BaseSvodyProcess» и переопределите абстрактный метод.

Метод фоновой работы:

|  |
| --- |
| /// <summary>Метод, переопределение которого должно содержать бизнес-логику процесса</summary>  /// <param name="context">Контекст выполнения quartz</param>  /// <param name="ct">Cancellation token для проверки запроса отмены выполнения</param>  /// <returns></returns>  protected abstract object ExecuteInternal(IJobExecutionContext context, CancellationToken ct); |

### Класс для создания параметров запуска экземпляра фонового процесса (показан только интерфейс класса)

Параметры запуска процесса:

|  |
| --- |
| /// <summary>Параметры запуска фонового процесса</summary>  public sealed class ProcessStartParams  {  ...  /// <summary>Описание</summary>  public string JobDescription { get; set; }    /// <summary>Время запуска процесса</summary>  public DateTimeOffset? StartAt { get; set; }    /// <summary>Признак запуска как можно быстрее после постановки в очередь</summary>  public bool StartNow  {  get { return !StartAt.HasValue; }  }    /// <summary>Признак единичного/множественного выполнения</summary>  public bool SingleExecution { get; set; }    /// <summary>Делегат настройки расписания выполнения триггера фонового процесса</summary>  public Action<SimpleScheduleBuilder> ScheduleBuilderAction { get; set; }    public ProcessStartParams()  {  \_jobData = new Dictionary<string, object>();  }    /// <summary>Получить JobDataMap для Quartz</summary>  public JobDataMap GetJobData()  {  return new JobDataMap(\_jobData);  }    /// <summary>Метод вычисления хэша, по которому определяется уникальность конкретного экземпляра процесса</summary>  public string GetParamsHash()  {  ...  }    /// <summary>Установить параметр</summary>  /// <typeparam name="TParam">Тип параметра</typeparam>  /// <param name="key">Ключ параметра</param>  /// <param name="value">Значение параметра</param>  public void SetJobParam<TParam>(string key, TParam value)  {  ...  }  ...  } |

Вместо свойства «Interval» класса «ScheduleParameters» реализована более гибкая возможность настройки расписания выполнения триггеров процесса с помощью делегата «Action<SimpleScheduleBuilder> ScheduleBuilderAction». Если делегат не передан, фоновый процесс будет выполнен только один раз.

### Пример добавления триггера выполнения фонового процесса с помощью нового API

Пример добавления триггера:

|  |
| --- |
| var startParams = new ProcessStartParams();  startparams.JobDescription = "Тест";  startParams.SetJobParam("ExecutionOwner", new ExecutionOwner  {  Type = ExecutionOwnerType.System,  UserLogin = "\_Система"  });  var triggerId = ProcessManager.AddTriggerForJob<ProcessClass>(startParams); |

### Конфигурирование менеджера фоновых процессов с помощью web.config

Конфигурация:

|  |
| --- |
| <!-- Настройки менеджера фоновых процессов -->  <Bars.ProcessManager Culture="ru-RU" ThreadCount="5"></Bars.ProcessManager> |

## Пермиссии и роли пользователя

### Пермиссия. Определение, виды и реализация

Для ограничения или разрешения доступа к определенной функциональности пользователей в Системе реализован механизм ролей и связанных с ним пермиссий.

Пермиссия – логическая единица доступа к какому-либо ресурсу. Если у пользователя есть пермиссия, это значит, что у него есть допуск к действию определенному данной пермиссией.

Сервис, отвечающий за хранение пермиссий, реализует интерфейс «IPermissionsContainer».

Интерфейс «IPermissionsContainer»:

|  |
| --- |
| /// <summary> Контейнер пермиссий </summary>  public interface IPermissionsContainer  {  /// <summary> Зарегистрировать пермиссии </summary>  /// <param name="permissions">Пермиссии</param>  void RegisterPermissions(params Permission[] permissions);    /// <summary> Получить все зарегистрированные пермиссии </summary>  /// <returns> Все зарегистрированные в системе пермиссии </returns>  IEnumerable<string> GetAllPermissions();  } |

Класс, реализующий модель пермиссии – «Permission»*.* Отдельная пермиссия состоит из четырех частей: категория, подкатегория, код пермиссии и ключ. Ключ формируется как комбинация трех остальных частей пермиссии.

Сервис, для регистрации пермиссий реализует интерфейс «IPermissionsRegistrator»*:*

|  |
| --- |
| /// <summary> Контракт регистратора пермиссий объявленных в модуле </summary>  public interface IPermissionsRegistrator  {  /// <summary> Зарегистрировать пермиссии модуля в контейнере пермиссий </summary>  /// <param name="container">Контейнер пермиссий</param>  void Register(IPermissionsContainer container);  } |

Отображаемые пермиссии создаются в модулях приложения, в том числе пермиссии можно создавать в прикладных API (Рисунок 55).

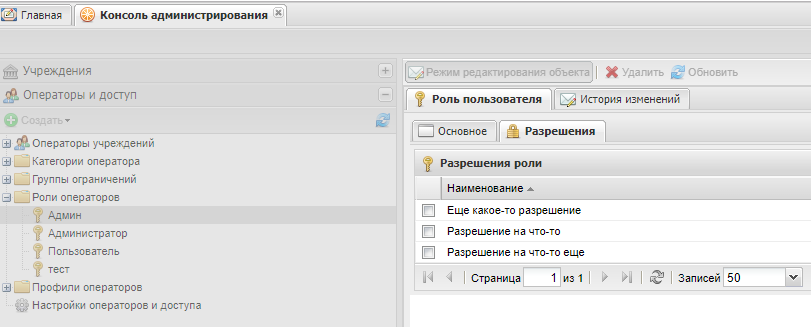


Рисунок 55 – Отображаемые пермиссии

#### Существующие в Системе пермиссии

По умолчанию в Системе регистрируются пермиссии, представленные в таблице ниже (Таблица 58).

Таблица 58 – Пермиссии, регистрирующиеся по умолчанию

| Категория | Подкатегория | Код |
| --- | --- | --- |
| Учреждение | Учреждение | * Просмотр * Добавление * Редактирование * Удаление |
| ВидыУчреждений |
| ТипыУчреждений |
| ТерриториальноеОтношение |
| ДополнительныеАтрибуты | * Просмотр * Редактирование |
| ОператорыДоступ | ОператорыУчреждений | * Просмотр * Добавление * Редактирование * Удаление |
| КатегорииОператоров |
| ГруппыОграничений |
| РолиОператоров |
| ПрофилиОператоров |
| НастройкаОператоровДоступ | * Просмотр * Редактирование |
| РегиональныеСправочники | РегиональныеСправочники | * Просмотр * Добавление * Удаление |
| ОписателиАтрибутов | * Просмотр |
| ЗаписьСправочника | * Просмотр * Добавление * Редактирование * Удаление |
| Справочники | Справочники | * Просмотр * Добавление * Редактирование * Удаление |
| ЗаписьСправочника |
| ОписателиАтрибутов |
| СдачаОтчетности | СписокОтчетныхПериодов | * Просмотр * Добавление * Редактирование * Удаление |
| КомпонентОтчетногоПериода |
| ЦепочкиСдачиОтчетности | СписокЦепочекСдачиОтчетности | * Просмотр * Добавление * Редактирование * Удаление |
| СписокЭлементовЦепочки |
| ПакетыОтчетныхФорм | Списокпакетов | * Просмотр * Добавление * Редактирование * Удаление |
| АналитическиеВыборки | СписокАналитическихВыборок | * Просмотр |
| НастройкиАналитическихВыборок | * Просмотр * Добавление * Редактирование * Удаление |
| ПользовательскиеУвязки | ПользовательскиеУвязки | * Просмотр * Добавление * Редактирование * Удаление |
| Метаописание форм | СписокХранимыхМетаописанийОтчетныхФорм | * Просмотр * Добавление * Удаление |
| ИсторияМетаописаний | * Просмотр * Скачивание |
| СрокиСдачиОтчетности | СрокиСдачиОтчетности | * Просмотр * Добавление * Редактирование * Удаление |
| ЖурналСроковСдачиОтчетности | * Просмотр |
| ФИАС | НастройкиФИАС | * Просмотр * Добавление * Редактирование * Перезапуск обновления |
| Прочее | НастройкиПриложения | * Просмотр * Редактирование |
| НастройкиПланировщика |
| ДополнительныеПризнакиОтчетныхФорм | * Просмотр * Добавление * Редактирование * Удаление |
| НастройкиИсторииИзменений | * Просмотр * Добавление * Редактирование |
| Пользователь | МойПрофиль | * Просмотр * Редактирование |
| МоеУчреждение |
| МоиХранимыеБлокировки | * Просмотр * Разблокировать |
| Дизайнер | ОтчетныеФормы | * Создание отчетных форм |

Итоговый ключ пермиссии формируется по шаблону:

**Категория.Подкатегория.Код**

#### Задание собственных пиктограмм пермиссий, отображаемых имен и списка разрешенных кодов

Сервис, задающий пиктограммы для пермиссий для десктопной версии и веб-версии, отображаемое имя для категорий и подкатегорий, список полных ключей разрешенных пермиссий, включенные коды пермиссий в Системе реализуется классом «PermissionMapper», файлы с ресурсами для маппера расположены по пути: **\src\Ядро\Resources**.

Чтобы задать свою версию такого маппера, воспользуйтесь интерфейсом «ICustomPermissionMapper»:

|  |
| --- |
| /// <summary>Интерфейс кастомного маппера для получения данных по категории пермиссии</summary>  public interface ICustomPermissionMapper  {  /// <summary>Возвращает иконку категории в base64 для Win клиента</summary>  /// <param name="category">Ключ категории</param>  /// <returns>Иконка в base64</returns>  string GetWinIcon(string category);    /// <summary>Возвращает класс иконки для Web клиента</summary>  /// <param name="category">Ключ категории</param>  /// <returns>Класс иконки</returns>  string GetWebIcon(string category);    /// <summary>Возвращает список включенных полных кодов пермиссий</summary>  /// <param name="fullKey">Полный ключ пермисии</param>  /// <returns>Список включенных полных кодов пермиссий</returns>  List<string> GetIncludePermission(string fullKey);    /// <summary>Возвращает список включенных кодов разрешений</summary>  /// <param name="permissionCode">Код разрешения</param>  /// <returns>Список включенных кодов разрешений</returns>  List<string> GetPermissionIncludeCodes(string permissionCode);    /// <summary>Возвращает отображаемое имя категории пермиссии</summary>  /// <param name="categoryName">Имя категории</param>  /// <returns>Отображаемое имя</returns>  string GetPermissionCategoryDisplayName(string categoryName);    /// <summary>Возвращает отображаемое имя подкатегории пермиссии</summary>  /// <param name="subCategoryName">Имя подкатегории</param>  /// <returns>Отображаемое имя</returns>  string GetPermissionSubCategoryDisplayName(string subCategoryName);  } |

Все пермиссии заданные в Системе используются для составления экземпляра «МатрицаПравПользователя», который инкапсулирует все пермиссии, определенные для пользователя и логику по проверке/добавлению/удалению пермиссий. При добавлении ограничения пользователю, из его матрицы прав удаляется соответствующая пермиссия.

Матрица прав определяется для пользователя с помощью функции ролей и некоторых вспомогательных классов.

#### Создание пермиссии

Для создания пермиссии выполните следующие действия:

1. создайте класс, реализующий интерфейс «IPermission»:

|  |
| --- |
| namespace Барс.RolePermissions  {  /// <summary> Пермиссия </summary>  public interface IPermission  {  /// <summary> Ключ пермиссии </summary>  string Key { get; }    /// <summary> Отображаемое наименование пермиссии </summary>  string Name { get; }  }  } |

Данные экземпляры являются DTO для отображения в консоли администрирования.

**Примечание** – Ключ пермиссии (Key) должен быть уникальным во всем приложении.

Пример реализации:

|  |
| --- |
| namespace Барс.RolePermissions  {  public class Permission1 : IPermission  {  public string Key => "1";  public string Name => "Разрешение на что-то";  }    public class Permission2 : IPermission  {  public string Key => "2";  public string Name => "Разрешение на что-то еще";  }    public class Permission3 : IPermission  {  public string Key => "3";  public string Name => "Еще какое-то разрешение";  }  } |

1. создайте класс-регистратор пермиссий, реализующий интерфейс «IPermissionsRegistrator»:

|  |
| --- |
| namespace Барс.RolePermissions  {  /// <summary> Контракт регистратора пермиссий объявленных в модуле </summary>  public interface IPermissionsRegistrator  {  /// <summary> Зарегистрировать пермиссии модуля в контейнере пермиссий </summary>  /// <param name="container">Контейнер пермиссий</param>  void Register(IPermissionsContainer container);  }  } |

1. в методе «Register» зарегистрируйте пермиссии объявленные в данном модуле в контейнере пермиссий.

Пример реализации регистратора:

|  |
| --- |
| namespace Барс.RolePermissions  {  public class ModulePermissionsRegistrator : IPermissionsRegistrator  {  public void Register(IPermissionsContainer container)  {  container.RegisterPermissions(new Permission1(), new Permission2(), new Permission3());  }  }  } |

Для выполнения проверки, назначена ли пермиссия хоть одной роли пользователя реализованы статические методы:

* «МенеджерОграниченияРолей.IsCurrentUserHavePermission»:

/// <summary> Текущий пользователь имеет разрешение </summary>  
/// <param name="permissionKey">Ключ пермиссии</param>  
/// <returns>true - если текущий пользователь имеет разрешение</returns>  
public static bool IsCurrentUserHavePermission(string permissionKey)

* «МенеджерОграниченияРолей.IsUserHavePermission»:

/// <summary> Пользователь имеет разрешение </summary>  
/// <param name="userId">Идентификатор пользователя</param>  
/// <param name="permissionKey">Ключ пермиссии</param>  
/// <returns>true - если пользователь имеет разрешение</returns>  
public static bool IsUserHavePermission(Guid userId, string permissionKey)

### Роли пользователя

Пермиссии связываются с пользователями с помощью сущности «Роль».

Роль – сущность, включающая набор пермисий, который позже может быть установлен пользователю. Роли расположены в таблице БД «BARS\_USER\_ROLE».

Роль связывается с определенной пермиссией при помощи третьего отношения (ввиду связи Многое ко многим) «*RolePermission»*. Таблица в БД –«ROLE\_PERMISSION». Это отношение ставит в соответствие роли ключ пермиссии. При формировании экземпляра «МатрицаПравПользователя», определяется роль пользователя, затем выявляются пермиссии, связанные с этой ролью, которые образуют саму матрицу.

Для пользователя может быть определено несколько ролей. Пример роли в системе представлен ниже (Рисунок 56).

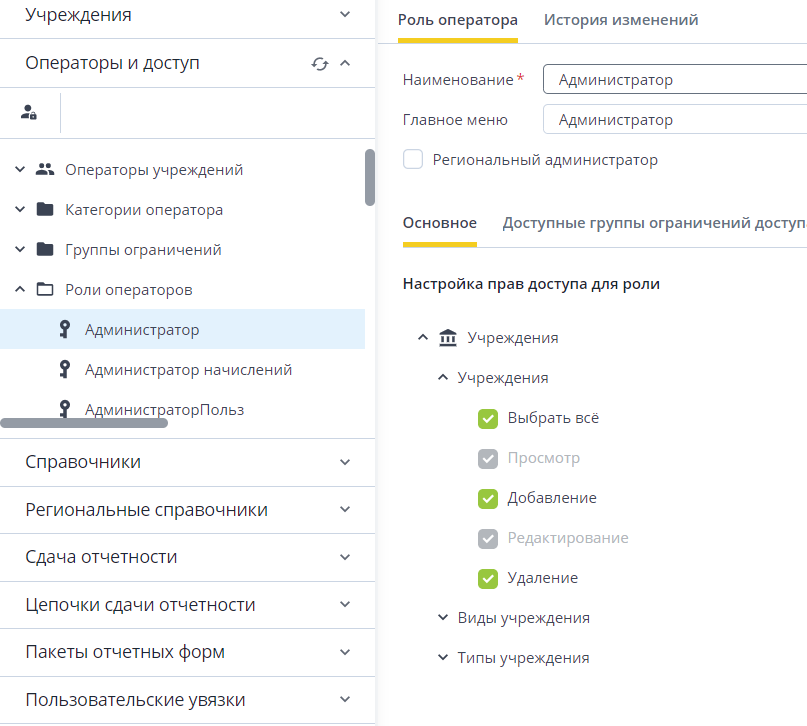


Рисунок 56 – Пример роли

Если пермиссии задаются исключительно в коде, роли могут задаваться внутри Системы.

На рисунке под строкой «Настройка прав доступа для роли» отображаются пермиссии связанные с этой ролью (см. Рисунок 56).

Основные категории отмечены пиктограммами, уровнем ниже располагаются субкатегории. Следующим уровнем расположены коды пермиссии, которые можно выбрать установкой «флажков»

**Примечание** – Настройки ролей могут изменяться.

Сервис, отвечающий за «ViewModel» пермиссий ролей реализует интерфейс «IRolePermissionsViewModel»:

|  |
| --- |
| /// <summary> ViewModel пермиссий ролей </summary>  public interface IRolePermissionsViewModel  {  /// <summary> Получить пермиссии по параметрам грида </summary>  /// <param name="roleId">Идентификатор роли</param>  /// <returns>Список пермиссий</returns>  PermissionsDto GetRolePermissions(Guid? roleId);    } |

За связь между ViewModel и свойствами ролей отвечает другой сервис, реализующий интерфейс «IRolePermissionsService»:

|  |
| --- |
| /// <summary> Сервис работы с пермиссиями ролей </summary>  public interface IRolePermissionsService  {  /// <summary> Получить назначенные роли пермиссии с ключами из списка</summary>  /// <param name="roleId">Идентификатор роли</param>  /// <param name="permissionKeys">Список ключей получаемых пермиссий</param>  /// <returns>Список назначенных роли пермиссий</returns>  IList<string> GetRolePermissions(Guid roleId, IEnumerable<string> permissionKeys);    /// <summary> Заполнить выбранные пермиссий роли</summary>  /// <param name="role">Роль</param>  /// <param name="checkedRolePermissions">Список выбранных пермиссий</param>  void FillCheckedPermission(Роль role, IEnumerable<RolePermissionDto> checkedRolePermissions);    /// <summary>Удалить невыбранные пермиссии роли </summary>  /// <param name="role">Роль</param>  /// <param name="checkedRolePermissions">Список выбранных пермиссий, на основе него формируется список не выбранных пермиссий</param>  void RemoveNonCheckedPermission(Роль role, IEnumerable<RolePermissionDto> checkedRolePermissions);    /// <summary> Получить назначенные роли пермиссии</summary>  /// <param name="roleId">Идентификатор роли</param>  /// <returns>Список назначенных роли пермиссий</returns>  IList<string> GetRolesPermission(Guid roleId);    /// <summary>Получить сгруппированные пермиссии по категории</summary>  /// <returns>Сгруппированные пермиссии по категориям</returns>  Dictionary<string, List<KeyHandler>> GetGroupingPermissionsByCategory();    /// <summary>Получить сгруппированные пермиссии категории по субкатегории</summary>  /// <param name="categoryValues">Пермиссии категории</param>  /// <returns>Сгруппированные пермиссии по субкатегориям</returns>  Dictionary<string, List<KeyHandler>> GetSubcategoryGroups(List<KeyHandler> categoryValues);  } |

#### Задание роли в Системе и установка ее пользователю

Создание списка ролей операторов в Системе осуществляется при помощи справочника «Роли операторов». Для перехода к справочнику выберите пункт «Администрирование/ Консоль администрирования» в главном меню. В открывшемся окне выберите пункт «Операторы и доступ/ Роли операторов». Чтобы создать роль, на панели инструментов окна со списком ролей нажмите на кнопку и выберите пункт «Создать» (Рисунок 57).

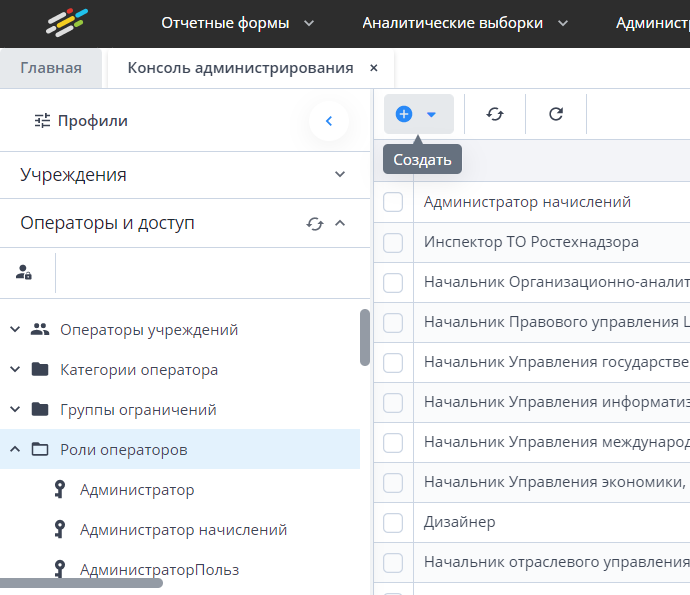


Рисунок 57 – Кнопка «Создать»

Откроется окно «Роль оператора» (Рисунок 58). В поле «Наименование» введите название роли. Название роли должно быть уникальным.

В поле «Главное меню» выберите значение из выпадающего списка. Выберите тип главного меню, которое будет отображаться в главном окне Системы:

* тип главного меню «Пользователь» – оператор сможет работать только с разделом отчетных форм и аналитических выборок. Оператору с ролью «Пользователь» доступны просмотр и заполнение отчетных форм, назначенных для заполнения учреждению, оператором которого он является, а также сбор аналитических выборок и сводных отчетов;
* тип главного меню «Администратор» – оператор с данной ролью сможет администрировать систему, имея доступ к разделу «Администрирование». Оператору с ролью «Администратор» доступна вся функциональность администрирования: заведение и редактирование профилей пользователей, настройка ролей, наложение ограничений, создание и редактирование справочников, используемых в Системе. Администратор по необходимости может выдать другим операторам права доступа к Системе.

На вкладке «Основное» задаются пермиссии роли.

Во вкладке «Доступные группы ограничения доступа» задаются группы ограничения для роли:

* на формы/компоненты/отчетные периоды. Ограничения могут быть наложены на следующие действия: проверка увязок форм, изменение статуса формы, редактирование сводной формы;
* на доступ к определенным элементам в форме, на аналитические выборки.

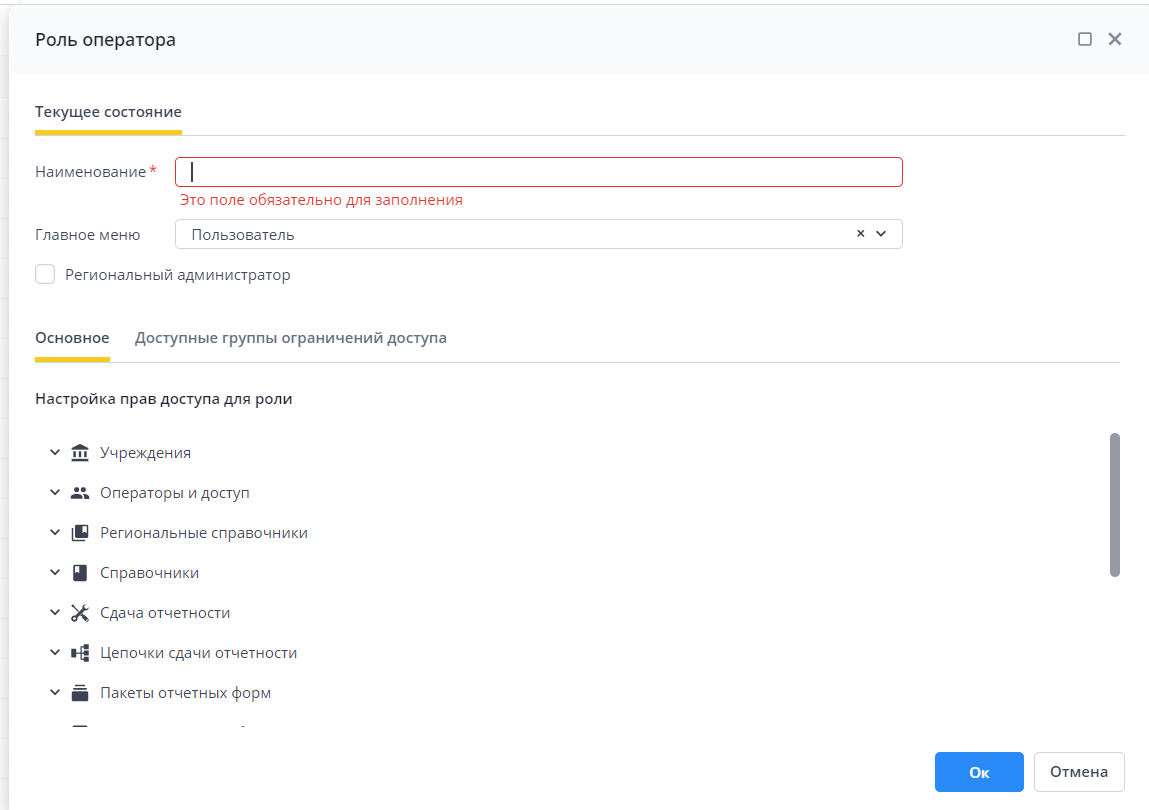


Рисунок 58 – Окно «Роль оператора»

Для назначения роли пользователю выберите пункт «Администрирование/ Консоль администрирования» в главном меню. В открывшемся окне выберите пункт «Операторы и доступ/ Операторы учреждений». Выберите учетную запись пользователя. На вкладке «Оператор учреждения» в блоке полей «Настройки доступа» выберите роль в поле «Роль» (Рисунок 59).

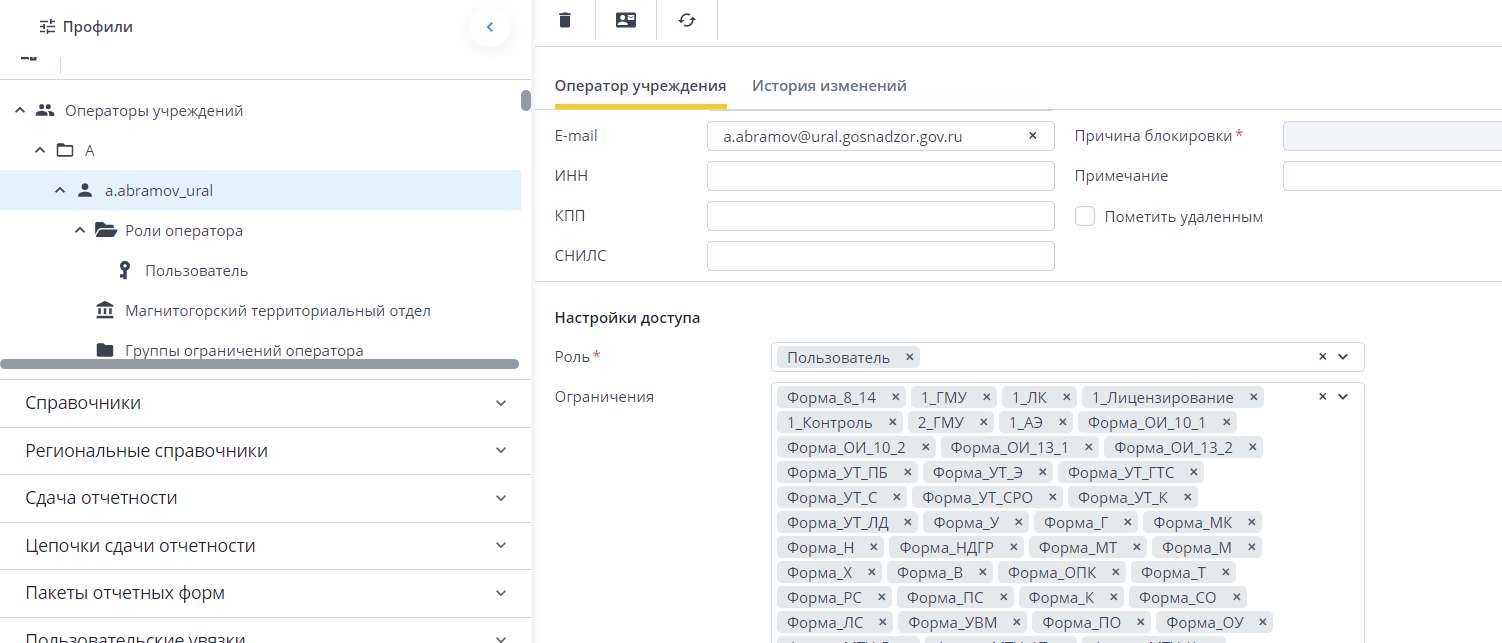


Рисунок 59 – Вкладка «Оператор учреждения» в карточке оператора

### Пример создания новой пермиссии

Для создания новой пермиссии выполните следующие действия:

1. задайте регистратор пермиссий IPermissionsRegistrator;

Пример контейнера:

|  |
| --- |
| using Барс.RolePermissions;    public sealed class TestPermission : IPermissionsRegistrator  {  public void Register(IPermissionsContainer container)  {  container.RegisterPermissions(new Permission("TestCathegory", "TestSubCathegory", "TestCode"));  }  } |

1. для модификации отображения пермиссии задайте маппер ICustomPermissionMapper:

|  |
| --- |
| using System.Collections.Generic;    using Барс.RolePermissions;    public class CustomMapper : ICustomPermissionMapper  {  public string GetWinIcon(string category)  {  return string.Empty;  }    public string GetWebIcon(string category)  {  return category == "TestCathegory"  ? "icon-test-permission"  : string.Empty;  }    public List<string> GetIncludePermission(string fullKey)  {  return new List<string> { "TestCathegory.TestSubCathegory.TestCode"};  }    public List<string> GetPermissionIncludeCodes(string permissionCode)  {  return new List<string>() {"ТestCode"};  }    public string GetPermissionCategoryDisplayName(string categoryName)  {  return categoryName == "TestCathegory"  ? "КастомКатегория"  : string.Empty;  }    public string GetPermissionSubCategoryDisplayName(string subCategoryName)  {  return subCategoryName == "TestSubCathegory"  ? "КастомСубКатегория"  : string.Empty;  }  } |

1. зарегистрируйте классы в контейнере:

|  |
| --- |
| container.Register(Component.For<IPermissionsRegistrator>().ImplementedBy<TestPermission>().LifestyleTransient());  container.Register(Component.For<ICustomPermissionMapper>().ImplementedBy<CustomMapper>().LifestyleTransient()); |

1. при необходимости задайте SCSS-класс с пиктограммой.

Кастомный стиль:

|  |
| --- |
| $test\_permission\_icon: "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16pt' height='16pt' viewBox='0 0 331 331'%3E%3Crect width='331' height='331' fill='%23C00'/%3E%3Cpath d='m165.49 15.47c30.28 10.787 58.307 27.796 81.948 49.566 22.675 21.132 41.915 47.019 51.072 76.93 6.2062 19.233 4.9292 40.017 0.82663 59.543-4.0982 18.189-12.101 35.528-23.755 50.119 13.217 12.923 26.307 25.974 39.558 38.863-1.7648 6.603-4.4693 13.351-9.8987 17.802-4.5399 3.9187-10.352 5.8133-16.072 7.2378-12.941-13.445-25.948-26.823-38.896-40.261-16.129 11.019-35.594 16.827-55.031 17.685-24.481 1.0821-48.954-6.8092-68.929-20.826-10.771-7.4027-20.48-16.254-29.387-25.794-1.5255 1.5006-3.1903 2.847-4.9022 4.1295 0.42452 3.2013 0.88448 6.3985 1.3259 9.6005-3.3426 0.19229-6.6797 1.0096-9.4884 2.8931-8.6657 5.4918-14.683 14.017-20.834 22.02-7.6271 9.9016-14.482 20.892-25.111 27.906-6.4414 3.9792-15.973 2.2974-19.918-4.3912-4.1259-6.5622-1.7458-15.032 2.6981-20.789 10.861-15.963 30.253-22.786 42.394-37.492 3.1577-3.9512 5.5457-8.5309 6.7303-13.461 3.101 0.19817 6.2023 0.39974 9.3044 0.60175 2.6274-2.6649 5.2677-5.3163 7.8851-7.9898 2.9183-0.0754 5.8923 0.23651 8.7736-0.32607 3.0076-2.6527 5.4052-5.9141 8.1214-8.8522 2.9751 0.8335 4.6657 3.6241 6.8187 5.617 13.307 12.12 28.525 22.682 45.869 28.068 22.179 7.1054 46.889 4.8481 67.694-5.4908-41.672-42.905-83.205-85.947-124.82-128.91-9.3317 9.8253-18.627 19.688-28.092 29.385-11.559-11.792-23.231-23.473-34.849-35.207 21.898-21.92 44.042-43.594 66.061-65.393 19.037 5.1825 38.049 10.456 57.103 15.577-13.021 11.821-26.257 23.403-39.323 35.173 43.419 42.566 86.622 85.358 130.11 127.86 10.498-12.773 16.407-29.037 17.21-45.504 1.4809-21.788-3.5608-43.463-10.743-63.901-10.152-26.973-29.562-49.179-50.658-68.282-13.185-11.72-26.774-23.002-40.801-33.701z' fill='%23ffd600'/%3E%3C/svg%3E";    .icon-test-permission{  background-image: url($test\_permission\_icon)  } |

Результат настройки представлен ниже на рисунке (Рисунок 60).

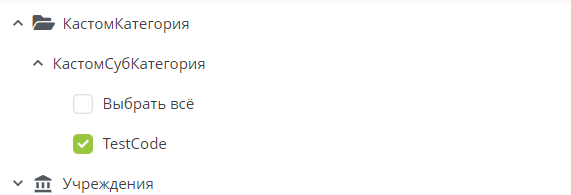


Рисунок 60 – Результат создания новой пермиссии

### Наложение пермиссий

Основные разрешения накладываются на учетную запись пользователя при его авторизации в Системе обращением на контроллер рабочего стола «DesktopContoller», метод «GetNavigationParameters».

Цепочка вызовов при проверке наличия пермиссии: «МенеджерПользователей.CheckPermission/ МенеджерОграниченийРолей.HasPermission/МатрицаПрав.HasPermission». Проверка выполняется по ключу пермисии.

На Front-Office проверка на наличие основных пермиссий: (АналитическиеВыборки, Дизайнер, Консоль администрирования, ЖурналСроковСдачиОтчетности) осуществляется проверкой свойств.

Отправление пермисий на Front-Office:

|  |
| --- |
| public ActionResult GetNavigationParameters()  {  return this.Direct(  new  {  hasAccessToViewAnalytics = AnalyticsAccess.HasAccessToAnalytics(),  hasAccessToEditAnalytics = AnalyticsAccess.HasAccessToAnalyticsSettings(),  hasAccessToDesigner = DesignerService.IsConfiguredProperly() && DesignerService.HasPermission(),  hasAccessToAdminConsole = ManagementConsoleTreeRulesRegistry.IsAnySectionAvailable(),  hasAccessToMonJournal = МенеджерПользователей.CheckPermission("СрокиСдачиОтчетности.ЖурналСроковСдачиОтчетности.Просмотр"),  hiddenParentMenuItems = Application.HiddenParentMenuItems,  hiddenMenuItems = Application.HiddenMenuItems,  });  } |

Получение и использование:

|  |
| --- |
| //пример функции  getNavigationParams: async () => await DesktopAsync.GetNavigationParameters(),  let navigationParams = await this.getNavigationParams();    //ограничение видимости  let toolbarParameters = view.navigationParams;    if (toolbarParameters.hasAccessToViewAnalytics ||  toolbarParameters.hasAccessToEditAnalytics){    analytics.setVisible(true);  analyticsList.setVisible(toolbarParameters.hasAccessToViewAnalytics);  analyticsSettings.setVisible(toolbarParameters.hasAccessToEditAnalytics);  }  else{  view.remove(analytics);  analytics.destroy();  } |

На Back-Office проверяется условие «МенеджерПользователей.CheckPermission» и в зависимости от результата, выполняется заданная логика.

Применение пермиссии на Back-Office:

|  |
| --- |
| if (МенеджерПользователей.CheckPermission("Категория.Подкатегория.Код"))  {  //логика  } |

## Инициализация веб-приложения

В Систему добавлена точка расширения логики инициализации веб-приложения.

В сборку BARS.Svody.Web.Init добавлен интерфейс «IAppInitService»:

|  |
| --- |
| /// <summary> Сервис дополнительной логики инициализации веб-приложения </summary>  /// <remarks> Реализуется в прикладных API </remarks>  public interface IAppInitService  {  /// <summary> Инициализировать </summary>  Task InitAsync();  } |

Создайте данный интерфейс и зарегистрируйте его в IOC-контейнере.

После основной логики инициализации приложения Система получит все реализации данного интерфейса и в цикле вызовет метод «Init».

**Примечания**

1 На всю дополнительную логику инициализации выделяется 60000 мс (1 минута). Это означает, что если существует несколько реализаций «IAppInitService», то общее время выполнения их методов «Init» не должно превышать 60000 мс (1 минута).

2 Если во время выполнения метода «Init» возникло необработанное исключение или инициализация завершилась по таймауту (60000 мс), то работа приложения будет экстренно завершена.

### Пример

Реализация интерфейса «IAppInitService»:

|  |
| --- |
| public class AppInitService : IAppInitService  {  public Task InitAsync()  {  //необходимая логика, например, запуск фонового процесса работающего по расписанию  }  } |

Регистрация в IOC-контейнере:

|  |
| --- |
| //ApplicationComponentsInstaller находится в прикладном API  public class ApplicationComponentsInstaller : IWindsorInstaller  {  public void Install(IWindsorContainer container, IConfigurationStore store)  {  container.Register(Component.For<IAppInitService>().ImplementedBy<AppInitService>().LifestyleSingleton());  }  } |

## Логин/ логаут

Существуют следующие способы авторизации в Системе:

* авторизация с вводом логина и пароля на соответствующей странице приложения;
* альтернативные способы входа, например с помощью кросс-авторизации.

В пунктах ниже рассмотрены все способы авторизации.

### Классический способ входа в приложение через страницу логина

Авторизация в Системе при таком способе осуществляется с помощью ввода логина и пароля от аккаунта в соответствующие поля формы (Рисунок 61). Данные отправляются на контроллер LoginController.

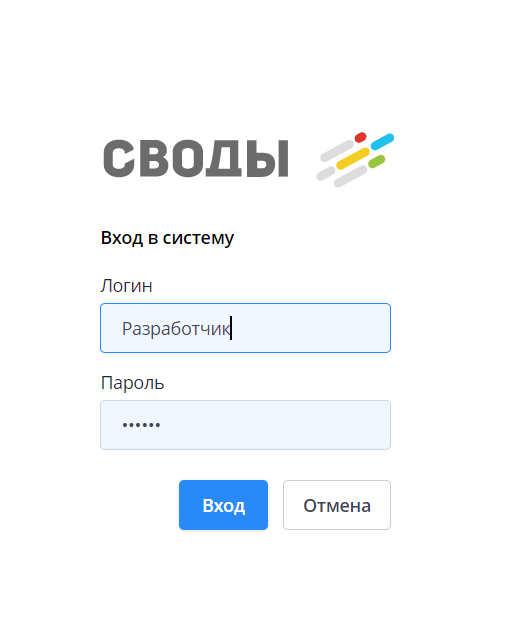


Рисунок 61 – Форма ввода данных для авторизации в Системе

**Примечание** – Пароли в Системе хранятся только в виде хеш, который вычисляется с помощью класса «МенеджераПользователей», метод «ComputePasswordHash».

Для входа в root:

* проверка осуществляется с помощью службы из IOC-контейнера, реализующей интерфейс «IControlConfigurationPanelManager»:

|  |
| --- |
| /// <summary>Интерфейс менеджера панели конфигурации управления</summary>  public interface IControlConfigurationPanelManager  {  /// <summary>Проверить доступ к управлению конфигурацией приложения</summary>  /// <param name="login">Логин пользователя</param>  /// <param name="password">Пароль пользователя</param>  /// <returns>Разрешен доступ или нет</returns>  bool CheckAccess(string login, string password);  } |

* данные по логину  в базовой реализации загружаются из svody.config, секция «Bars.ControlConfigurationPanel» (Рисунок 62).

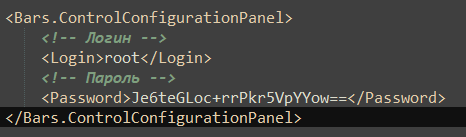


Рисунок 62 – Секция «Bars.ControlConfigurationPanel»

Для входа в приложение:

* сервис, отвечающий за аутентификацию и авторизацию, реализует интерфейс «IMembershipProvider»:

|  |
| --- |
| public interface IMembershipProvider  {  /// <summary>Смена пароля пользователя</summary>  /// <param name="userId">Идентификатор пользователя</param>  /// <param name="oldPassword">Старый пароль</param>  /// <param name="newPassword">Новый пароль</param>  void ChangePassword(Guid userId, string oldPassword, string newPassword);    /// <summary>Установить пароль пользователя</summary>  /// <param name="userId">Идентификатор пользователя</param>  /// <param name="newPassword">Пароль пользователя</param>  void SetPassword(Guid userId, string newPassword);    /// <summary>Установить признак смены пароля при следующем входе</summary>  /// <param name="userId">Идентификатор пользователя</param>  void SetUserPasswordChangeSign(Guid userId);    /// <summary>Авторизация пользователя по логину и паролю</summary>  /// <param name="model">Модель с данными для авторизации</param>  /// <returns>Результат авторизации</returns>  AuthorizationViewModel SignIn(AuthorizationViewModel model);    /// <summary>Обработать выход пользователя</summary>  void SignOut(АгентПодключения connectionAgent = null);    /// <summary>Сгененировать исключение если пароль не соответствует требованиям безопасности</summary>  /// <param name="userId">Идентификатор пользователя</param>  /// <param name="oldPassword">Старый пароль</param>  /// <param name="newPassword">Новый пароль</param>  void ThrowIfPasswordValidationFails(Guid userId, string oldPassword, string newPassword);    /// <summary>  /// Получить пользователя по логину  /// </summary>  /// <param name="логин">Логин</param>  /// <returns>Пользователь</returns>  Пользователь GetUserByLogin(string логин);    /// <summary>Установить всем пользователям признак смены пароля при входе</summary>  void SetChangePasswordSignToAllUsers();    /// <summary>Установить всем пользователям признак смены пароля при входе принудительно</summary>  void ForceChangePasswordSignToAllUsers();  } |

* проверка введенных данных в базовой реализации IMembershipProvider осуществляется с помощью службы, реализующей интерфейс «ISvodyAppAuthorizationModel»:

|  |
| --- |
| public interface ISvodyAppAuthorizationModel  {  /// <summary>Авторизация в системе с данными модели</summary>  /// <param name="authorizationViewModel">ViewModel авторизации</param>  /// <returns> Флаг успеха авторизации </returns>  bool Authorize(AuthorizationViewModel authorizationViewModel);  } |

* за сохранение параметров запуска приложения типа «РежимРазработчика» с привязкой к логину, отвечает сервис реализующий интерфейс «ILoginAppParams»:

|  |
| --- |
| /// <summary>  /// сохранение параметров запуска приложения, таких как /РежимРазработчика, с привязкой к логину в Ioc т.к. в  /// случае web-приложения экземпляр Барс.Клиент.Приложение.Параметры один на всех пользователей  /// </summary>  public interface ILoginAppParams  {  /// <summary>Проверка параметра вохода пользователя</summary>  /// <param name="login">Логин</param>  /// <param name="paramName">ИмяПараметра</param>  /// <returns>Признак наличия параметра</returns>  bool LoginParamEnabled(string login, string paramName);    /// <summary>Установить параметры для логина</summary>  /// <param name="login">Логин</param>  /// <param name="loginParams">Параметры входа</param>  void SetLoginParams(string login, IEnumerable<string> loginParams);  } |

Если проверка логина, пароля, а так же установленной лицензии прошла успешно, то в http-сессию установятся необходимые для входа данные и пользователь успешно будет авторизован, иначе будет выведено сообщение об ошибке.

Сервис, отвечающий за содержание новостей (новостной ленты) на странице логина, реализует интерфейс «INewsService»:

|  |
| --- |
| /// <summary>интерфейс сервиса новостей</summary>  public interface INewsService  {  /// <summary>Получить актуальные новости проекта</summary>  /// <returns>html строка с новостями</returns>  Task<string> GetActualNewsAsync();    /// <summary>Получить сохраненые новости проекта</summary>  /// <returns>html строка с новостями</returns>  Task<string> GetSavedNewsAsync();    /// <summary>Сохранить новости проекта</summary>  /// <param name="news">Html строка новостей</param>  Task SaveNewsAsync(string news);    /// <summary>Обновить новости проекта</summary>  Task UpdateNewsAsync();  } |

#### События и сервисы при логине/логауте используемые в базовой реализации ISvodyAppAuthorizationModel

При авторизации можно определить различные события, срабатывающие до, после и во время неудачной авторизации.

Доступные события:

* «ПередИнициализациейПриложения»;
* «ПослеУспешнойИнициализацииПулаСоединений»;
* «ПриНеуспешнойИнициализацииПулаСоединений»;
* «ПередАвторизациейПользователя»;
* «ПослеУспешнойАвторизации»;
* «ПриНеуспешнойАвторизации»;
* «ПередНачаломЦиклаОжидания»;
* «ПриЗакрытииПриложения»;
* «BeforeLogout».

Чтобы определить такие события, создайте класс, наследующий «СобытияИнициализацииПриложения» и в наследнике переопределите необходимое событие.

События при авторизации:

|  |
| --- |
| public class СобытияИнициализацииПриложения  {  /// <summary>  /// Возникает при передачи управления в ядро прикладной подсистемы для выполнения  /// процедуры инициализации внутренних структур.  /// </summary>  public virtual void ПередИнициализациейПриложения()  {  }    /// <summary>  /// Возникает после успешной инициализации пула соединения, т.е. когда  /// удалось установить первичное соединение с сервером базы данных  /// </summary>  public virtual void ПослеУспешнойИнициализацииПулаСоединений()  {  }    /// <summary>  /// Возникает в случае, если не удалось инициализировать пул соединений, т.е.  /// когда не получилось установить первичное соединение с сервером базы данных  /// (например, сервер БД недоступен)  /// </summary>  public virtual void ПриНеуспешнойИнициализацииПулаСоединений()  {  }    /// <summary>  /// Возникает каждый раз перед попыткой авторизации пользователя прикладной подсистемы  /// </summary>  public virtual void ПередАвторизациейПользователя()  {  }    /// <summary>  /// Возникает в случае, если авторизация пользователя прикладной подсистемы  /// выполнена успешно  /// </summary>  public virtual void ПослеУспешнойАвторизации(ЛицензионныйКлюч license)  {  }    /// <summary>  /// Возникает каждый раз, когда попытка аторизации пользователя (т.е. сравнение  /// переданного логина и пароля пользователя неуспешно)  /// </summary>  public virtual void ПриНеуспешнойАвторизации()  {  }    /// <summary>  /// Возникает, когда приложение переходит в цикл ожидания выбора пользователем  /// пункта главного меню.  /// </summary>  /// <remarks>На данный момент не реализовано</remarks>  public virtual void ПередНачаломЦиклаОжидания()  {  }    /// <summary>  /// Возникает каждый раз, когда пользователь закрывает приложение  /// </summary>  public virtual void ПриЗакрытииПриложения()  {  }    /// <summary>Возникает каждый раз, когда пользователь выходит из системы</summary>  public virtual void BeforeLogout(Пользователь logoutUser)  {  } |

Чтобы обработать событие при успешной авторизации с возможностью отмены, создайте сервис, реализующий интерфейс «IAuthenticationHandler»:

|  |
| --- |
| public interface IAuthenticationHandler  {  /// <summary>Обработать успешную аутентификация, с возможностью отмены</summary>  /// <param name="authenticatedUser">Авторизованный пользователь</param>  /// <returns>Ошибка</returns>  string OnAuthenticateSuccess(Пользователь authenticatedUser);  } |

Сервис прокси для авторизации в Active Directory (только для реализации в API, .Net Core не поддерживает Active Directory) реализует интерфейс «IAdAuthenticationProxy»:

|  |
| --- |
| public interface IAdAuthenticationProxy  {  /// <summary> Получить результат авторизации в AD </summary>  /// <param name="login">Логин пользователя</param>  /// <param name="password">Пароль пользователя</param>  /// <param name="noPasswordCheck">Флаг необходимости провери пароля</param>  bool AuthenticateInAd(string login, string password, bool noPasswordCheck);  } |

#### Сервис обработки ограничений на количество попыток входа

Сервис обработки ограничений на количество попыток входа реализует интерфейс «IAuthenticationLimitService»:

|  |
| --- |
| /// <summary>Интерфейс сервиса обработки лимитированного количества попыток аутентификации</summary>  public interface IAuthenticationLimitService  {  /// <summary>Обнулить количество попыток аутентификации пользователя</summary>  /// <param name="user">Пользователь</param>  void ResetCounter(Пользователь user);    /// <summary>Проверяет и обнуляет количество попыток аутентификации если прошло 30 минут</summary>  /// <param name="user">Пользователь</param>  bool TryResetCounter(Пользователь user);    /// <summary>Увеличивает количество попыток аутентификации и проверяет необходимо ли заблокировать доступ</summary>  /// <param name="user">Пользователь</param>  bool AuthenticationFailCountIsOver(Пользователь user);  } |

### Кроссплатформенная авторизация

Кроссплатформенная авторизация может осуществляться различными способами: с помощью встроенных моделей или с помощью собственного контроллера.

#### Open ID Connect / Keycloak

Описание представлено в п. 28.9.5

#### Авторизация по заголовкам запросов

Описание представлено в п. 28.9.3.

#### Кросс-авторизация (на платформенных сущностях)

Для реализации кросс-авторизации, использующей платформенные сущности, создайте собственную реализацию интерфейса «IAuthorizationModel» и зарегистрируйте его Ioc/DI контейнере:

|  |
| --- |
| public interface IAuthorizationModel  {  ILogger Logger { get; set; }  IWindsorContainer Container { get; set; }  string GetCrossAuthHash(string login);  void SetUserPasswordChangeSign(Пользователь user, int passwordStrength);  bool Authorize(AuthorizationViewModel authorizationViewModel, bool noPasswordCheck = false);  bool CrossAuthorize(string login, string hash, IIdentity userIdentity);  bool AuthenticateUser(IHeaderDictionary headers, string openAmSessionId);  SsoType GetSsoType { get; }  } |

Вход в систему данным способом описан в п. 28.9.3.

#### Кросс-авторизация (собственный контроллер

Если логика, требуемая при кросс-авторизации, не укладывается в платформенные сущности, то можно написать свой контроллер.

Пример контроллера:

|  |
| --- |
| //получаем модель авторизации  public IAuthorizationModel AuthModel { get; set; }    public ActionResult LoginCross(/\*требуемые параметры для входа, можно обработать как get, так и post, зависит от задачи\*/)  {  //добавляем в http кэше сессии информацию для входа  RIASession.AddUserSession();    if (/\*проверяем данные для авторизации, способ зависит от задачи, условие необязательно\*/)  {  //создаем модель для данных авторизации, тут же указывается ХЭШ пароля, если авторизации происходит с ним.  var authViewModel = new AuthorizationViewModel  {  Login = userLogin  };  // авторизируемся с помощью соответствующей модели, в этом методе происходят внутрении проверки пользователя  // например, не заблокирован ли он, есть ли вообще такой пользователь итп.  if (AuthModel.Authorize(authViewModel, /\*флаг необходимости проверить хэш пароля в модели данных\*/ true))  {  //если успешно, то редирект в систему  BARS.Svody.Web.Legacy.HttpContext.Current.Session.IsAuthenticated = true;  ActiveConnectionsMonitor.UpdateUsedLicensesInfo(RIASession.GetUsedLicensesInfo());  return this.RedirectToAction("Index", "Desktop");  }    //если уже есть авторизованная сессия, то редирект в систему  if (BARS.Svody.Web.Legacy.HttpContext.Current.Session.IsAuthenticated)  {  return this.RedirectToAction("Index", "Desktop");  }  }    //иначе, http кэш сессии для системы зачищаем и редирект на страницу логина  RIASession.Clear();  return this.RedirectToAction("LoginPage", "Login");  } |

### Кросс-авторизация в Системе. Открытие формы во Frame

#### Открытие формы во Frame

Чтобы открыть форму, вызовите контроллер Home/Index.

**Пример**

**http://{app\_domain}/Home/Index?login=test&hash=OirlkeiBElwFIEfbbJ3X5A==&code=585&componentId=f3364424-3e27-43b1-a141-805e5cea0d36&departmentId=5b9f4390-0833-4998-b7b7-311932aa9f64&cellType=2&actualDate=**

где:

* app\_domain – адрес приложения Системы;
* login – логин пользователя в Системе (тип string);
* hash – хеш-код, генерируемый в стороннем приложении (тип string);
* code – код формы (тип string);
* componentId – id компонента отчетного периода (тип guid);
* departmentId – id учреждения (тип guid);
* cellType – тип элемента цепочки (тип int);
* actualDate – дата актуальности данных формы (тип DateTime?)

#### Пример генерации хеш

Хеш в Системе формируется следующим образом;

string.Format("{0} {1:dd.MM.yyyy HH:mm} cross\_auth", login, date)

где:

* login - логин пользователя в Системе (тип string);
* date - серверной время в БД.

Пример генерации hash в c#:

|  |
| --- |
| private string GetSvodyAuthentificationHashInternal(string login)  {  var serverDateTime = this.Container.Resolve<ISessionProvider>().GetCurrentSession().CreateSQLQuery("select sysdate from dual").UniqueResult<DateTime>();    var date = new DateTime(  serverDateTime.Year,  serverDateTime.Month,  serverDateTime.Day,  serverDateTime.Hour,  (serverDateTime.Minute / 10) \* 10,  0);    var targetStr = string.Format("{0} {1:dd.MM.yyyy HH:mm} cross\_auth", login, date);  byte[] bytes = Encoding.UTF8.GetBytes(targetStr);    return Convert.ToBase64String(new MD5CryptoServiceProvider().ComputeHash(bytes));  } |

Реализация кросс-авторизации в Системе:

|  |
| --- |
| var autorizationModel = this.Container.ResolveAll<IAuthorizationModel>();  if (autorizationModel.Any(am => am.GetSsoType == SsoType.Nothing))  {    // Кросс-авторизация с другими системами  if (autorizationModel.First(am => am.GetSsoType == SsoType.Nothing).CrossAuthorize(login, hash, Request.LogonUserIdentity))  {  if (formIdentifier == null || formIdentifier.ComponentId == System.Guid.Empty)  {  return this.RedirectToAction("Index", "Desktop");  }  else  {  var routedValueDictionary = new RouteValueDictionary();  routedValueDictionary["metaName"] = formIdentifier.Code;  routedValueDictionary["departmentId"] = formIdentifier.DepartmentId;  routedValueDictionary["componentId"] = formIdentifier.ComponentId;  routedValueDictionary["cellType"] = formIdentifier.ChainElementType;  routedValueDictionary["openFromOutside"] = true;  return this.RedirectToAction("Open", "Forms", routedValueDictionary);  }  }  // если пользователь уже залогинился, то переводим его на главную страницу приложения  if (МенеджерПользователей.ТекущийПользователь != null)  {  return this.RedirectToAction("Index", "Desktop");  }  }      var viewModel = new HomeViewModel()  {  PageTitle = ResourcesStrings.Главная,  FrameworkVersion = applicationInstance.FrameworkAssemblyVersion,  RIAVersion = applicationInstance.RIAAssemblyVersion,  ServerInfo = string.Format("{0}({1})", Барс.ПеременныеСессии.IPАдресСервера, HttpContext.Server.MachineName)  };    return View(viewModel); |

#### Версия 4.1

С версии Системы 4.1 и позже со стороны Системы в файл svody.config добавлена секция, которая позволяет настроить параметры защиты от CSRF-атак, отличные от применяемых по умолчанию в NetCore. При использовании кросс-авторизации заполните эту секцию конфигурационного файла:

|  |
| --- |
| <!-- Настройки безопасности, определяющие параметры защиты от CSRF-атаки на приложение    AllowedDomainsForShowInFrame - указывается значение http-заголовка, который позволяет добавлять в исключения домены, на которых возможно открытие UI платформы в IFRAME.  например ALLOW-FROM http://your.site.  Значение по умолчанию - SAMEORIGIN    SameSiteCookiePolicy - указывается политика, применяемая к сессионной cookie, выдаваемой сервером каждому пользователю, который смог успешно авторизоваться в системе.  Подробнее о значениях - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite, https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.http.samesitemode?view=aspnetcore-3.1  Значение по умолчанию - Lax. Для кроссавторизации необходимо указать значение Unspecified  -->  <CsrfSecurity>  <AllowedDomainsForShowInFrame>SAMEORIGIN</AllowedDomainsForShowInFrame>  <SameSiteCookiePolicy>Lax</SameSiteCookiePolicy>  </CsrfSecurity> |

### Авторизация по заголовкам запроса

Реализована точка расширения платформы для авторизации в Системе по заголовкам запроса на стартовую страницу веб-приложения сводов.

Выполните следующие действия:

1. создайте класс, реализующий интерфейс «IRequestHeadersAuthModel»:

|  |
| --- |
| namespace Svody.RIA.Security  {  using System.Collections.Specialized;  using BARS.Svody.Security.Models;    /// <summary> Модель выполнения аутентификации по заголовкам запроса </summary>  public interface IRequestHeadersAuthModel  {  /// <summary> Возможно аутентифицировать по заголовкам </summary>  /// <param name="headers">Коллекция заголовков</param>  /// <returns>true - если модель может выполнить аутентифицировать</returns>  bool CanAuthenticateByHeaders(NameValueCollection headers);    /// <summary> Выполнить аутентификацию пользователя </summary>  /// <param name="headers">Коллекция заголовков</param>  /// <returns>DTO результата аутентификации</returns>  AuthResultDto AuthenticateUser(NameValueCollection headers);  }  } |

1. зарегистрируйте данный класс в «Windsor-контейнере»:

* «CanAuthenticateByHeaders» – метод возвращает значение «true», если данный класс может выполнить аутентификацию и авторизацию пользователя;
* «AuthenticateUser» – аутентификация пользователя по заголовкам и авторизация в Системе. В результате метод возвращает «DTO AuthResultDto». Если авторизация пользователя не выполнена, то заполняются свойства «AuthorizationFailReason» и «AuthorizationFailDetail», текст из данных свойств будет выведен на стартовой странице (Рисунок 63).

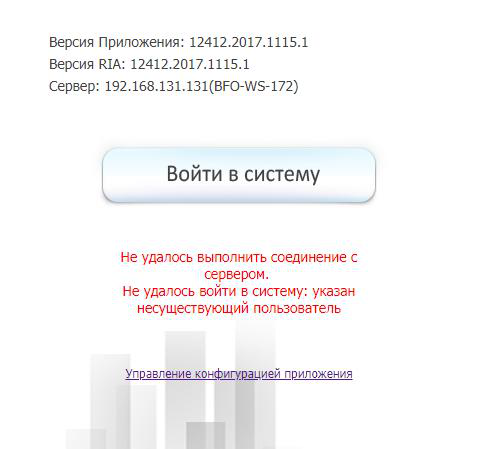


Рисунок 63 – Пример текста из свойств «AuthorizationFailReason» и «AuthorizationFailDetail» на стартовой странице

### Настройка авторизации по протоколу Open ID Connect

Для использования кросс-авторизации по протоколу OpenID Connect (Далее OIDC), то есть входа через пару логин/пароль сторонней системы (Далее SSO), проведите настройку как самой SSO, так и приложения Системы в файле svody.config. После успешной настройки на главной странице авторизации приложения Системы отобразится кнопка, позволяющая выполнить вход с парой логин/пароль другой системы. Предусмотрена функция подключения нескольких способов авторизации, в том числе совмещение авторизации по умолчанию и авторизации по OIDC.

#### Логика авторизации

После успешной авторизации (ввод логина и пароля) на странице авторизации Bars.AM браузер перенаправляет пользователя обратно в Систему и передает код авторизации. Система по коду авторизации запрашивает токены и сохраняет их в Cookie. Время жизни Cookie совпадает с таймаутом сессии Системы. Это значит, что как минимум в течение этого времени пользователь Системы будет считаться авторизованным даже если его сессия была завершена администратором.

Кроме времени жизни Cookie существует другой параметр – время жизни токена авторизации, который был получен из Bars.AM и сохранен в этом Cookie.

**Примечание** – Время жизни cookie и время жизни токена авторизации это два независимых параметра, которые контролируют разные процессы в ходе работы пользователя в системе.

При каждом запросе в Систему происходит проверка времени жизни токена авторизации. Если оно истекло, то происходит повторное обращение в Bars.AM для обновления токенов и их времени жизни. Результатом этого процесса является обновление Cookie в браузере пользователя Системы. Таким образом, пока пользователь работает в Системе (у него активна вкладка браузера с Системой) периодически происходит обновление времени жизни Cookie и времени жизни токенов авторизации.

Данный механизм работы авторизации означает, что если сессия пользователя была завершена на стороне Bars.AM, то выход из Системы произойдет не ранее чем через время жизни токена авторизации (а скорее всего позже – после первого запроса на сервер Системы, выполненного после того как время жизни токена закончилось). Время жизни токена авторизации задается в настройках домена Bars.AM. При настройке этого параметра следует учитывать тот факт, что чем меньше значения, тем больше будет запросов по сети к Bars.AM от всех пользователей на обновление токенов.

#### Подготовка SSO

##### Bars.AM

Выполните следующие действия:

1. разверните и настройте Bars.AM;
2. настройте двустороннюю сетевую связанность между приложением Bars.AM и приложением Системы;
3. создайте пользователей в Bars.AM. У каждого пользователя обязательно укажите Email, т.к. сопоставление пользователей в Системе выполняется по email, указанному в свойствах оператора Системы;
4. зарегистрируйте Систему в Bars.AM (Рисунок 64):

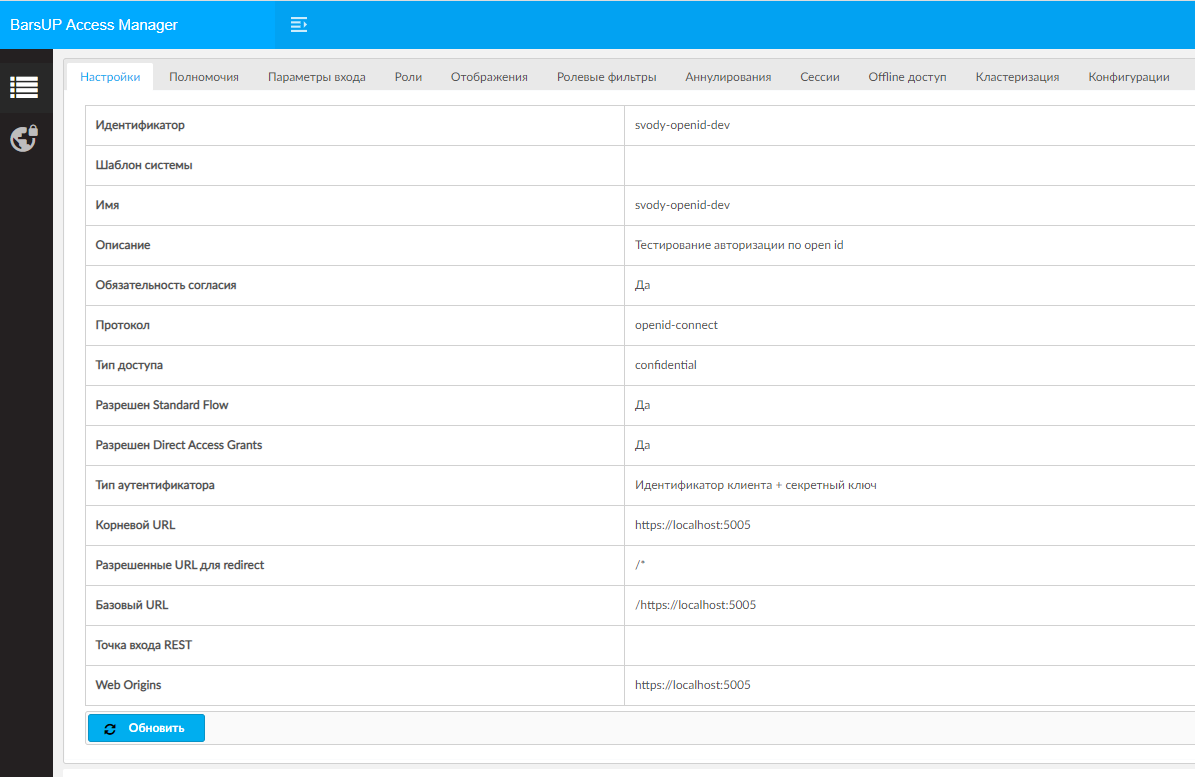


Рисунок 64 – Пример регистрации Системы в Bars.AM

* укажите уникальный идентификатор;
* укажите имя (может совпадать с идентификатором);
* укажите описание (необязательно);
* в поле «Обязательность согласия» выберите значение «ДА» (при первом входе будет предупреждение клиента, какая информация из его профиля в Bars.AM запрашивается Системой);
* выберите протокол – «openid-connect»;
* вы берите тип доступа – «confidential»;
* в поле «Разрешен standard Flow» выберите значение «ДА» (при авторизации выдается код авторизации, который затем обменивается на токены. Таким образом, токены не будут отображаться в URL и не могут быть украдены);
* в поле «Разрешен Direct Access Grants» выберите значение «ДА»;
* выберите тип аутентификатора – «Идентификатор клиента + секретный ключ»;
* в поле «Корневой URL» укажите корневой URL Системы;
* в поле «Разрешенные URL для redirect» выберите значение:
  + «(/\*)» – для тестирования;
  + (/signin-oidc) для production.
* в поле «Базовый URL» введите «/» и Корневой URL;
* в поле «Web Origins» укажите Корневой URL (URL, Разрешенные для CORS);
* настройте отображения. Примеры представлены ниже (Рисунок 65 – Рисунок 76);

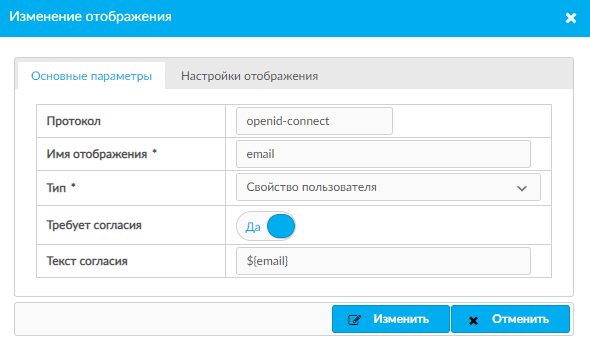


Рисунок 65 – Пример настройки «email». Часть 1

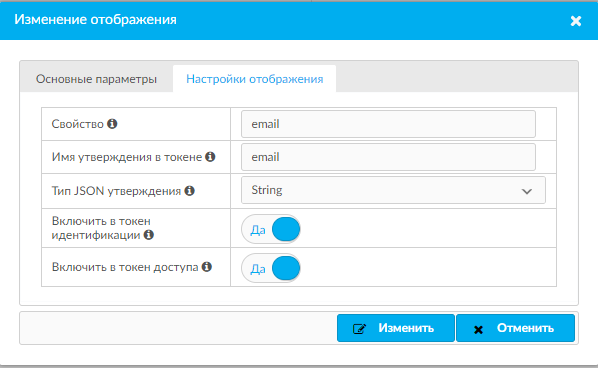


Рисунок 66 – Пример настройки «email». Часть 2

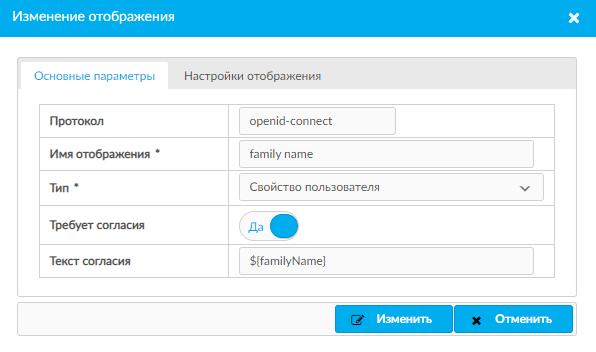


Рисунок 67 – Пример настройки «family name». Часть 1

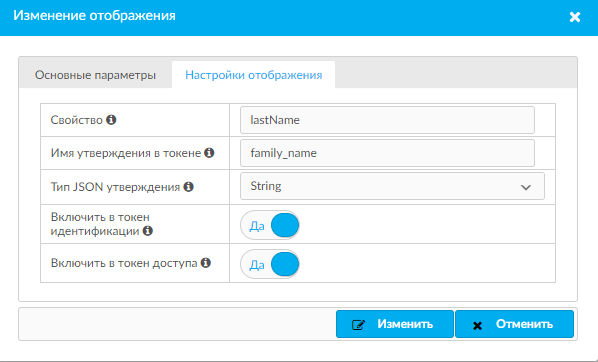


Рисунок 68 – Пример настройки «family name». Часть 2

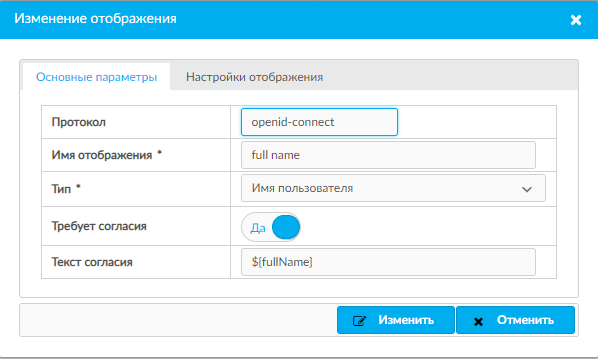


Рисунок 69 – Пример настройки «full name». Часть 1

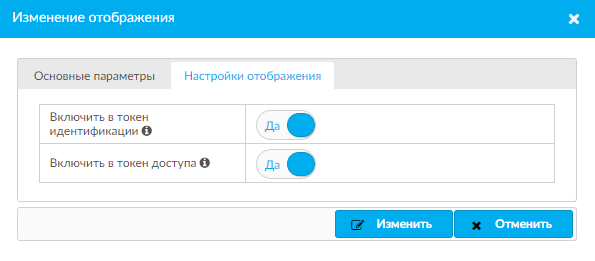


Рисунок 70 – Пример настройки «full name». Часть 2

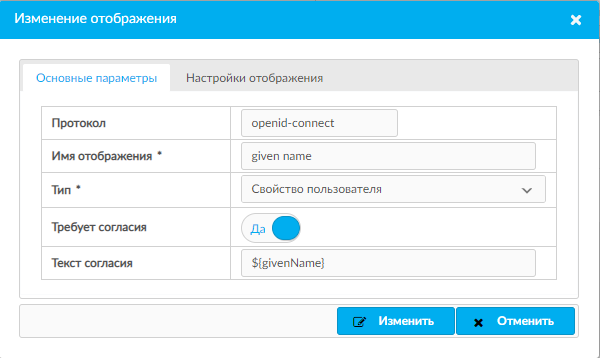


Рисунок 71 – Пример настройки «given name». Часть 1

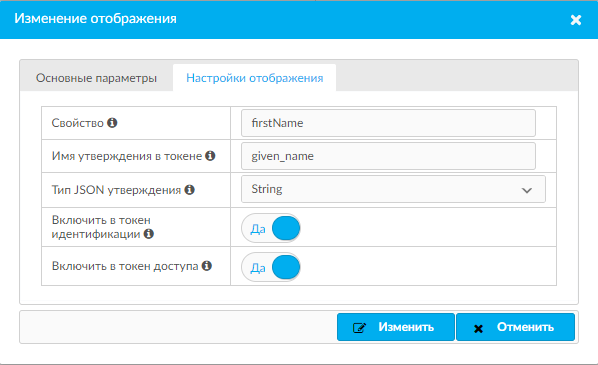


Рисунок 72 – Пример настройки «given name». Часть 2

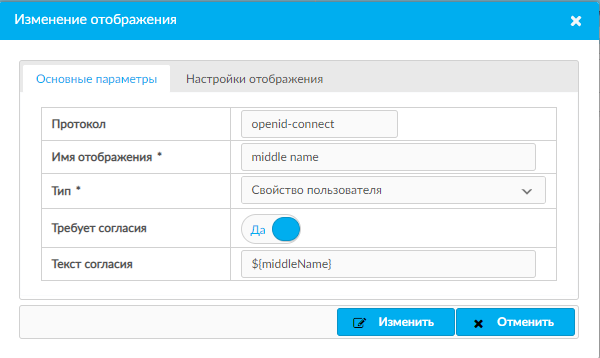


Рисунок 73 – Пример настройки «middle name». Часть 1

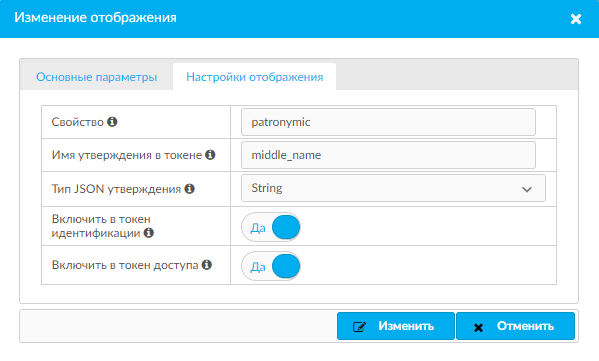


Рисунок 74 – Пример настройки «middle name». Часть 2

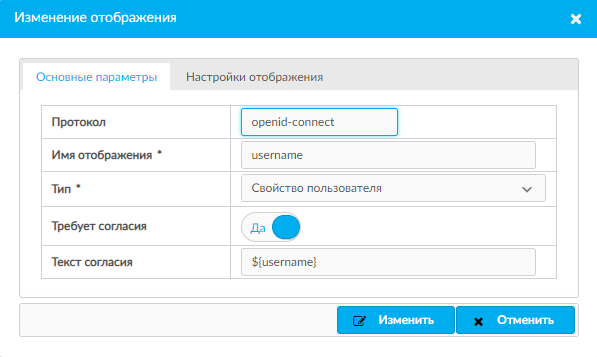


Рисунок 75 – Пример настройки «username». Часть 1

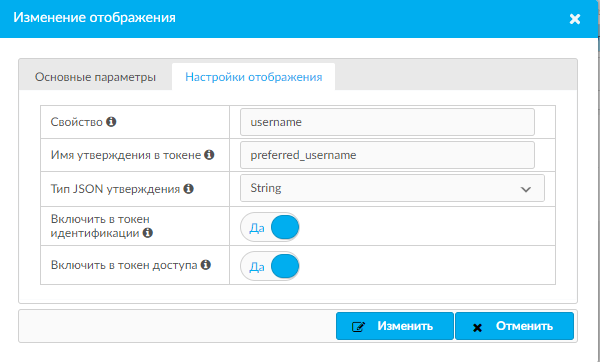


Рисунок 76 – Пример настройки «username». Часть 2

* из вкладки «Полномочия» скопируйте секретный ключ Системы;
* зайдите в настройки домена Bars.AM. Перейдите на вкладку «Ключи». Скопируйте публичный ключ домена.

1. перейдите по адресу http://{ip/домен Bars.AM}/realms/master/.well-known/openid-configuration и скопируйте значения параметров:

issuer  
  
authorization\_endpoint  
  
token\_endpoint  
  
userinfo\_endpoint  
  
end\_session\_endpoint

##### Keycloak

Выполните следующие действия:

1. разверните и настройте Keycloak;
2. настройте двустороннюю сетевую связанность между приложением Keycloak и приложением Системы;
3. создайте домен. Откройте меню в административной консоли в Keycloack. Нажмите на кнопку «Add realm». Откроется форма создания домена. Введите имя и нажмите на кнопку «Создать» (Рисунок 77). После создания автоматически откроются настройки созданного домена. Переключать домен можно с помощью выпадающего меню;

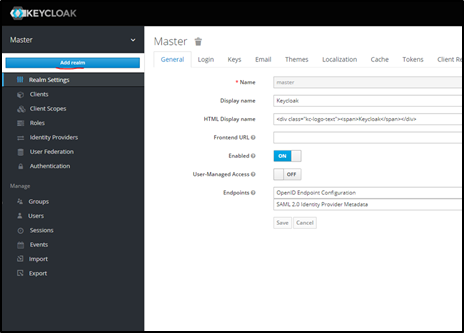


Рисунок 77 – Создание домена

1. создайте пользователей в Keycloak. У каждого пользователя обязательно укажите email, т.к. сопоставление пользователей в Системе выполняется по email, указанному в свойствах оператора Системы;
2. зарегистрируйте Систему в Keycloak (Рисунок 78):

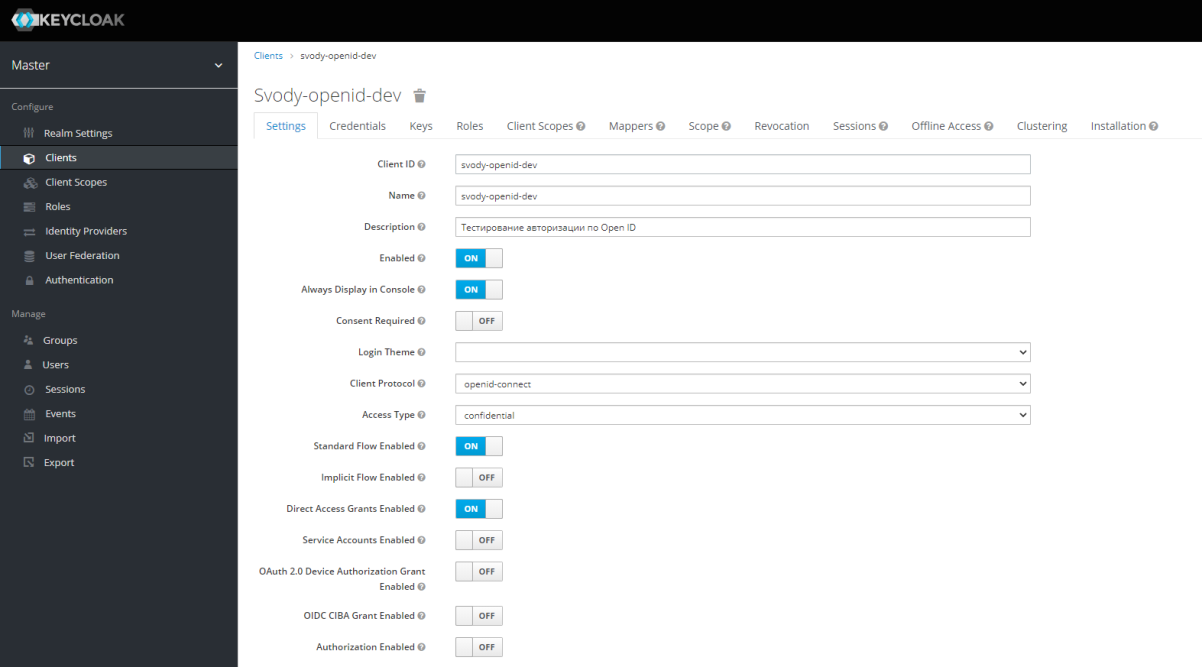


Рисунок 78 – Пример регистрации Системы в Keycloak

* + укажите уникальный идентификатор;
  + укажите имя (может совпадать с идентификатором);
  + укажите описание (не обязательно);
  + в поле «Обязательность согласия» выберите значение «ДА» (при первом входе будет предупреждение клиента, какая информация из его профиля в Bars.AM запрашивается Системой);
  + в поле «Протокол» выберите значение «openid-connect»;
  + в поле «Тип доступа» выберите значение «confidential»;
  + в поле «Разрешен standard Flow» выберите значение «ДА» (при авторизации выдается код авторизации, который затем обменивается на токены. Таким образом токены не будут отображаться в URL и не могут быть украдены);
  + в поле «Разрешен Direct Access Grants» выберите значение «ДА»;
  + в поле «Тип аутентификатора» введите Идентификатор клиента и секретный ключ;
  + в поле «Корневой URL» укажите корневой URL Системы;
  + в поле «Разрешенные URL для redirect» введите;
    - «(/\*)» – для тестирования;
    - «(/signin-oidc)» – для production.
  + в поле «Базовый URL» введите «/» и Корневой URL;
  + в поле «Web Origins» введите Корневой URL (url, Разрешенные для CORS);
  + для не описанных выше полей оставьте значения по умолчанию;
  + настройте отображение на вкладке «Mappers». Нажмите на кнопку «Add Buildin» и по очереди добавьте отображение согласно примеру (Рисунок 79).

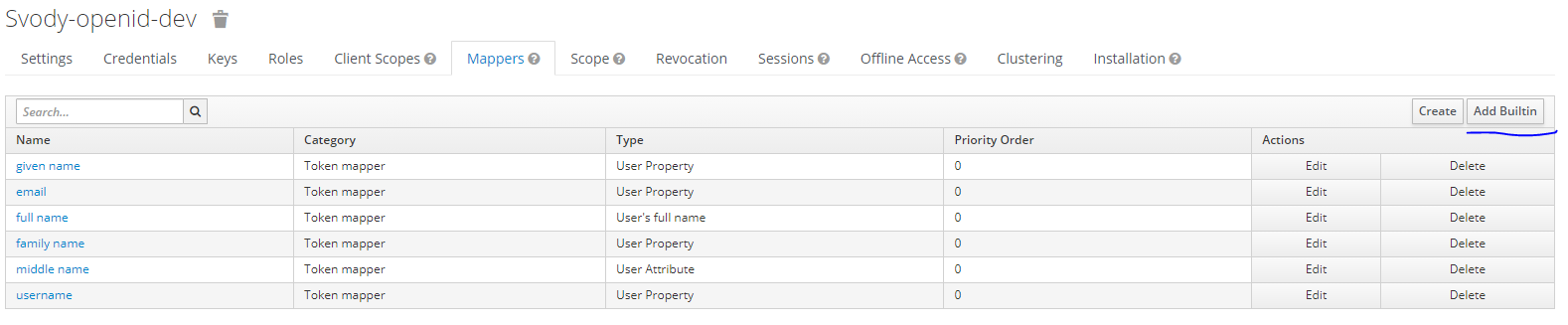


Рисунок 79 – Пример настройки отображения

1. из вкладки «Credentials» скопируйте секретный ключ системы (Secret);
2. перейдите в настройки домена Keycloak (**http://{ip/домен Keykloak}/auth/admin/master/console/#/realms**, выберите нужный домен (по умолчанию master)). Слева выберите значение «Realm Settings». Перейдите на вкладку «Keys». Скопируйте публичный ключ (Algorithm="RS256", Type="RSA", Use="SIG"). Нажмите кнопку «Public key». Скопируйте ключ;
3. перейдите на вкладку «General» раздела «Realm Settings», нажмите на кнопку «OpenId Endpoint Configuration» и скопируйте значения параметров:

issuer  
  
authorization\_endpoint  
  
token\_endpoint  
  
userinfo\_endpoint  
  
end\_session\_endpoint

##### Keycloak (массовый экспорт пользователей)

Предусмотрена функция массового экспорта пользователей с помощью кнопки «добавить» в разделе «Операторы и доступ».

Выполните следующие действия:

1. включите настройку «Service Accounts Enabled» в настройках конкретного клиента в Keycloack и сохраните изменения (Рисунок 80);

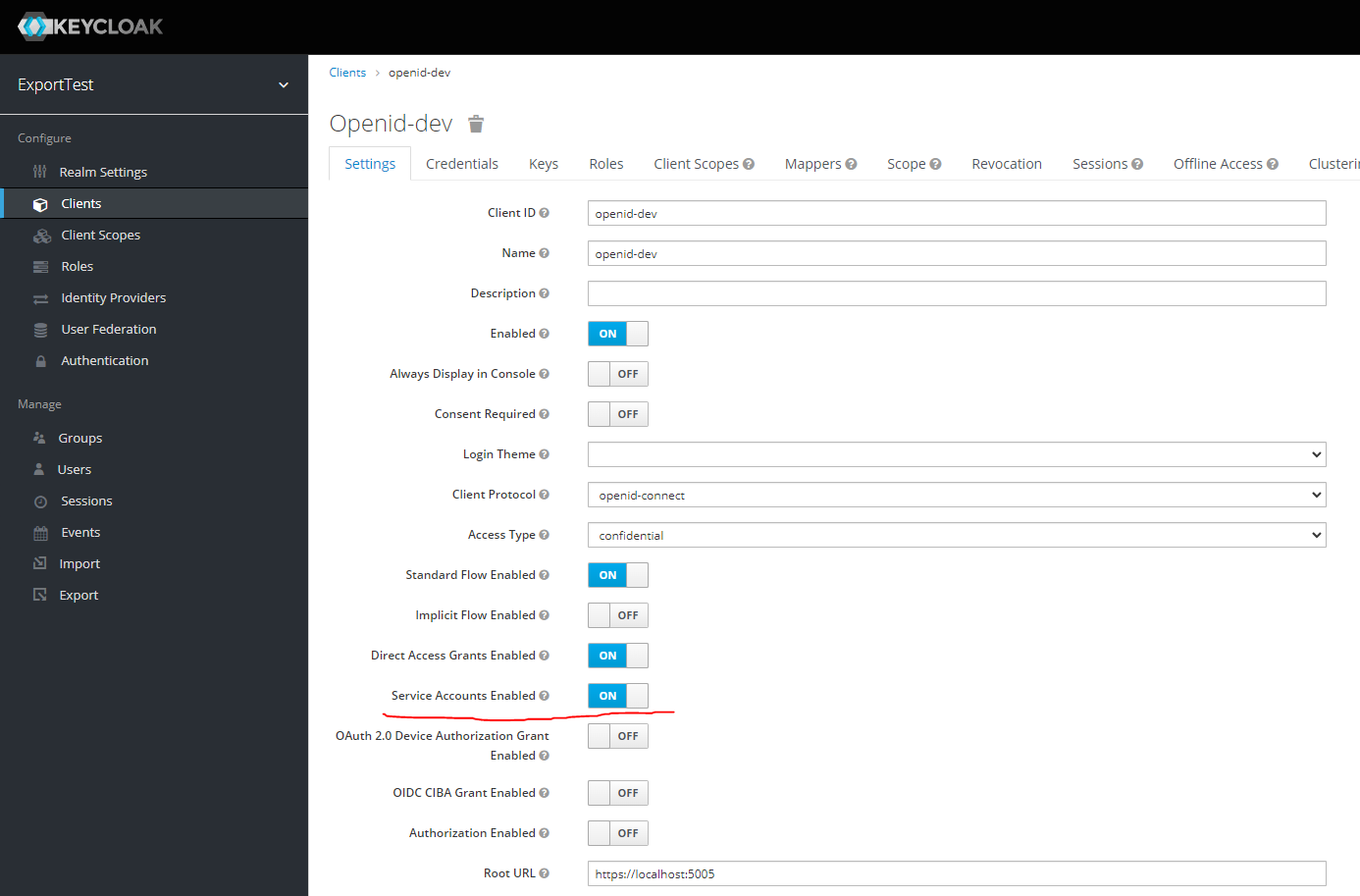


Рисунок 80 – Настройка «Service Accounts Enabled»

1. откройте настройки клиента, отобразится новая вкладка «Service Account Roles» (Рисунок 81);

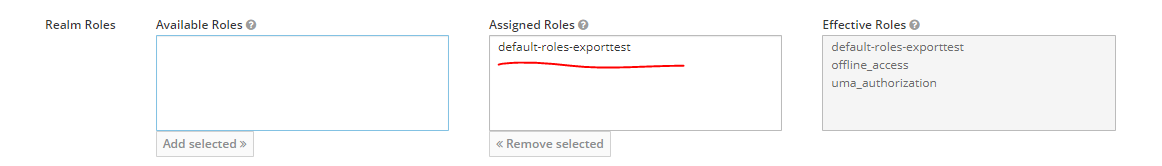


Рисунок 81 – Вкладка «Service Account Roles»

1. перейдите в эту вкладку и назначьте права домена (Realm Roles) и права «Client»;
2. права «Client» зависят от того где был создан клиент, на домене master или на другом домене. Если был создан на другом домене, в «Combobox» выберите «realm-management» и выберите роль «manage-users»;

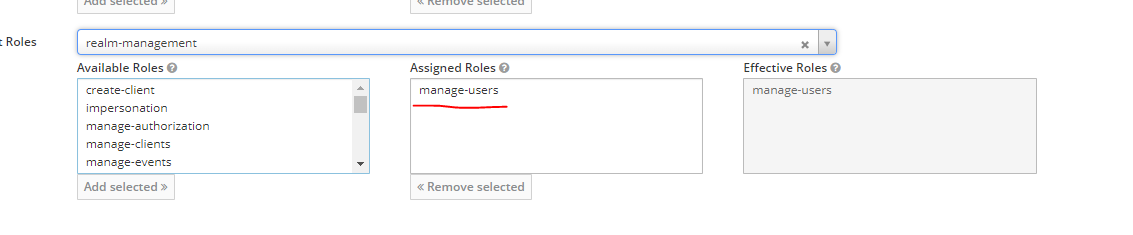


Рисунок 82 – Выбор «realm-management»

1. если клиент был создан на домене master, в Combobox выберите «master-realm» и выберите роль «manage-users» (Рисунок 83);

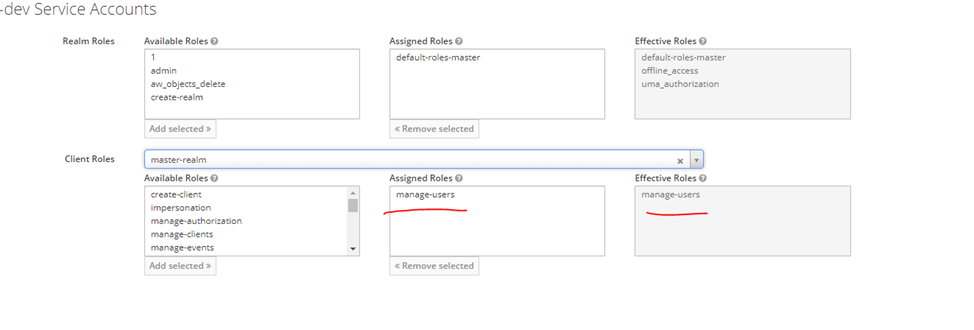


Рисунок 83 – Выбор «master-realm»

1. Измените параметр «RealmsName». Пример:

<RealmsName>master</RealmsName>

#### Конфигурация Системы

**Примечание** – Для авторизации по протоколу SAML в файле svody.config использовалась секция «Bars.Esia». Так как протокол SAML больше не поддерживается, секция «Bars.Esia» была удалена

Настройка авторизации по протоколу Open ID Connect описывается в новой секции «Bars.Authorization» файла «svody.config». Для настройки используйте значения, полученные при настройке SSO.

##### Вид секции

Секция «Bars.Authorization»:

|  |
| --- |
| <Bars.Authorization>  <Entries Name="Svody" Enabled="true" ButtonName="Войти в систему" SortOrder="1" AuthorizationType="Default" />    <Entries Name="Keycloak" Enabled="false" ButtonName="Войти через Keycloak" SortOrder="2" AuthorizationType="OIDC">  <OpenIdConnectProviderConfig>  <ReverseProxyUrl><!-- URL, на который будет перенаправлен пользователь после успешной авторизации на стороне OpenId. Необходимо указывать внешний URL приложения, по которому приходят пользователь (это может быть URL прокси-сервера, на котором установлен https-сертификат) --></ReverseProxyUrl>  <Issuer><!-- значение issuer --></Issuer>  <AuthorizationEndpoint><!-- значение authorization\_endpoint--></AuthorizationEndpoint>  <TokenEndpoint><!-- значение token\_endpoint--></TokenEndpoint>  <UserInfoEndpoint><!-- значение userinfo\_endpoint--></UserInfoEndpoint>  <SignOutEndpoint><!-- значение end\_session\_endpoint--></SignOutEndpoint>  <ClientId><!-- значение Идентификатора системы (вкладка настройки) --></ClientId>  <ClientSecret><!-- значение "Секретный ключ" системы (вкладка полномочия) --></ClientSecret>  <Authority><!-- корневой URL Keycloak -->/</Authority>  <RealmsName><!-- Имя домена SSO провайдера, в котором был создан клиент--></RealmsName>  <ProviderId><!-- ИД SSO провайдера (BARS.AM, KeyCloak), настроенного а AW на подключение к тому же клиенту --></ProviderId>  <ProviderPublicKey><!-- Публичный ключ домена Keycloak --></ProviderPublicKey>  </OpenIdConnectProviderConfig>  </Entries>  </Bars.Authorization> |

##### Параметры

Каждый блок «Entries» отвечает за свой способ авторизации с настройками внутри, первый блок отвечает за авторизацию по умолчанию (через Систему).

Общие параметры:

* «Name» – наименование блока, для авторизации по умолчанию. Изменение значения параметра на значение отличное от «Svody» приведет к неработоспособности Системы;
* «Enabled» – принимает два значения: «true» и «false». При значении «true», на экране авторизации отобразится новая кнопка, при значении «false», блок не учитывается;
* «ButtonName» – текст внутри кнопки;
* «SortOrder» – порядок сортировки кнопок, принимает числовые значения, так при наличии нескольких способов авторизации можно поменять порядок кнопок;
* «AuthorizationType» – принимает значение «Default» для авторизации по умолчанию и «OIDC» для авторизации через SSO.

Параметры SSO:

* «ReverseProxyUrl» – укажите ссылку на приложение;
* «Issuer» – значение «issuer» из настроек «endpoint SSO»;
* «AuthorizationEndpoint» – значение «authorization\_endpoint» из настроек «endpoint SSO»;
* «TokenEndpoint» – значение «token\_endpoint» из настроек «endpoint SSO»;
* «UserInfoEndpoint – значение «userinfo\_endpoint» из настроек «endpoint SSO»;
* «SignOutEndpoint» – значение «end\_session\_endpoint» из настроек «endpoint SSO»;
* ClientId – значение уникального идентификатора системы;
* «ClientSecret» – значение секретного ключа системы;
* «Authority» – ссылка на SSO;
* RealmsName – имя домена из настроек домена, например, master (только для Keycloak, необходимо для возможности массового импорта пользователей из Системы в Keycloak, можно оставить пустым при необходимости);
* «ProviderId» – закомментируйте или удалите, если аналитические выборки не будут использоваться, иначе не отобразится кнопка авторизации;
* «ProviderPublicKey» – публичный ключ домена.

##### Пример настроенной секции

Пример настройки Open ID Connect Provider:

|  |
| --- |
| <Bars.Authorization>  <Entries Name="Svody" Enabled="false" ButtonName="Войти в систему" SortOrder="1" AuthorizationType="Default" />    <Entries Name="Keycloak" Enabled="true" ButtonName="Войти через Keycloak" SortOrder="2" AuthorizationType="OIDC">  <OpenIdConnectProviderConfig>  <ReverseProxyUrl>https://svody.bars.group/svody-openid-dev</ReverseProxyUrl>  <Issuer>http://192.168.233.212:8080/realms/master</Issuer>  <AuthorizationEndpoint>http://192.168.233.212:8080/realms/master/protocol/openid-connect/auth</AuthorizationEndpoint>  <TokenEndpoint>http://192.168.233.212:8080/realms/master/protocol/openid-connect/token</TokenEndpoint>  <UserInfoEndpoint>http://192.168.233.212:8080/realms/master/protocol/openid-connect/userinfo</UserInfoEndpoint>  <SignOutEndpoint>http://192.168.233.212:8080/realms/master/protocol/openid-connect/logout</SignOutEndpoint>  <ClientId>svody-openid-dev</ClientId>  <ClientSecret>a72d6205-4c8b-4c63-ab94-2345f9370eeb</ClientSecret>  <Authority>http://192.168.233.212:8080/</Authority>  <RealmsName>master</RealmsName>  <ProviderPublicKey>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyJdXqJ8lU8qlTP0fuPukQzzZZTi2hD6jUQtKT1gxJgAE7s7Lgw4x2hMwK217Ho99GJPo9Yeo1Yd6S7X8GCcmJzfdumXELUiaVBzfohHDaiAfndzR3vuJRzy72/FNfWOE7J3XFk4/bRtSqRgzma7Rgbt02LUhjhmni3RAeVZfgKToirZR3jMLBMHQq+WCnNtik0KYnFQwxzDokO/gdRI3u6PsLwpEBaXOrddNy3s4XbdGs2R/FfAyeTbHFUcR3L/4u42SBa2P+7tQZoT37CI48dhrQvwRGhvjDcPZfGuCbE6D9rAkgOizRofgZ4Kf55QLR/cyM+MVrErEENj/giOnzwIDAQAB</ProviderPublicKey>  </OpenIdConnectProviderConfig>  </Entries>  </Bars.Authorization> |

После редактирования файла svody.config выполните перезагрузку приложения Системы.

**Примечание –** При настройке секции, можно добавлять несколько блоков «Entries» подряд, в зависимости от того какое количество систем необходимо подключить (например BarsAM и Keycloak одновременно).

##### Особенности настройки приложения СВОДЫ на Linux с Nginx

Внесите изменения в конфигурационный файл nginx приложения по пути **etc/nginx/conf.d**и выполните перезагрузку. Изменения касаются параметров:

buffer.

proxy\_buffer\_size 64k;  
proxy\_buffers 4 64k;  
proxy\_busy\_buffers\_size 64k;

Пример настроенного файла:

|  |
| --- |
| location /sss {  limit\_req zone=one burst=10;  limit\_conn two 10;  client\_max\_body\_size 500M;  proxy\_pass http://127.0.0.1:5011/sss;  proxy\_http\_version 1.1;  proxy\_set\_header Upgrade $http\_Upgrade;  proxy\_set\_header Host $host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $realscheme;  proxy\_set\_header Connection keep-alive;  proxy\_set\_header Connection "upgrade";  proxy\_send\_timeout 600s;  proxy\_read\_timeout 600s;  proxy\_connect\_timeout 600s;  proxy\_buffer\_size 64k;  proxy\_buffers 4 64k;  proxy\_busy\_buffers\_size 64k;  proxy\_temp\_file\_write\_size 1024k;  proxy\_headers\_hash\_max\_size 512;  proxy\_headers\_hash\_bucket\_size 128;    } |

Если настройка не выполнена, то авторизация не будет выполняться.

### Логаут

Выход из приложения самостоятельно пользователем осуществляется нажатием кнопки «Выйти» (Рисунок 84).

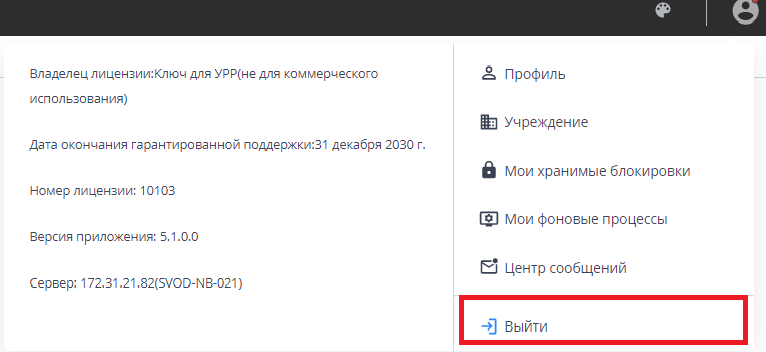


Рисунок 84 – Кнопка «Выйти»

Выход пользователя может быть обработан в событии BeforeLogout, который вызывается в методе «SignOut», реализуемый сервисом «IMembershipProvider».

Удаление сессии пользователя выполняется сервисом, реализующим интерфейс «ISystemConnectionsManager»:

|  |
| --- |
| public interface ISystemConnectionsManager  {  /// <summary>Пометить сессию пользователя удаленной</summary>  /// <param name="sessionIdentifier">Идентификатор сессии</param>  void MarkUserSessionDeleted(string sessionIdentifier);    /// <summary>Помечена ли сессия удаленной</summary>  /// <param name="sessionIdentifier">Идентификатор сессии</param>  /// <returns>true - если сессия помечена удаленной</returns>  bool IsUserSessionMarkedAsDeleted(string sessionIdentifier);    /// <summary>Удалить сессию пользователя</summary>  /// <param name="sessionIdentifier">Идентификатор сессии</param>  void DeleteUserSession(string sessionIdentifier);    /// <summary>Получить тип подключения пользователя по идентификатору сессии.</summary>  /// <param name="sessionIdentifier">Идентификатор сессии.</param>  /// <returns>Тип подключения пользователя: 1-WinКлиент, 2-WebКлиент.</returns>  int? GetClientType(string sessionIdentifier);    /// <summary>Получить идентификатор WEB подключения по идентификатору сессии.</summary>  /// <param name="sessionIdentifier">Идентификатор сессии.</param>  /// <returns>Идентификатор WEB подключения.</returns>  string GetWebSessionId(string sessionIdentifier);    /// <summary>Получить идентификатор сессии веб-приложения по идентификатору агента подключения.</summary>  /// <param name="sessionIdentifier">Идентификатор агента подключения.</param>  /// <returns>Идентификатор сессии веб-приложения</returns>  string GetRiaSessionId(string sessionIdentifier);  } |

Для пользователя при выходе из Системы завершаются все запущенные от его имени процессы и макросы.

#### Очистка неактивных сессий

Система сама каждую минуту очищает сессии, неактивные в течение некоторого времени. Максимальное время жизни неактивной сессии задается в svody.config. «TimeoutInMinutes» задает время в минутах, по прошествии которого неактивная сессия будет очищена Системой (Рисунок 85).

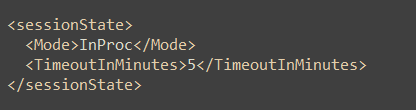


Рисунок 85 – «TimeoutInMinutes»

## Переопределение стартовой страницы Системы

В версии 4.1 и в более поздних версиях Системы реализована возможность переопределения стартовой страницы (см. п. 28.10.3).

### Описание способов переопределения стартовой страницы

Существует два способа переопределить стартовую страницу:

* первый способ частично описан в п. 28.10.3 и включает переопределения только razor-представления. Для переопределения стилей внесите необходимые правки в js и css-файлы:
  + начиная с версии 4.1 и до версии 5.0 Системы:
    - для DEBUG: static/themes/
    - для RELEASE: wwwroot/static/themes/
  + начиная с версии 5.0 Системы:
    - для DEBUG: BARS.Svody.Web.ExtJsClient/
    - для RELEASE: wwwroot/

**Примечание** – В этом случае не нужно вносить изменения в svody.config, но так же будет невозможно быстро переключаться между переопределенной и первоначальной стартовой страницей.

* второй способ заключается в добавлении шаблона стартовой страницы. Для этого выполните следующие действия:
  1. в AddInLib добавьте следующую структуру папок: **Views/Templates/{templateName}/Index.cshtml**;
  2. добавьте стили и js-файлы:
     + начиная с версии 4.1 и до версии 5.0 Системы:
       - для DEBUG создайте папку с названием темы в **static/themes**. Структура папок: **static/themes/{customSourcesFolderName}**;
       - для RELEASE создайте папку с названием темы в **wwwroot/static/themes**. Структура папок: **wwwroot/static/themes/{customSourcesFolderName}**.
     + с версии 5.0 Системы для доступа к первоначальным файлам ресурсов создайте отдельную тему:
       - для DEBUG создайте папку с названием темы в папке проекта «BARS.Svody.Web.ExtJsClient». Структура папок: **BARS.Svody.Web.ExtJsClient/{customSourcesFolderName}**;
       - для RELEASE создайте папку с названием темы в **wwwroot** Структура папок: wwwroot/{customSourcesFolderName}.
  3. измените наименование текущей темы в файле «svody.config»:
  + начиная с версии 4.1 и до версии 5.0 Системы:
    - значение поля «ThemeName» измените на наименование папки ресурсов {customSourcesFolderName}, созданной на шаге 2;
    - значение поля «IndexTemplateName» измените на наименование папки шаблона представления {templateName} созданного на шаге 1.
  + с версии 5.0 Системы значение поля «IndexTemplateName» измените на наименование папки шаблона представления {templateName} созданного на шаге 1.

### Пример добавления шаблона стартовой страницы

В svody.config пропишите имя шаблона customTemplate (Рисунок 86).

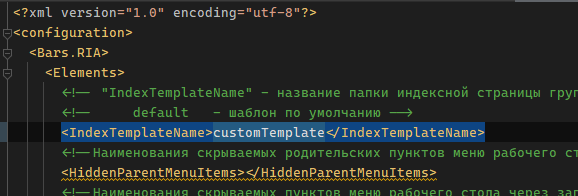


Рисунок 86 – Имя шаблона customTemplate

В AddInLib создайте соответствующую структуру данных и добавьте в нее новый шаблон стартовой страницы (например, с другим текстом и цветом кнопки, которая отображает форму ввода логина и пароля, Рисунок 87)

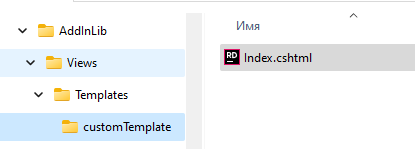


Рисунок 87 – Новый шаблон стартовой страницы

Запустите приложение и проверьте отображение стартовой страницы   
(Рисунок 88).

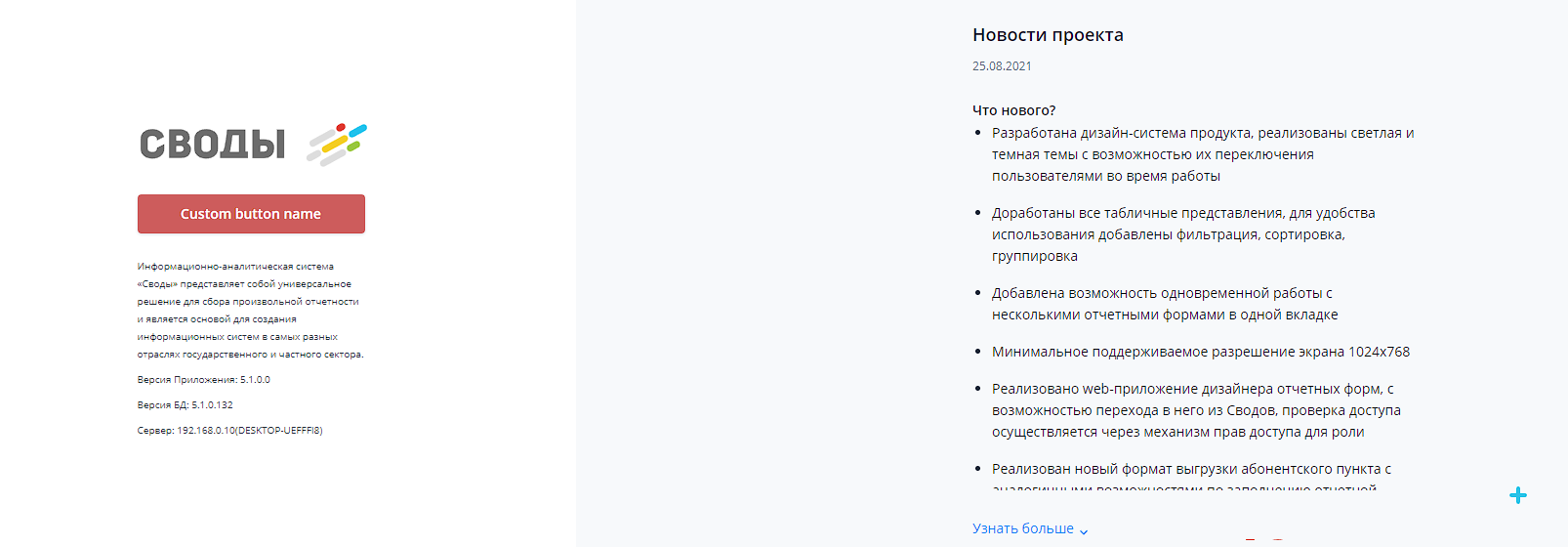


Рисунок 88 – Стартовая страница приложения

Для переключения между шаблонами стартовой достаточно будет изменить значение IndexTemplateName в svody.config. Введите наименование шаблона по-умолчанию «default» (Рисунок 89).

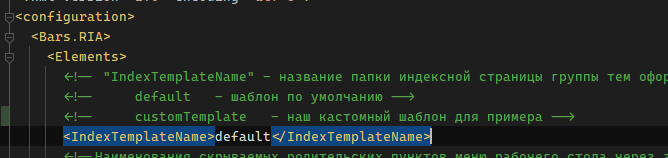


Рисунок 89 – Выбор шаблона по-умолчанию

Перезапустите приложение и проверьте отображение стартовой страницы (Рисунок 90).

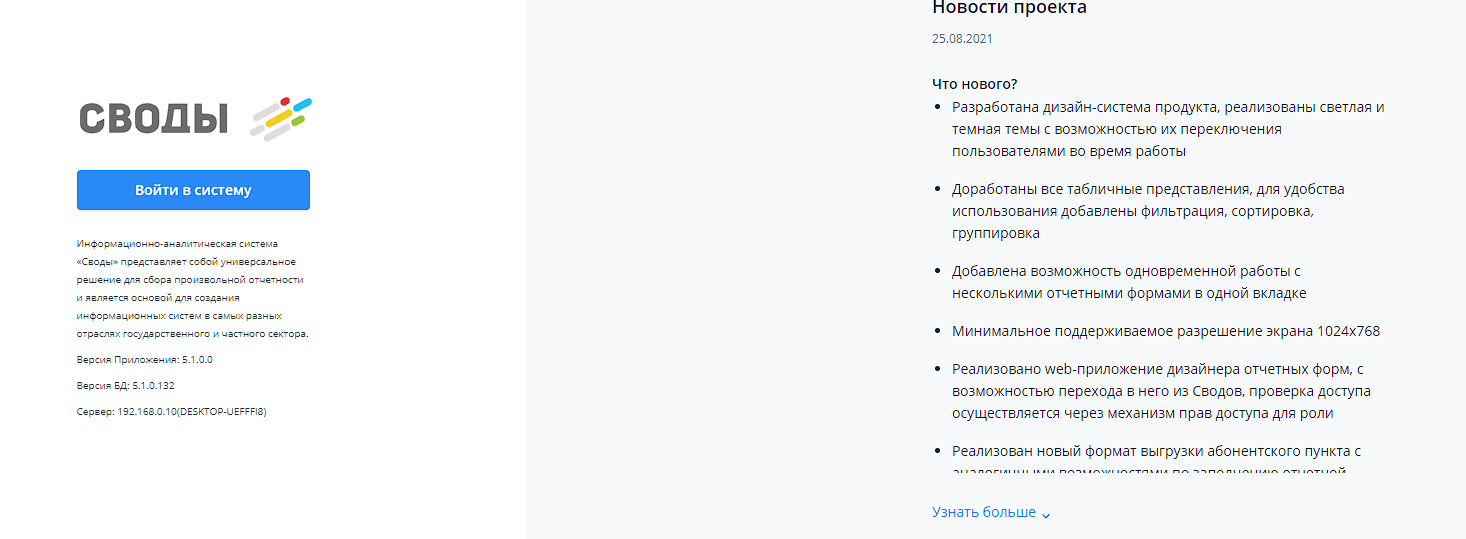


Рисунок 90 – Стартовая страница приложения с шаблоном по-умолчанию

### Переопределение Razor view

Для переопределения view Системы реализована загрузка view-файлов из папки AddInLib.

Ниже приведен пример настройки стартовой страницы:

|  |
| --- |
| return View("~/Views/Home/Index.cshtml", viewModel); |

Для переопределения Razor view, создайте структуру папок по примеру на рисунке (Рисунок 91).

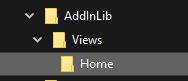


Рисунок 91 – Структура папок

Разместите в папке Home файл Index.cshtml, данная страница будет скомпилирована при первом обращении.

## Кастомизация Front-Office части Системы

Начиная с версии 5.0 в связи с переходом на новый UI и ExtJs 7+ изменился механизм загрузки прикладных js файлов (ранее загружался файл theme.js из папки themes/выбранная\_тема).

Теперь расположение и название точки входа прикладного кода изменилось на apiJs/Main.js

В случае debug-сборки точка входа прикладного кода находится в папке websvody\src\BARS.Svody.Web.ExtJsClient\apiJs.

В release-сборке – путь до приложения на веб сервере\wwwroot\apiJs

**Примечание** – В отличии от кастомизации с помощью тем (см. п. 28.20.1) этот вариант поддерживает динамическую загрузку JS и CSS-файлов. Важно, что SCSS в этом случае не поддерживается.

### Создание и загрузка прикладного JS

Ниже представлено содержание по умолчанию для файла **websvody\src\BARS.Svody.Web.ExtJsClient\apiJs\Main.js**:

|  |
| --- |
| Ext.define('Svody.apiJs.Main', {  enter: async function(options){  }  }) |

За аргумент принимается объект «options», содержащий свойства:

* «isAdmin» – признак, указывающий на то, что пользователь является администратором;
* «isDev» – признак, указывающий на то, что пользователь авторизован с режимом разработчика

Возможно:

* описание дополнительной логики в методе «enter»:

|  |
| --- |
| Ext.define('Svody.apiJs.Main', {  enter: async function(options){  // получаем кнопку "Дополнительно" на тулбаре навигации  const additionals = Ext.getApplication().getMainView().down('#additionals');  // добавляем в меню новый пункт  additionals.getMenu().add({  text: 'Новый пункт меню в разделе Дополнительные функции',  handler: function(){  console.log('Клик');  }  })    if (options.isDev){  Ext.require('Svody.apiJs.jobs.DevUtils', function(){  additionals.getMenu().add({  text: 'Пункт только для разработчика',  handler: function(){  console.log('Я разработчик');  }  })  })  }  }  }) |

* вынесение различной логики в разные файлы:

|  |
| --- |
| Ext.define('Svody.apiJs.Main', {  requires: [  'Svody.apiJs.Custom'  ],    enter: async function(options){  if (options.isDev){  Ext.require('Svody.apiJs.CustomDevWindow', function(){  Ext.create('Svody.apiJs.CustomDevWindow').show();  })  }    if (options.isAdmin){  Ext.require('Svody.apiJs.CustomAdminWindow', function(){  Ext.create('Svody.apiJs.CustomAdminWindow').show();  })  }    Svody.apiJs.Custom.init();  }  }) |

Класс «Svody.apiJs.Custom» размещается в файле «Custom.js» рядом с файлом Main.js:

|  |
| --- |
| Ext.define('Svody.apiJs.Custom', {  singleton: true,  init: function(){  // получаем кнопку "Дополнительно" на тулбаре навигации  const additionals = Ext.getApplication().getMainView().down('#additionals');  // добавляем в меню новый пункт  additionals.getMenu().add({  text: 'Новый пункт меню в разделе Дополнительные функции',  handler: function(){  console.log('Клик');  }  })  }  }); |

Поддерживается структура папок. Например, класс «Svody.apiJs.additional.Custom» будет загружен по пути **apiJs/additional/Custom.js**.

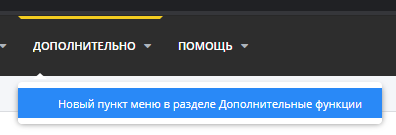


Рисунок 92 – Новый пункт меню в разделе Дополнительные функции

**Примечание** – Если существует папка apiJs, то в ней обязательно должен быть файл Main.js, в котором должен быть класс «Svody.apiJs.Main» с методом «enter».

В директорию «apiJs» добавлены следующие папки:

* «view» – для реализации собственных компонентов, контроллеров, scss стилей;
* «model» – для создания собственных моделей;
* «overrides» – для переопределения ext.js-компонентов, компонентов Системы;
* «store» – для создания собственных хранилищ данных;
* «locale\{app.locale}» – например, для ru путь будет apiJs\locale\ru.

#### Пример настройки кастомизации

Выполните следующие действия:

1. создайте новый компонент с наименованием myWindow в файле apiJs\view\test\Window.js. Для компонента myWindow объявите контроллер testController;

Svody.apiJs.view.test.Window:

|  |
| --- |
| Ext.define('Svody.apiJs.view.test.Window', {  extend: 'Ext.window.Window',  controller: 'testController',  xtype: 'myWindow',  letterSize: 'l',    layout: {  type: 'fit'  },    initComponent () {  Ext.apply(this, {  title: this.locale.titleText,  buttons: [  {  text: this.locale.closeButtonText,  ui: 'btn-primary',  handler: 'close'  }  ]  });  this.callParent()  }  }); |

1. создайте контроллер testController в файле «apiJs\view\test\WindowController.js»;

Svody.apiJs.view.test.WindowController:

|  |
| --- |
| Ext.define('Svody.apiJs.view.test.WindowController', {  extend: 'Ext.app.ViewController',  alias: 'controller.testController',    close(){  const view = this.getView();  view.close();  }    }) |

1. в окне myWindow используется кнопка «Закрыть окно». Текст кнопки загрузите из локали. Создайте локаль в файле: apiJs\locale\ru\view\test\Window.js;

Svody.apiJs.locale.ru.view.test.Window:

|  |
| --- |
| Ext.define('Svody.apiJs.locale.ru.view.test.Window', {  override: 'Svody.apiJs.view.test.Window',    locale: {  closeButtonText: 'Закрыть окно',  titleText: 'Новое окно'  }  }); |

1. измените содержимое Main.js.

Svody.apiJs.Main:

|  |
| --- |
| Ext.define('Svody.apiJs.Main', {  enter: async function(){  // получаем кнопку "Дополнительно" на тулбаре навигации  const additionals = Ext.getApplication().getMainView().down('#additionals');  // добавляем в меню новый пункт  additionals.getMenu().add({  text: 'Новый пункт меню в разделе Дополнительные функции',  handler: function(){  new Svody.apiJs.view.test.Window().show();  }  })  }  }) |

Результат настройки представлен на рисунке ниже (Рисунок 93).

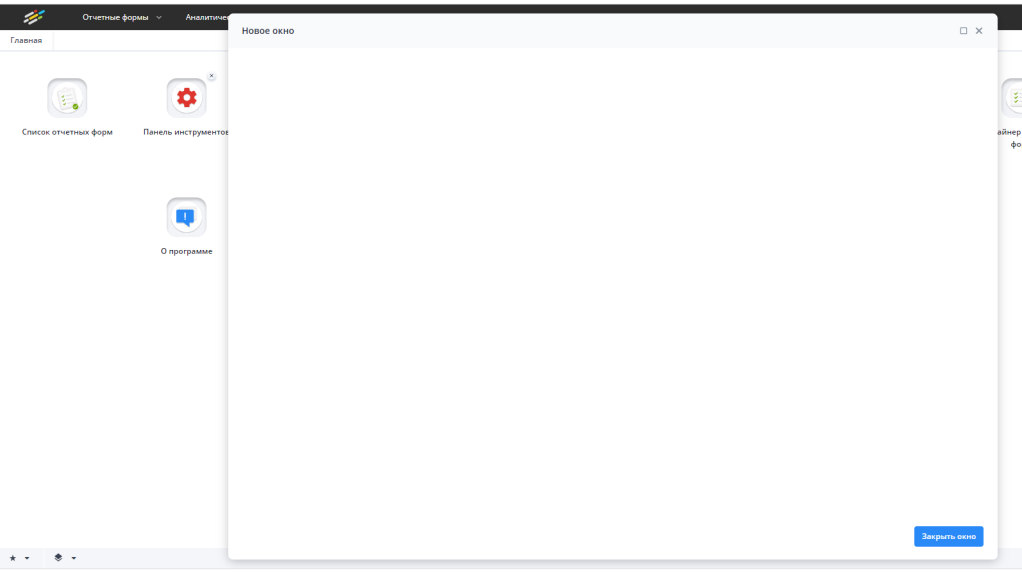


Рисунок 93 – Результат настройки

## Миграции в API

В логику обновления БД из zip-архива API добавлены два новых шага перед загрузкой файлов в БД (в таблицу bars\_source):

* шаг генерации и выполнения SQL с DLL таблиц по типам, которые наследуют тип «ХранимыйОбъект» и определены в сборке API;
* шаг выполнения миграций, реализованных с помощью библиотеки ECM7 в сборке API.

### Возможности Системы

Описание новых возможностей Системы:

1. в Систему добавлена возможность генерации DDL по типам, которые наследуют тип платформы «ХранимыйОбъект». Для генерации SQL используется та же логика, что и для всех других сущностей Системы. Для использования данной возможности достаточно просто определить типы своих хранимых объектов в API и добавить необходимые атрибуты. Миграции будут сгенерированы и выполнены в момент обновления API в БД в соответствующем разделе страницы конфигурации Web  (см. п. 4), либо при обновлении API через менеджер схем;
2. в Систему добавлена возможность выполнения SQL-миграций, реализованных с использованием библиотеки ECM7, в API. Для реализации таких миграций в сборку API добавьте ссылки на сборки, либо ссылку на nuget-пакет с этими сборками (Рисунок 94);

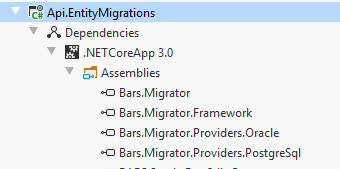


Рисунок 94 – Файловый менеджер

Ниже приведен пример кода. Обязательно добавьте атрибут уровня сборки MigrationAssemly с указанием идентификатора сборки, который будет сохранен в соответствующем столбце таблицы SchemaInfo. Значения атрибута используется библиотекой ECM7.

Пример миграции:

|  |
| --- |
| using ECM7.Migrator.Framework;    [assembly: MigrationAssembly("Bars.ApiMigrations")]    namespace Api.EntityMigrations  {  [Migration(20191202)]  public class Mig:Migration  {  public override void Apply()  {  Database.ExecuteScalar("SELECT SYSDATE FROM DUAL");  }  }  } |

Миграции будут выполнены в момент обновления API в БД в соответствующем разделе страницы конфигурации Web (см. п. 4), либо при обновлении API через менеджер схем.

1. в Систему добавлена возможность использования типов хранимых объектов, определенных в API, при выполнении запросов к БД через интерфейс «IDataStore»;
2. раздел, в котором можно обновить API на схеме из веб-интерфейса, представлен на рисунке (Рисунок 95).

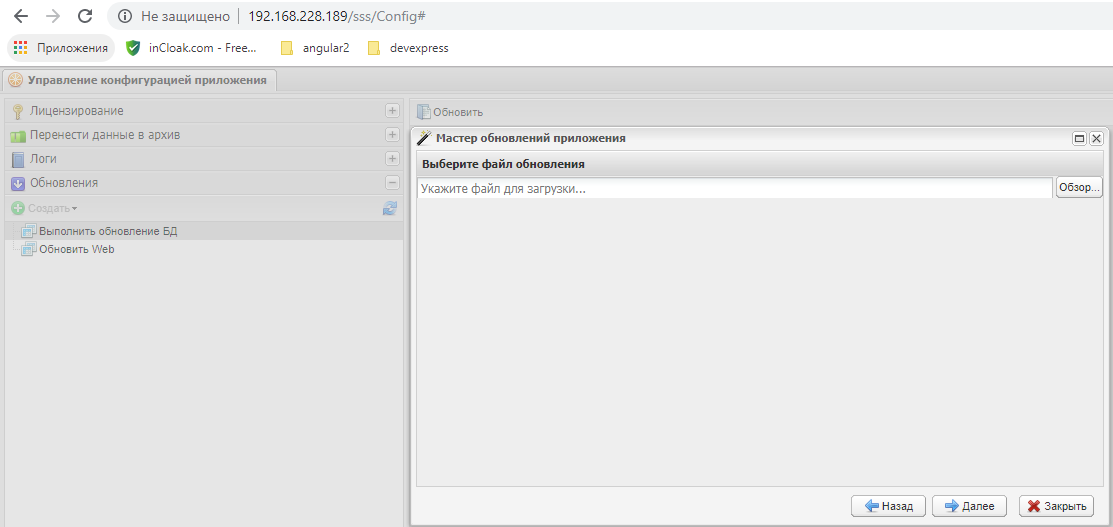


Рисунок 95 – Раздел, в котором можно обновить API

### Рекомендации по реализации API

Рекомендации по реализации API, в которых определены свои типы хранимых объектов и миграции «ECM7»:

* выделение отдельной сборки API, в которой кроме типов хранимых объектов и миграций «ECM7» ничего нет;
* так как загрузка и выполнение таких миграций может выполняться из web-приложения, собранного под netcoreapp3.0, сборки API только под netFramework 4.7.2, либо под netstandard2.0 недостаточно. Поэтому настройте сборку проекта API с сущностями;

Пример .csproj сборки API с миграциями:

|  |
| --- |
| <Project Sdk="Microsoft.NET.Sdk">  <PropertyGroup>  <TargetFrameworks>net472;netcoreapp3.0</TargetFrameworks>  </PropertyGroup>  </Project> |

* файл, собранный под netcoreapp3.0, переименуйте, добавив cуффикс «.migrations» перед расширением файла;
* добавьте переименованный файл и файл, собранный под NetFramework 4.7.2, в общий архив с папкой Addinlib, содержащий остальные файлы API, которые необходимо обновить в БД. Загрузите архив через веб-интерфейс в указанном выше разделе страницы конфигурации или через менеджер схем;
* для удобства автоматизируйте процесс переименования файла с использованием приведенного ниже примера powershell-скрипта;

Пример скрипта, автоматизирующего переименование файла сборки с миграциями и копирование в папку Addinlib:

|  |
| --- |
| $netFrameworkFile = ".\bin\Debug\net472\Api.EntityMigrations.dll"  $netCoreFile = ".\bin\Debug\netcoreapp3.0\Api.EntityMigrations.dll"  $dirPath = "AddinLib"    Write-Output "shwabra"  $exists = Test-Path -Path ".\$dirPath"  if ($exists -eq $True)  {  Remove-Item ".\$dirPath" -Recurse  }    Write-Output "create output folder"  New-Item -Name $dirPath -ItemType "directory"    Write-Output "copy api dlls"  Copy-Item $netFrameworkFile -Destination $dirPath -Force    $migrationsFile= $netCoreFile -replace ".dll", ".migrations.dll"  Copy-Item $netCoreFile -Destination $migrationsFile -Force  Copy-Item $migrationsFile -Destination $dirPath -Force  Remove-Item $migrationsFile |

Скрипт необходимо запускать после выполнения сборки исходников API с помощью dotnet build. Предполагается, что скрипт запускается относительно корневой папки проекта API. В результате выполнения скрипта в папке проекта будет создана папка Addinlib, в которую будет скопировано 2 сборки (Рисунок 96).

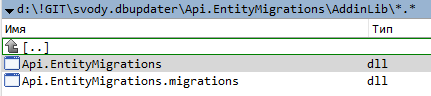


Рисунок 96 – Пример папки Addinlib

* для обновления API веб-приложения (копирование файлов в папку «AddinLib», которая находится в корневой папке веб-приложения) подготовьте отдельный архив с папкой Addinlib, в которой будут только файлы API, собранные под netcoreapp3.0. Раздел обновления представлен на рисунке (Рисунок 97);

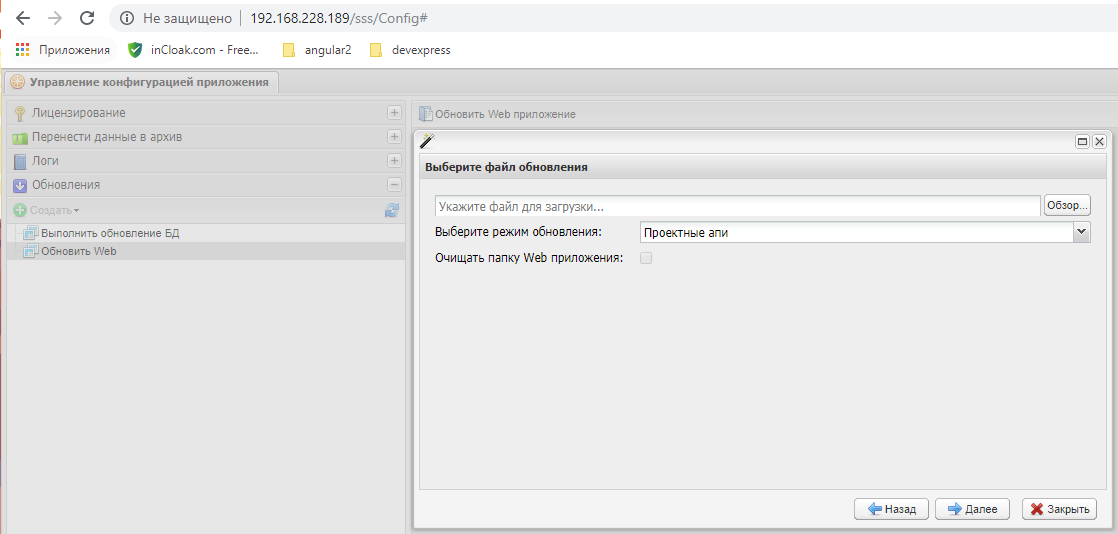


Рисунок 97 – Раздел обновления

**Примечание** – Данную функцию не стоит путать с функцией загрузки файлов API в БД и выполнения миграций, определенных в API, которая доступна в разделе «выполнить обновление БД».

* для обновления API только через менеджер схем, достаточно добавить в zip-архив с папкой Addinlib только файл, собранный под Net Framework 4.7.2.

## Работа через DataStore

Контракт IDataStore:

|  |
| --- |
| /// <summary>Единая точка доступа к IQueryable-провайдеру запросов</summary>  **public** **interface** IDataStore  {  /// <summary>Получить объект запроса</summary>  /// <returns>Iquryable-запрос</returns>  IQueryable<**object**> GetQuery(Type entityTYpe);    /// <summary>Получить объект запроса</summary>  /// <typeparam name="TEntity">Тип хранимого объекта</typeparam>  /// <returns>Iquryable-запрос</returns>  IQueryable<TEntity> GetQuery<TEntity>()  where TEntity : **class**, **new**();    TEntity ExecuteRecursive<TEntity>(Func<IQueryFactory, TEntity> queryFunc)  where TEntity : **class**, **new**();    /// <summary> Получить объект запроса ко всем инстансам БД </summary>  /// <param name="collisionResolver">Стратегия решения коллизий</param>  /// <typeparam name="TEntity">Тип хранимого объекта</typeparam>  /// <returns>IQueryable-запрос</returns>  IQueryable<TEntity> GetQueryFromAllDb<TEntity>(ICollisionResolver collisionResolver = **null**)  where TEntity : **class**, **new**();    /// <summary> Получить объект запроса к конкретному инстансу к БД </summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <typeparam name="TEntity">Тип хранимого объекта</typeparam>  /// <returns>IQueryable-запрос</returns>  IQueryable<TEntity> GetQuery<TEntity>(**string** dbKey)  where TEntity : **class**, **new**();  } |

Чтобы воспользоваться DataStore, добавьте его в необходимый класс через свойство или конструктор, если класс зарегистрирован в контейнере и определяется из него.

Если необходимый класс не зарегистрирован в контейнере или создается, например, через new, то выберите способ получения из контейнера зависимостей.

Пример Injection:

|  |
| --- |
| **public** **class** DataStoreTest  {  /// <summary>  /// Injection через свойство  /// </summary>  **public** IDataStore DataStore { **get**; **set**; }    /// <summary>  /// Injection через конструктор  /// </summary>  **private** IDataStore \_dataStore;    **public** DataStoreTest(IDataStore dataStore)  {  \_dataStore = dataStore;  }    /// <summary>  /// Достаем из контейнера  /// </summary>  **private** IDataStore \_dataStore = КонтейнерЗависимостей.Текущий.Resolve<IDataStore>();  } |

Ниже по порядку приведены примеры использования:

|  |
| --- |
| //Получаем IQueryable<object>  **var** allReportPeriodFromDb = \_dataStore.GetQuery(**typeof**(ОтчетныйПериод)).WhereDynamic(x => x.Property<Guid>("Идентификатор") == reportPeriod.Идентификатор);    //Используем тип dynamic  **foreach** (dynamic repPer **in** allReportPeriodFromDb)  {  //Получаем идентификатор  **var** reportPeriodId = repPer.Идентификатор;  } |

|  |
| --- |
| //Получаем все компоненты из БД  **var** allComponentsFromDb = \_dataStore.GetQuery<КомпонентОтчетногоПериода>(); |

|  |
| --- |
| Получаем полный путь до группы справочника.  **public** **string** GetGroupFullName(Guid groupId)  {  **var** list = DataStore.ExecuteRecursive(  factory =>  {  **var** cte = factory  .GetCte<CteData>(  groupsQuery =>  factory  .GetQuery<DictionaryGroup>()  .Where(g => g.Идентификатор == groupId)  .Select(  g => **new** CteData  {  Id = g.Идентификатор,  ParentId = g.ParentИдентификатор,  Name = " " +g.Наименование  })  .Concat(  from g **in** factory.GetQuery<DictionaryGroup>()  from g1 **in** groupsQuery.InnerJoin(g1 => g.Идентификатор == g1.ParentId)  select **new** CteData  {  Id = g.Идентификатор,  ParentId = g.ParentИдентификатор,  Name = g.Наименование + "." + g1.Name.Trim()  })  );  **var** final = cte.Select(g => g.Name);  **return** final.ToList();  });  **return** list.Last().Trim();  } |

|  |
| --- |
| //Получаем компоненты из всех инстансов БД (основной и архивных)  //В качестве параметра можно передать реализацию ICollisionResolver, если у вас реализован свой кастомный, который реализует логику разрешения коллизий.  **var** allComponentsFromAllDb = \_dataStore.GetQueryFromAllDb<КомпонентОтчетногоПериода>(); |

|  |
| --- |
| /Получаем компоненты из всех инстансов БД (основной и архивных)  **var** allComponentsFromAllDb = \_dataStore.GetQueryFromAllDb<КомпонентОтчетногоПериода>();    //Получаем компонент из определенной БД  **var** dbKey = allComponentsFromAllDb.FirstOrDefault().DbKey;  **var** componentConcreteDb = \_dataStore.GetQuery<КомпонентОтчетногоПериода>(dbKey); |

Дополнительные примеры:

|  |
| --- |
| //Получаем компонент определенного отчетного периода  //Идентификатор взят рандомный, по факту использовать конкретный.  **var** exampleReportPeriodId = Guid.NewGuid();  **var** components = from reportPeriod **in** \_dataStore.GetQuery<ОтчетныйПериод>()  join component **in** \_dataStore.GetQuery<КомпонентОтчетногоПериода>()  on reportPeriod.Идентификатор equals component.ОтчетныйПериодИдентификатор  where reportPeriod.Идентификатор == exampleReportPeriodId  select component; |

Ниже на рисунках приведены примеры работы «FillFields"():

* без FillFields (Рисунок 98);

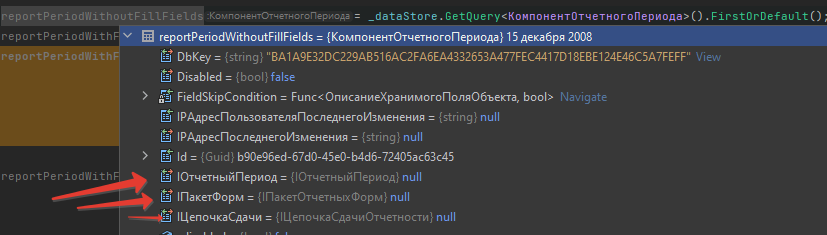


Рисунок 98 – Работа без FillFields()

* с FillFields (Рисунок 99).

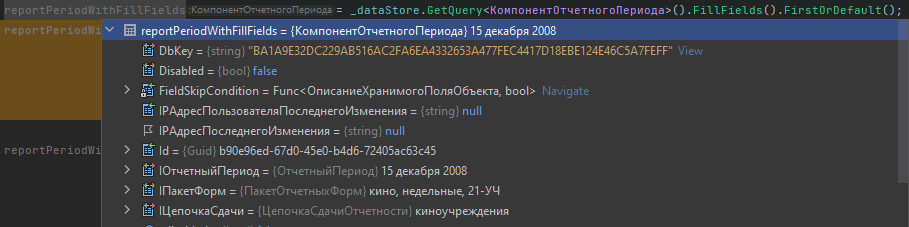


Рисунок 99 – Работа с FillFields

FillFields – метод расширения для выражения IQueryable-запроса. FillFields заполняет поля навигации только на один уровень. При построении проекции в выражении Select необходимо учитывать, что значения свойств, уровень вложенности которых больше чем 1, не будут учитываться. Например, ОтчетныйПериод.Компонент.ЦепочкаСдачи, где ЦепочкаСдачи – свойство 3 уровня. Оно заполнено не будет. Свойство «Компонент» будет заполнено, если используется «FillFields».

**Пример**

С использованием FillFields возвращаются компоненты отчетного периода, которые состоят в какой-либо группе цепочек.

Материализация ChainGroup:

|  |
| --- |
| **var** componentsWithFillFields = (from component **in** \_dataStore.GetQuery<КомпонентОтчетногоПериода>()  join cepochka **in** \_dataStore.GetQuery<ЦепочкаСдачиОтчетности>()  on component.ЦепочкаСдачиИдентификатор equals cepochka.Идентификатор  where cepochka.ChainGroupИдентификатор != Guid.Empty  select component).FillFields().ToList(); |

В данном примере цепочка сдачи у компонента материализовалась, а группа цепочки у цепочки сдачи нет, хотя у группы есть идентификатор по свойству «ChainGroupИдентификатор» (Рисунок 100).

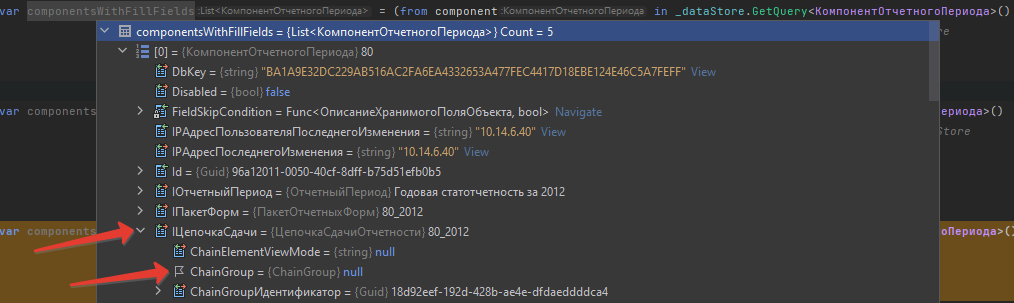


Рисунок 100 – Пример использования FillFields

Материализуйте ChainGroup через Select (Рисунок 101).

Материализация ChainGroup:

|  |
| --- |
| **var** componentsWithoutFillFields = (from component **in** \_dataStore.GetQuery<КомпонентОтчетногоПериода>()  join cepochka **in** \_dataStore.GetQuery<ЦепочкаСдачиОтчетности>()  on component.ЦепочкаСдачиИдентификатор equals cepochka.Идентификатор  where cepochka.ChainGroupИдентификатор != Guid.Empty  select **new** { component, component.ЦепочкаСдачи.ChainGroup }).ToList(); |

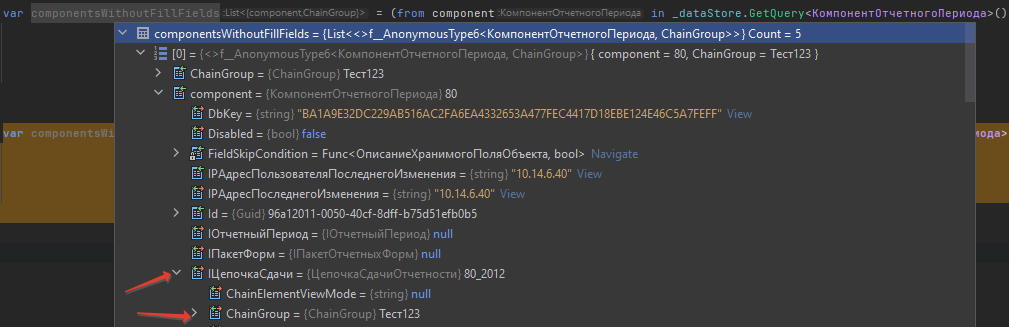


Рисунок 101 – Материализация ChainGroup

В данном случае не материализовались свойства «ОтчетныйПериод» и «ПакетФорм».

Обратите внимание на то, что FillFields используется для запроса компонентов отчетного периода dataStore.GetQuery<КомпонентОтчетногоПериода>().FillFields() и при выполнении join необходимо будет указывать идентификатор следующим образом: «component.ЦепочкаСдачи.Идентификатор», потому что при выполнении FillFields не будет заполнено свойство «component.ЦепочкаСдачиИдентификатор».

Материализация ChainGroup:

|  |
| --- |
| **var** componentsWithFillFields3 = (from component **in** \_dataStore.GetQuery<КомпонентОтчетногоПериода>().FillFields()  join cepochka **in** \_dataStore.GetQuery<ЦепочкаСдачиОтчетности>()  on component.ЦепочкаСдачи.Идентификатор equals cepochka.Идентификатор  where cepochka.ChainGroupИдентификатор != Guid.Empty  select **new** {component, component.ЦепочкаСдачи.ChainGroup}).ToList(); |

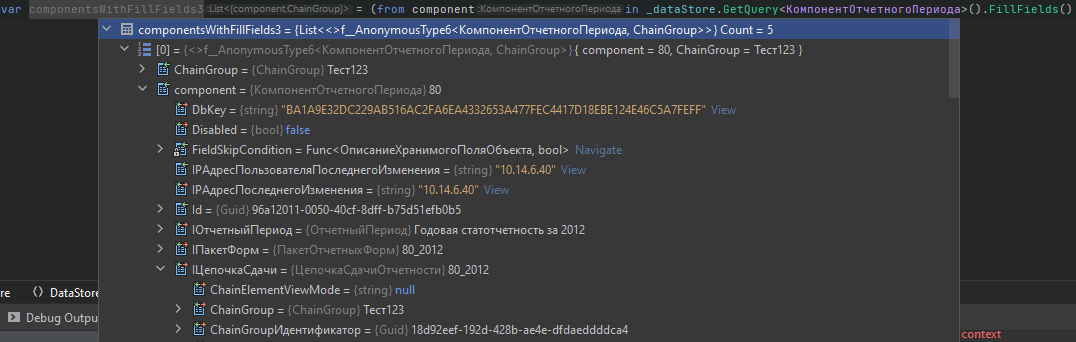


Рисунок 102 – Материализация ChainGroup

Необходимо оценивать необходимость использования FillFields(), потому как операция получения данных становится более затратным по ресурсам. Если необходимы не все поля, то лучше воспользоваться функцией «Select».

## Работа с экспертизой

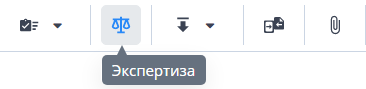


Рисунок 103 – Экспертиза

Ниже представлены основные классы работы с экспертизой:

* (ХО) ЭкспертизаФормы (Таблица 59);

Таблица 59 – Связь между решением эксперта и хранимыми данными формы

|  |  |  |
| --- | --- | --- |
| Свойство | Тип | Описание |
| ДанныеФормы | ХранимыеДанныеФормы | Хранимые данные формы, которые подвергаются экспертизе. |
| Эксперт | ЭкспертФормы | Эксперт для данной экспертизы. |
| Решение | СостояниеЭкспертизы | СостояниеЭкспертизы текущей экспертизы:  НеНачато = 0, Согласовано = 1, НеСогласовано = 2 |
| Сообщения | ПодчиненнаяВыборка<СообщениеЭксперта> | все сообщения эксперта по экспертизе |

* (ХО) СообщениеЭксперта (Таблица 60);

Таблица 60 – Свойства ХО СообщениеЭксперта

|  |  |  |
| --- | --- | --- |
| Свойство | Тип | Описание |
| Сообщение | string | сообщение эксперта |
| Статус | bool | статус сообщения |
| ЭкспертизаФормы | ЭкспертизаФормы | связь с экспертизой формы |

* (ХО) ЭкспертизаДанных (Таблица 61);

Таблица 61 – Связь между Экспертом и хранимыми данными формы

| Свойство | Тип | Описание |
| --- | --- | --- |
| Эксперт | ЭкспертФормы | Эксперт, осуществляющий экспертизу данных |
| Данные | ХранимыеДанныеФормы | Хранимые данные формы, которые подвергаются экспертизе |

* ЭкспертФормы (Таблица 62).

Таблица 62 – Эксперт элемента цепочки

|  |  |  |
| --- | --- | --- |
| Свойство | Тип | Описание |
| Пользователь | Пользователь | пользователь проводимой экспертизы |
| ЭлементЦепочки | ЭлементЦепочки | ЭлементЦепочки формы, для которой проводится экспертиза |
| ГлавныйЭксперт | bool | true - данный эксперт главный, false - в противном случае |
| ПорядокСортировки | int | порядок сортировки форм для экспертизы |
| Недействующий | bool | true - данный эксперт ныне недействующий, false - в противном случае |
| УровеньЭкспертизы | УровеньЭкспертизы | текущий уровень экспертизы формы |

### Информация об экспертизе в хранимых данных

Методы и свойства в классе «ХранимыеДанныеФормы» для работы с экспертизой представлены ниже (Таблица 63):

|  |
| --- |
| **var** хранимыеДанныеФормы = ХранилищеДанныхФорм.ПолучитьДанныеФормы(Форма.Идентификатор);  **var** экспертиза = хранимыеДанныеФормы.ВыборкаЭкспертизТекущейФормы; |

Таблица 63 – Методы и свойства в классе «ХранимыеДанныеФормы» для работы с экспертизой

| Название | Тип | Описание |
| --- | --- | --- |
| Экспертиза | ПодчиненнаяВыборка<ЭкспертизаДанных> | подчиненная выборка экспертиз текущих данных форм |
| ТекущийУровеньЭкспертизы | УровеньЭкспертизы | номер текущего уровня экспертизы |
| ТекущийУровеньЭкспертизыИдентификатор | Guid | Идентификатор уровня экспертизы |
| СостояниеНовойЭкспертизы | string | строка, характеризующая состояние новой экспертизы |
| NewExpertiseStatus | string |  |
| СостояниеЭкспертизы | СостояниеЭкспертизыДанных | состояние экспертизы данных |
| УровниЭкспертизы | List<УровеньЭкспертизы> | коллекция уровней экспертизы для данной формы. |
| ФормаНаходитсяНаПоследнемУровнеЭкспертизы | bool | True, если на форме многоуровневая экспертиза и она на последнем уровне |
| НаФормеИспользуетсяМногоуровневаяЭкспертиза | bool | True, если в форме используется многоуровневая экспертиза |
| ПользовательЯвляетсяЭкспертомТекущегоУровня | bool | Принадлежность авторизовавшегося пользователя текущему уровню экспертизы |
| ВыборкаЭкспертизТекущейФормы | Выборка<ЭкспертизаФормы> | Выборка экспертиз текущей формы |
| ПереходИзСтатусаЭкспертизыВозможен() | bool | Метод проверяет, у всех ли экспертов данной формы началась проверка экспертизы. Если хотя бы у одного эксперта формы состояние проверки СостояниеЭкспертизы =  НеНачато, то метод вернет «false». Состояние экспертизы у выключенных экспертов не учитывается. |
| IsStatusTransitionAvailableOnCurrentExpertiseLevel(IReportFormStatus endStatus) | bool | Проверка возможности перехода для указанного конечного состояния |
| СброситьВыборкуЭкспертизТекущейФормы() | void | Сбрасывает выборку экспертиз текущей формы. Вызывается, когда текущая выборка становится неактуальной |

### Информация об экспертах в элементе цепочки

Cвойства в классе «ЭлементЦепочки» для работы с экспертизой представлены в ниже (Таблица 64):

|  |
| --- |
| **var** element = \_dataStore.GetQuery<ЭлементЦепочки>()  .FillFields()  .Where(elem => elem.ИдентификаторОбъекта == compоnent.IЦепочкаСдачи.ИдентификаторОбъекта)  .FirstOrDefault();    **var** expert = element.Эксперты; |

Таблица 64 – Свойства в классе «ЭлементЦепочки» для работы с экспертизой:

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| НаследуемыеУровни | IEnumerable<УровеньЭкспертизы> | Наследуемые уровни экспертизы, получаемые от родительского элемента цепочки. |
| УровниПередаваемыеДочернимЭлементам | IEnumerable<УровеньЭкспертизы> | Уровни экспертизы, передаваемые дочерним элементам. Данные уровни наследуемы для выбранных дочерних элементов данного элемента |
| НаследуемыеЭксперты | IEnumerable<ЭкспертФормы> | Наследуемые эксперты, получаемые от родительского элемента цепочки. |
| ЭкспертыПередаваемыеДочернимЭлементам | IEnumerable<ЭкспертФормы> | Эксперты, передаваемые дочерним элементам. Данные эксперты наследуемы для выбранных дочерних элементов данного элемента цепочки. |
| ActiveExperts | IEnumerable<ЭкспертФормы> | Действующие эксперты |
| СобственныеЭксперты | IEnumerable<ЭкспертФормы> | Эксперты, созданные на элементе цепочки. Принадлежат собственным уровням |
| ПолучаетУровниЭкспертизыРодителя | bool | Значение, которое показывает принимают ли уровни родителя участие в многоуровневой экспертизе на данном элементе цепочки |
| СобственныеУровни | IEnumerable<УровеньЭкспертизы> | Уровни, созданные на элементе цепочки. С этих уровней начинается экспертиза формы для данного элемента цепочки |
| Эксперты | ПодчиненнаяВыборка<ЭкспертФормы> | Подчиненная выборка ЭкспертФормы, относящихся к данному элементу формы |
| Уровни | ПодчиненнаяВыборка<УровеньЭкспертизы> | Подчиненная выборка УровеньЭкспертизы, относящихся к данному элементу формы |

Получить объекты экспертизы можно через «IDataStore».

Пример взаимодействия и получения данных: Приведенная ниже обработка удаляет выбранных экспертов из цепочек выбранного компонента:

|  |
| --- |
| // Про добавление пользовательских параметров https://conf.bars.group/pages/viewpage.action?pageId=62550090    public void УдалениеЭкспертовИзЦепочки()  {  var comp = (СписокПараметровОперации.ПолучитьЗначениеПараметра("Компонент") as List<ReportPeriodComponentValueDto>).FirstOrDefault(); // получаем выбранные компонент  var expertsDto = СписокПараметровОперации.ПолучитьЗначениеПараметра("Эксперты") as List<ExpertDto>; //получаем выбранных экспертов    var expertIds = expertsDto.Select(exp => exp.Id);  var compId = comp.Id;  var storedData = \_dataStore.GetQuery<ХранимыеДанныеФормы>()  .Where(data => data.КомпонентПериодаИдентификатор == compId)  .Join(  \_dataStore.GetQuery<ЭкспертизаДанных>().WhereContains(exp => exp.ЭкспертИдентификатор, expertIds),  data => data.Идентификатор,  exp => exp.ДанныеИдентификатор,  (data, exp) => data)  .FillFields()  .ToList();    var paket = new ПакетХранимыхОбъектов();  foreach (var data in storedData)  {  foreach (var экспертиза in data.ВыборкаЭкспертизТекущейФормы)  {  if (экспертиза.Решение.Equals(СостояниеЭкспертизы.НеСогласовано) || экспертиза.Решение.Equals(СостояниеЭкспертизы.Согласовано))  {  экспертиза.Решение = СостояниеЭкспертизы.НеНачато;  paket.Добавить(экспертиза);  }  }  }    paket.СохранитьОбъектыAsync();    var experts = \_dataStore.GetQuery<ЭкспертФормы>()  .WhereContains(x => x.Идентификатор, expertIds)  .FillFields()  .ToList();    var formExpertises = \_dataStore.GetQuery<ЭкспертизаФормы>()  .Where(exp => expertIds.Contains(exp.ЭкспертИдентификатор))  .FillFields()  .ToList()  .GroupBy(expertise => expertise.Эксперт.Идентификатор)  .ToDictionary(group => group.Key, group => group.ToList());    var dataExpertises = \_dataStore.GetQuery<ЭкспертизаДанных>()  .WhereContains(exp => exp.ЭкспертИдентификатор, expertIds)  .FillFields()  .ToList()  .GroupBy(expertise => expertise.Эксперт.Идентификатор)  .ToDictionary(group => group.Key, group => group.ToList());    var elements = \_dataStore.GetQuery<ЭлементЦепочки>()  .FillFields()  .Where(elem => elem.ЦепочкаСдачи.Код == comp.ChainCode)  .ToList();    paket = new ПакетХранимыхОбъектов();  foreach (var expert in experts)  {  foreach (var элемент in elements)  {  if (элемент.Эксперты.Any(x => x.Идентификатор == expert.Идентификатор))  {  var paketExp = new ПакетХранимыхОбъектов();  if (formExpertises.TryGetValue(expert.Идентификатор, out var formExp))  {  paketExp.Добавить(formExp);  }    if (dataExpertises.TryGetValue(expert.Идентификатор, out var dataExp))  {  paketExp.Добавить(dataExp);  }    paketExp.УдалитьОбъекты();    элемент.Эксперты.УдалитьЭлемент(expert);  paket.Добавить(expert);  }  }  }    paket.УдалитьОбъекты();  } |

## Виды справочников и возможность работы с ними

### УниверсальныйСправочник

Для работы со справочником используется интерфейс «IУниверсальныйСправочник» (Таблица 65).

**Примечание** – Для форм, которые используются в версии 4.1 и выше, рекомендуется использовать новые справочники. Методы интерфейса «IУниверсальныйСправочник» работают, но они более затратны по времени и производительности.

Таблица 65 – Интерфейс «IУниверсальныйСправочник»

| **Название метода/свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| ГруппаСправочника | string | Возвращает строковое представление группы, которой принадлежит текущий справочник |
| КодСправочника | string | Возвращает и задает обязательный реквизит код справочника |
| ПолноеНаименование | string | Возвращает полное наименование справочника. Полное наименование передается по следующей схеме: ГруппаСправочника.КодСправочника |
| ДатаНачалаДействия | DateTime | Возвращает и задает дату начала действия текущего универсального справочника |
| ДатаОкончанияДействия | DateTime | Возвращает и задает дату окончания действия текущего универсального справочника |
| ДатаАктуальности | DateTime | Возвращает и задает дату актуальности справочника |
| ValidationDate | DateTime | Возвращает и задает дату валидации справочника |
| ОписаниеСправочника | string | Возвращает и задает краткое описание справочника |
| МаскаПредставления | string | Возвращает маску представления, которая применяется при отображении записей справочника |
| СкрываемыеСтолбцы | string | Возвращает столбцы, которые не показываются при отображении записей справочника |
| РежимВыбораЭлементов | РежимыВыбораЭлементовСправочника | Возвращает и задает режим выбора элементов справочника. Доступны следующие режимы: ОпределяетсяСправочником, Все, Групповые, НеГрупповые, Запрещено |
| ТипВыбораЭлементов | ТипВыбораЭлементовСправочника | Возвращает и задает тип выбора элементов справочника. Доступны следующие типы: ОпределяетсяСправочником, Форма, Список |
| Записи | List<IЗаписьСправочника> | Возвращает список всех записей справочника |
| АтрибутыСправочника | List<IАтрибутСправочника> | Возвращает список всех атрибутов |
| ИдентификаторОбъектаСтрокой | string | Возвращает строковое представление глобального универсального идентификатора хранимого объекта текущего справочника |
| IsCached | bool | Флаг с указанием необходимости кэширования справочника |
| РежимУникальности | РежимУникальностиСправочника | Возвращает и задает режим уникальности справочника. Доступны следующие типы: Код, КодНаименование |
| ЯвляетсяРегиональным | bool | Возвращает признак регионального справочника |
| РежимРедактирования | bool | Возвращает режим редактирования справочника из отчетной формы данных |
| IУчреждение | IУчреждение | Возвращает и задает учреждение текущего универсального справочника (для создания региональных версий справочников) |
| GetAllFilteredRecords() | List<IЗаписьСправочника> | Получить список всех записей согласно назначенному справочнику фильтра |
| ДобавитьЗапись(IЗаписьСправочника записьСправочника) | IЗаписьСправочника | Осуществляет добавление переданного объекта записи справочника к списку записей текущего справочника и возвращает добавленную запись |
| ПолучитьЗаписьПоID(Guid идентификаторЗаписи) | IЗаписьСправочника | Метод получения записи справочника по идентификатору записи справочника |
| ПолучитьЗаписьПоКоду(string кодЗаписи, bool учитыватьПровайдерФильтра) | IЗаписьСправочника | Метод получения записи справочника по коду записи справочника с учетом фильтра справочника |
| ПолучитьЗаписьПоКоду(string кодЗаписи) | IЗаписьСправочника | Метод получения записи справочника по коду записи справочника |
| ПолучитьКорневыеЗаписи() | List<IЗаписьСправочника> | Метод получения корневых записей (без родителей) |
| ПолучитьКорневыеЗаписиНаДату(DateTime actualDate) | List<IЗаписьСправочника> | Получить только корневые записи с учетом указанной даты актуальности |
| ПолучитьНаименованиеЗаписиПоКоду(string кодЗаписи) | string | Метод получения наименования записи справочника по коду записи справочника |
| ПолучитьСписокЗаписейСПовторяющимисяКодами() | List<IЗаписьСправочника> | Получает список записей справочника с дублирующими кодами |
| Синхронизировать(IУниверсальныйСправочник справочник) | void | Синхронизировать свойства текущего справочника со свойствами переданного справочника |
| СодержитКод(string кодЗаписиСправочника) | bool | Определяет, содержит текущий справочник запись с переданным кодом записи |
| СписокСкрытыхПредков(bool обновитьСписок) | Dictionary<Guid, IЗаписьСправочника> | Получает словарь записей справочника, не отображаемых в списке записей справочника из-за фильтрации, по их идентификаторам |

Интерфейс «IЗаписьСправочника» используется для работы с записью справочника (Таблица 66).

Таблица 66 – Интерфейс «IЗаписьСправочника»

| **Название метода/свойства** | **Тип** | **Описание** |
| --- | --- | --- |
| Код | string | Возвращает и задает обязательный реквизит код записи справочника |
| КодПолный | string | Возвращает код записи справочника с учетом кодов родительских записей |
| КодПостобработанный | string | Возвращает код записи справочника с учетом постобработки, заданной в ячейке формы |
| КодПолныйРодительскойЗаписи | string | Возвращает и задает полный код родительской записи справочника |
| КодСборкиСводной | string | Возвращает и задает полный код записи для сборки сводной |
| ПользовательПоследнегоИзмененияОбъекта | string | Возвращает имя пользователя, изменявшего запись справочника последним |
| Представление | string | Возвращает отображаемое значение записи справочника согласно с настройками справочника |
| Классификатор | IУниверсальныйСправочник | Возвращает универсальный справочник, которому принадлежит текущая запись справочника |
| РодительскаяЗапись | IЗаписьСправочника | Возвращает и задает ссылку на родительскую запись справочника |
| IРодительскаяЗапись | IЗаписьСправочника | Возвращает и задает ссылку на родительскую запись справочника |
| ИдентификаторОбъектаСтрокой | string | Возвращает строковое представление глобального универсального идентификатора хранимого объекта текущей записи справочника |
| RegionalParent | IЗаписьСправочника | Ссылка на родительскую запись в БД, которая используется в записях регионального справочника |
| РежимВыбора | РежимВыбораЭлементаСправочника | Возвращает и задает свойство, определяющее режим выбора элемента справочника. Доступны следующие значения: ВыборРазрешен, ВыборЗапрещен, ОпределяетсяСправочником |
| ПодчиненныеЗаписи | List<IЗаписьСправочника> | Возвращает список подчиненных записей для текущей записи справочника |
| ДатаПоследнегоИзмененияОбъекта | DateTime | Возвращает дату последнего изменения хранимого объекта текущей записи справочника |
| ЕстьПодчиненныеЗаписи | bool | Возвращает свойство, показывающее, есть ли у текущей записи справочника подчиненные |
| ВозможностьРедактировать | bool | Возвращает свойство, показывающее есть ли для текущей записи возможность изменять/удалять её из справочника |
| ДатаНачалаДействия | DateTime | Дата начала действия |
| ДатаОкончанияДействия | DateTime | Дата окончания действия |
| Path | string | Возвращает и задает путь в дереве записей до корневого элемента |
| this[string кодАтрибута] | object | Возвращает объект значения атрибута для текущей записи справочника по указанному коду атрибута |
| ВыполнитьПостобработку(ОписаниеСсылкиНаСправочник описание) | void | Выполняет постобработку согласно описанию ссылки на справочник. Присваивает постобработанное значение свойству «КодПостобработанный» |
| ЗначенияВсехАтрибутов() | Dictionary<int, object> | Получает словарь всех объектов значений атрибутов текущей записи справочника по их порядковым номерам |
| ЗначенияВсехАтрибутов(IList<IАтрибутСправочника> атрибуты) | Dictionary<int, object> | Получает словарь объектов значений переданных в параметрах атрибутов текущей записи справочника по порядковым номерам |
| ОпределитьРежимВыбораВКонтексте(IУниверсальныйСправочник справочник, ОписаниеСсылкиНаСправочник описаниеСсылки, bool? естьПодчиненныеЗаписи) | РежимВыбораЭлементаСправочника | Определяет режим выбора записей справочника в зависимости от настроек справочника и от режима, заданного в описании ссылки на справочник |
| ПолучитьПодчиненнуюЗаписьПоКоду(string кодЗаписи) | IЗаписьСправочника | Находит среди подчиненных записей запись с переданным в параметрах кодом |
| ПолучитьПолныйПостобработанныйКод(ОписаниеСсылкиНаСправочник описание) | string | Получает полный постобработанный код с учетом кодов постобработанных родительских записей |
| ПредставлениеПоФормату(ОписаниеСсылкиНаСправочник описание) | string | Возвращает отображаемое значение текущей записи справочника согласно свойству «ФорматОтображения» переданного описания ссылки на справочник |
| Синхронизировать(IЗаписьСправочника исходнаяЗапись) | void | Синхронизировать свойства текущей записи справочника со свойствами переданной записи |
| Удалить() | void | Удаляет хранимый объект записи справочника из базы данных |
| Удалить(bool updateDictionaryCheckMark) | void | Удаляет хранимый объект записи справочника из базы данных с учетом признака необходимости обновления контрольной метки классификатора |
| GetPathIds() | string[] | Получает массив идентификаторов родительских записей в дереве записей (от текущего элемента до корневого) |
| IsVirtual | bool | Признак виртуальной записи справочника |
| DepartmentId | Guid | Идентификатор учреждения записи. Если данное поле не пусто, значит запись является региональной |

Чтобы получить справочник, используется свойство формы «IМенеджерСправочников» (Таблица 67).

Таблица 67 – Форма IМенеджерСправочников

| **Название метода** | **Тип** | **Описание** |
| --- | --- | --- |
| ПолучитьСправочник(string группа, string код, DateTime началоДействия) | IУниверсальныйСправочник | Возвращает справочник из БД |
| ПолучитьСправочник(string группа, string код, DateTime началоДействия, IПрофильПользователя профильПользователя) | IУниверсальныйСправочник | Возвращает справочник из БД с учетом профиля пользователя |
| ПолучитьСправочник(string группа, string код, DateTime началоДействия, DateTime окончаниеДействия, IПрофильПользователя профильПользователя) | IУниверсальныйСправочник | Возвращает справочник из БД с учетом профиля пользователя и даты окончания действия справочника |
| ПолучитьСправочник(string группа, string код, DateTime началоДействия, IПрофильПользователя профильПользователя, bool толькоФедеральный) | IУниверсальныйСправочник | Возвращает федеральный справочник из БД |
| ПолучитьСправочник(string полныйКод) | IУниверсальныйСправочник | Возвращает справочник из БД |
| ПолучитьСправочник(ИдентификаторУниверсальногоСправочника идентификаторУниверсальногоСправочника) | IУниверсальныйСправочник | Возвращает справочник из кэша |
| ПолучитьСправочник(ИдентификаторУниверсальногоСправочника идентификаторУниверсальногоСправочника, bool использоватьПрофильТекущегоПользователя) | IУниверсальныйСправочник | Возвращает справочник из кэша |
| ПолучитьСправочник(Guid идентификаторСправочника) | IУниверсальныйСправочник | Возвращает справочник |
| ПолучитьСправочник(Guid идентификаторСправочника, bool склееныйСправочник) | IУниверсальныйСправочник | Возвращает справочник |
| ПолучитьСправочник(Guid идентификаторСправочника, bool склееныйСправочник, bool использоватьКэш) | IУниверсальныйСправочник | Возвращает справочник |
| ПолучитьСправочник(ИдентификаторУниверсальногоСправочника идентификаторУниверсальногоСправочника, DateTime? датаАктуальности) | IУниверсальныйСправочник | Возвращает справочник с учетом даты актуальности |
| GetLegacyOrNull(ИдентификаторУниверсальногоСправочника идентификатор) | IУниверсальныйСправочник | Возвращает справочник из кэша, если он не поддерживает API работы с записями, реализованное в сборке Nsi.Contracts |
| GetNsiAdapter(ИдентификаторУниверсальногоСправочника идентификатор) | IУниверсальныйСправочник | Получить адаптер справочника, который поддерживает API, реализованное в сборке Nsi.Contracts |
| Exists(ИдентификаторУниверсальногоСправочника dictId) | bool | Проверка наличия справочника в БД |
| GetDictionaryStoredObject(string полныйКод) | IУниверсальныйСправочник | Возвращает хранимый объект справочника, не кэшированный, не склеенный |
| TryFindCachedRecordByCode(string recordCode, Guid? groupId, string dictCode, DateTime? actualDate, out IЗаписьСправочника record) | bool | Поиск записи по полному коду в кэшированных справочниках |
| TryFindInCache(Guid dictId, out IУниверсальныйСправочник cachedDict) | bool | Поиск кэшированного справочника |
| ПолучитьДополнительныеСправочники (ОписаниеСсылкиНаСправочник описание) | List<IУниверсальныйСправочник> | Получает список дополнительных справочников по заданному описанию ссылки на справочник |
| ПолучитьДополнительныеСправочники(string slaveDictCodes) | List<IУниверсальныйСправочник> | Получает список дополнительных справочников по заданному описанию ссылки на справочник |
| GetAdditionalDictionaries(string slaveDictCodes, DateTime actualDate) | List<IУниверсальныйСправочник> | Получает список дополнительных справочников по заданному коду и дате актуальности |
| RemoveCachedDict(Guid dictId) | void | Удалить справочник из кэша |
| GetHasRecordsTableAsync(in Guid dictId) | Task<bool> | Проверить наличие таблицы записей у справочника. Наличие таблицы говорит о том, что справочник был сконвертирован |
| GetGroupId(string groupCode) | Guid | Получить идентификатор группы справочников по полному коду |

Примеры получения справочника представлены ниже:

|  |
| --- |
| public IУниверсальныйСправочник ПолучитьСправочник(ДанныеФормы данныеФормы, string кодСправочника, bool загружатьРегиональные)  {  ИдентификаторУниверсальногоСправочника идентификатор = new ИдентификаторУниверсальногоСправочника();  идентификатор.КодСправочника = кодСправочника;  идентификатор.ГруппаСправочника = "ГруппаСправочника";  идентификатор.ДатаАктуальности = данныеФормы.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала;  if (загружатьРегиональные)  {  идентификатор.Учреждение = данныеФормы.Идентификатор.Учреждение;  }  else  {  идентификатор.СклеенныйСправочник = false;  }    return данныеФормы.МенеджерСправочников.ПолучитьСправочник(идентификатор);  } |

|  |
| --- |
| //Получаем и кэшируем справочник  private IУниверсальныйСправочник \_objectsMappings;  public IУниверсальныйСправочник ObjectsMappings  {  get  {  if (\_objectsMappings == null)  {  \_objectsMappings = МенеджерПриложения.МенеджерСправочников.ПолучитьСправочник(  new ИдентификаторУниверсальногоСправочника  {  ГруппаСправочника = "ГруппаСправочника",  КодСправочника = "КодСправочника",  ДатаАктуальности = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала  });  }    return \_objectsMappings;  }  }    public void FillForm()  {  //Получаем запись справочника  var objectDictionaryRecord = ObjectsMappings.ПолучитьЗаписьПоКоду("КодЗаписи");  if (objectDictionaryRecord == null)  {  return;  }    var newRow = Форма["КодТаблицы"].МатрицаЗначений.Добавить();  newRow["КодСтолбца"].Значение = objectDictionaryRecord;  } |

### «Новые» справочники

С версии Системы 4.1 были обновлены справочники и добавлена новая функциональность по работе с ними.

Пример получения записи справочника по значению атрибута:

|  |
| --- |
| private async System.Threading.Tasks.Task<dynamic> GetRecord(string attributeValue, Guid directoryId)  {  dynamic record = null;  var date = Форма.Идентификатор.КомпонентОтчетногоПериода.IОтчетныйПериод.ДатаНачала;  var syncDictionaryService = КонтейнерЗависимостей.Текущий.Resolve<ISyncDictionaryService>();    //получаем провайдер записей справочника  var recordsProvider = syncDictionaryService.GetRecordsProvider(directoryId, date);  var recordIdsByFieldValue = await recordsProvider.GetRecordIdsByFieldValueAsync("КодАтбрибута", attributeValue);    if (recordIdsByFieldValue.Any())  {  record = recordsProvider.GetDynamic(recordIdsByFieldValue.FirstOrDefault().Id);  }    return (dynamic)record;  } |

Чтобы заполнить ячейку записью справочника, приведите ячейку к типу СсылкаНаСправочник:

|  |
| --- |
| var newRow = tabel.МатрицаЗначений.Добавить();  var value = newRow["КодСтолбца"] as СсылкаНаСправочник;  value.RecordValue = await GetRecord("ЗначениеАтрибутаДляПоиска", directory.ИдентификаторОбъекта); |

Чтобы получить справочник, можно использовать IDictionaryService. Изменить справочник можно через IDictionaryImportService:

|  |
| --- |
| public IDictionaryService DictionaryService = КонтейнерЗависимостей.Текущий.Resolve<IDictionaryService>()  public IDictionaryImportService DictionaryImportService = КонтейнерЗависимостей.Текущий.Resolve<IDictionaryImportService>()  public async Task CreateRecordsAsync(string dictionaryFullCode, List<Employees> employees)  {  //Получаем id справочника  var dictionaryId = await DictionaryService.GetDictIdAsync(dictionaryFullCode, DateTime.Now);  if (!dictionaryId.HasValue)  {  return;  }    //Получаем провайдер записей справочника  var recordsProvider = await DictionaryService.GetRecordsProviderAsync(dictionaryId.Value, DateTime.Now);    //Получаем все атрибуты справочника  var attributes = recordsProvider.GetAttributeNameOrderMap();    var records = new List<RecordDto>();  foreach (var employee in employees)  {  var newRow = CreateRow(dictionaryId.Value, employee, attributes);  records.Add(newRow);  }    //Обновляем справочник - удаляем все записи справочника и импортируем новые  await DictionaryImportService.ImportRecordsAsync<IEnumerable<RecordDto>>(dictionaryId.Value, records, 1, Guid.Empty, "Система");  } |

Пример создания записи справочника и заполнения атрибута:

|  |
| --- |
| private RecordDto CreateRow(Guid dictionaryId, IDictionary<string, int> attributes)  {  //Создаем DTO новой записи справочника  var recordDto = new RecordDto  {  DictionaryId = dictionaryId,  Code = "КодЗаписи",  Name = "НазваниеЗаписи"  };    //Заполняем атрибут  FillAttributeValue(attributes, recordDto, "КодАтрибута", "ЗначениеАтрибута");  }    //Пример заполнения атрибута  private void FillAttributeValue(IDictionary<string, int> attributes, RecordDto recordDto, string attributeCode, object attributeValue)  {  if (attributes.TryGetValue(attributeCode, out var number))  {  recordDto.AttributesByOrder[number] = attributeValue;  }  } |

## Инструкция по работе с масками ввода

### Основные правила масок ввода (регулярных выражений)

Основные правила масок ввода (регулярных выражений) приведены в таблице ниже (Таблица 68).

Таблица 68 – Правила масок ввода (регулярных выражений)

| **Символ** | **Описание** | **Пример** |
| --- | --- | --- |
| ^ | начало строки | ^\d{3} соответствует "123" в "123-1234" |
| $ | конец строки | -\d{3}$ соответствует "-123" в "1234-123" |
| . | любой одиночный символ, кроме \n | a.e соответствует "ave" в "brave", "ate" в "water" |
| \* | предыдущий символ повторяется 0 и более раз | \d\*\.\d соответствует ".0", "19.9", "219.9" |
| + | предыдущий символ повторяется 1 и более раз | "be+" соответствует "bee" в "beer", "be" в "best" |
| ? | предыдущий символ повторяется 0 или 1 раз | "rai?n" соответствует "ran", "rain" |
| \s | любой пробельный символ | \w\s соответствует "D " в "ID field" |
| \S | любой символ не пробел | \s\S соответствует " \_" в "private \_dataStore" |
| \w | любой алфавитно-цифровой символ | \w соответствует "H", "i" в "Hi!" |
| \W | любой символ не из алфавита и не цифра | \W соответствует "!" в "Hi!" |
| \d | любая десятичная цифра | \d соответствует "0", "1" в "Hell0 Wor1d!" |
| \D | любой символ не десятичная цифра | \D соответствует "3", "1" в "version 3.1" |
| [abc] | любой символ в группе | [ae] соответствует "a" в "hand"; "a", "e" в "name" |
| [ ^abc] | любой символ не из группы | [^ae] соответствует "n", "m" в "name" |
| [a-z] | любой символ из диапазона | [A-Z] соответствует "A", "B" в "AB123" |
| \p{Lu} | любой символ заглавной латиницы (Latin upper) | \p{Lu} соответствует "C", "L" в "City Lights" |
| \p{IsCyrillic} | любой символ кириллицы | \p{IsCyrillic} соответствует "Д", "Ж" в "ДЖem" |
| \P{Lu} | любой символ прописной латиницы | \P{Lu} соответствует "i", "t", "y" в "City" |
| \P{IsCyrillic} | любой символ не кириллицы | \P{IsCyrillic} соответствует "e", "m" в "ДЖem" |
| {n} | предыдущий символ повторяется n-раз | ",\d{3}" соответствует ",543" и ",210" в "6,543,210" |
| {n,} | предыдущий символ повторяется как минимум n-раз | "\d{2,}" соответствует "166", "29" |
| {n,m} | предыдущий символ повторяется как минимум n-раз, но не более чем m-раз | "\d{3,5}" соответствует "19302" в "193024" |
| | | любой элемент (логическое ИЛИ) | "th(e|is|at)" соответствует "the", "this" в "this is the day." |

### Дизайнер

Наиболее часто используемые маски ввода доступны для выбора в Дизайнере при формировании столбцов таблицы строкового типа (Рисунок 104).

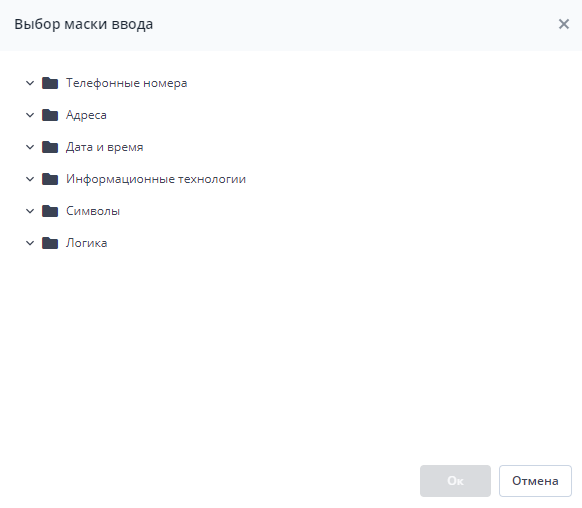


Рисунок 104 – Окно выбора маски ввода

Ниже представлено описание правил доступных масок ввода (Таблица 69 – Таблица 74).

Таблица 69 – Телефонные номера

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| 7-мизначный номер | \d\d\d-\d\d-\d\d | \_\_\_-\_\_-\_\_ |
| 7-мизначный номер с кодом города | \(\d{3,4}\)\d\d\d-\d\d-\d\d | (\_\_\_\_)\_\_\_-\_\_-\_\_ |
| 6-тизначный номер | \d\d-\d\d-\d\d | \_\_-\_\_-\_\_ |
| 6-тизначный номер с кодом города | \(\d{3,4}\)\d\d-\d\d-\d\d | (\_\_\_\_)\_\_-\_\_-\_\_ |
| Универсальный формат номера телефона | \(\d{3,5}\)\d{1,3}-\d\d-\d\d | (\_\_\_\_\_)\_\_\_-\_\_-\_\_ |

Таблица 70 – Адреса

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| РФ, Республика Татарстан, город $Город$, улица $Улица$, дом $дом$ | РФ, Республика Татарстан, город ([А-Я][а-я]+[ ]?)+?([-]\d{1,3})?, улица (\d{1,2})?([а-яА-Я -]+)?(\d{1,3})?, дом \d{1,3} | РФ, Республика Татарстан, город \_, улица \_, дом \_ |
| РФ, $Регион$, город $Город$, улица $Улица$, дом $дом$ | РФ, ([А-Я][а-яА-Я]+?[ -]?)+, город ([А-Я][а-я]+[ ]?)+?([-]\d{1,3})?, улица (\d{1,2})?([а-яА-Я -]+)?(\d{1,3})?, дом \d{1,3} | РФ, \_, город \_, улица \_, дом \_ |
| $Регион$, город $Город$, улица $Улица$, дом $дом$ | ([А-Я][а-яА-Я]+?[ -]?)+, город ([А-Я][а-я]+[ ]?)+?([-]\d{1,3})?, улица (\d{1,2})?([а-яА-Я -]+)?(\d{1,3})?, дом \d{1,3} | \_, город \_, улица \_, дом \_ |
| город $Город$, улица $Улица$, дом $дом$ | город ([А-Я][а-я]+[ ]?)+?([-]\d{1,3})?, улица (\d{1,2})?([а-яА-Я -]+)?(\d{1,3})?, дом \d{1,3} | город \_, улица \_, дом \_ |

Таблица 71 – Дата и время

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| Дата ДД.ММ.ГГ | ([012]?[1-9]|[123]0|31)\.(0?[1-9]|1[012])\.[0-9]{2} | ДД.ММ.ГГ |
| Дата ДД.ММ.ГГГГ | ([012]?[1-9]|[123]0|31)\.(0?[1-9]|1[012])\.([0-9]{4}) | ДД.ММ.ГГГГ |
| Дата ДД/ММ/ГГ | ([012]?[1-9]|[123]0|31)/(0?[1-9]|1[012])/[0-9]{2} | ДД/ММ/ГГ |
| Дата ДД/ММ/ГГГГ | ([012]?[1-9]|[123]0|31)/(0?[1-9]|1[012])/[0-9]{4} | ДД/ММ/ГГГГ |
| Время | (0?\d|1\d|2[0-3])\:[0-5]\d | ЧЧ:ММ |
| Время с секундами | (0?\d|1\d|2[0-3]):[0-5]\d:[0-5]\d | ЧЧ:ММ:СС |

Таблица 72 – Информационные технологии

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| Адрес E-Mail | [a-z0-9][a-z0-9.\_-]+@[a-z0-9][a-z.0-9\_-]+\.\p{Ll}{2,3} | \_\_@\_\_.\_\_ |
| Адрес E-Mail в домене RU | [a-z0-9][a-z0-9.\_-]+@[a-z][a-z.0-9\_-]+\.ru | \_\_@\_\_.ru |

Таблица 73 – Символы

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| Все символы | .+ | Все символы |
| Только буквы | \p{L}+ | Только буквы |
| Только буквы в верхнем регистре | \p{Lu}+ | Только буквы в верхнем регистре |
| Только буквы в нижнем регистре | \p{Ll}+ | Только буквы в нижнем регистре |
| Латинские символы | [a-zA-Z]+ | Латинские символы |
| Латинские символы и цифры | [a-zA-Z0-9]+ | Латинские символы и цифры |
| Кириллические символы | [а-яА-Я0-9]+ | Кириллические символы |
| Латинские и кириллические символы, цифры | [a-zA-Zа-яА-Я0-9]+ | Латинские и кириллические символы, цифры |

Таблица 74 – Логика

|  |  |  |
| --- | --- | --- |
| **Текст** | **Формат** | **Подсказка** |
| Да / Нет | Да|Нет | Да|Нет |
| Истина / Ложь | Истина|Ложь | Истина|Ложь |
| Yes / No | Yes|No | Yes|No |
| True / False | True|False | True|False |

### Реализация интерфейса

При реализации собственной части интерфейса есть возможность использовать форматы Ext JS (Рисунок 105).

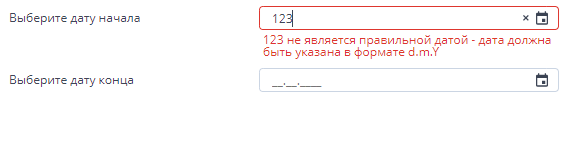


Рисунок 105 – Пример использования формата Ext JS

Тип элемента – datefield. Формат - d.m.Y. Ниже приведен пример кода:

|  |
| --- |
| Ext.define('Svody.apiJs.view.TestWindow', {  extend: 'Ext.window.Window',  xtype: 'testWindow',  letterSize: 'm',    initComponent () {  Ext.apply(this, {  title: this.locale.title,  defaults: {  labelWidth: 250,  width: 550  },  items: [  {  xtype: 'datefield',  fieldLabel: this.locale.startDateLabelText,  emptyText: '\_\_.\_\_.\_\_\_\_',  format: 'd.m.Y',  name: 'startDate',  reference: 'startDate',  allowBlank: true  },  {  xtype: 'datefield',  fieldLabel: this.locale.endDateLabelText,  emptyText: '\_\_.\_\_.\_\_\_\_',  format: 'd.m.Y',  name: 'endDate',  reference: 'endDate',  allowBlank: true  },  ],  buttons: [  {  text: this.locale.cancelButtonText,  ui: 'btn-secondary',  handler: 'close'  },  ]  });  this.callParent();  }  }); |

Подробнее дополнительный интерфейс описан в п. 28.20.

## MultiDb

Реализована возможность подключения приложения к нескольким БД одновременно.

При старте приложения, если в файле параметров подключения «UseArchiveDatabases» не задан явно, или его значение true - приложение будет подключено к основной БД вместе со всеми архивными (шардами).

Все существующее API платформы для выполнения запросов по умолчанию выполняет запросы в текущем контексте исполнения запросов (QueryContext).

Невозможно выполнение запросов содержащих команды «delete», «insert», «update», «merge», «truncate», «drop» в архивных БД.

### Получение параметров подключения

GetQuery:

|  |
| --- |
| var dbConnParametersProvider = КонтейнерЗависимостей.Текущий.Resolve<IDbConnectionParametersProvider>();    dbConnParametersProvider.AllConnections; // поключения ко всем доступным инстансам БД  dbConnParametersProvider.SecondaryConnections; // подключения ко всем архивным инстансам БД  dbConnParametersProvider.PrimaryConnection; // подключение к актуальному инстансу БД |

Описание возвращаемого типа приведено ниже.

ConnectionParameters:

|  |
| --- |
| public sealed class ConnectionParameters  {  internal ConnectionParameters()  {    }    /// <summary> Тип СУБД </summary>  public string DbDialect { get; set; }    /// <summary> Пулинг соединений на БД </summary>  public bool EnablePooling { get; set; }    /// <summary> Хост </summary>  public string Host { get; set; }    /// <summary> Порт </summary>  public string Port { get; set; }    /// <summary> Имя пользователя БД </summary>  public string UserName { get; set; }    /// <summary> Пароль пользователя БД </summary>  public string Password { get; set; }    /// <summary> Имя БД </summary>  public string DbName { get; set; }    /// <summary> Имя схемы БД </summary>  public string SchemaName { get; set; }    /// <summary> Использовать имя сервера как TNS </summary>  public string UseServerNameAsTns { get; set; }    /// <summary> Использовать TNS </summary>  public bool UseTns => bool.TryParse(UseServerNameAsTns, out var result) && result;    /// <summary> Дополнительные параметры подключения </summary>  public IDictionary<string, string> AdditionalConnectionParameters { get; set; }    /// <summary> Ключ-идентификатор БД </summary>  public string DbKey { get; set; }    /// <summary> Ключ-идентификатор родительской БД </summary>  public string ParentDbKey { get; set; }    /// <summary> Дата начала действия данных в БД </summary>  public DateTime DataStart { get; set; }    /// <summary> Дата окончания действия данных в БД </summary>  public DateTime DataEnd { get; set; }    /// <summary> Подключение является основным </summary>  public bool IsPrimary { get; set; }    /// <summary> Строка подключения к БД </summary>  public string ConnectionString { get; set; }    /// <summary> Приоритет подключения </summary>  public int Priority { get; set; }    /// <summary> Подключение только для чтения </summary>  /// <returns>true - если при подключении к БД запрещено изменять данные</returns>  public bool IsReadOnly() => DbKey != QueryContext.Default.DbKey;    /// ... Остальные члены  } |

В возвращаемом типе содержится свойство «DbKey», которое может использоваться для выполнения запросов в конкретных инстансах БД.

Так же в класс «ХранимыйОбъект» добавлено свойство «DbKey», которое отображает, из какой БД был получен хранимый объект, оно заполняется автоматически платформой после материализации любой сущности.

В случае использования свойств типа «ПодчиненнаяВыборка», запросы будут направляться в ту БД, из которой был получен родительский объект.

### API платформы

Для выполнения запросов к разным БД реализовано новое API платформы.

#### QueryContext

Класс «QueryContext» отвечает за контекст исполнения запросов к БД в текущем потоке приложения. Обычно задается платформой.

Best practice для выполнения запросов в нужном контексте.

SetCurrentContext:

|  |
| --- |
| var dbConnParametersProvider = КонтейнерЗависимостей.Текущий.Resolve<IDbConnectionParametersProvider>();    var archiveConn = dbConnParametersProvider.SecondaryConnections.FirstOrDefault();    using (QueryContext.BeginScope(archiveConn))  {  // этот код будет выполняться в контексте архивной БД  }    // код тут будет выполняться в контексте по умолчанию - главная БД |

#### DataStore

##### GetQueryFromAllDb

GetQueryFromAllDb:

|  |
| --- |
| /// <summary> Получить объект запроса ко всем инстансам БД </summary>  /// <param name="collisionResolver">Стратегия решения коллизий</param>  /// <typeparam name="TEntity">Тип хранимого объекта</typeparam>  /// <returns>IQueryable-запрос</returns>  IQueryable<TEntity> GetQueryFromAllDb<TEntity>(ICollisionResolver collisionResolver = null)  where TEntity : class, new(); |

Метод выполняет запрос ко всем доступным инстансам БД.

ICollisionResolver collisionResolver – стратегия для решения коллизий результатов запросов. По умолчанию используется DefaultCollisionResolver. Выполняется сортировка полученных результатов по приоритету БД и возвращаются все, исключая повторяющиеся, идентификаторы

##### GetQuery

GetQuery:

|  |
| --- |
| /// <summary> Получить объект запроса к конкретному инстансу к БД </summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <typeparam name="TEntity">Тип хранимого объекта</typeparam>  /// <returns>IQueryable-запрос</returns>  IQueryable<TEntity> GetQuery<TEntity>(string dbKey)  where TEntity : class, new(); |

Метод выполняет запрос к конкретному инстансу БД.

string dbKey – ключ-идентификатор БД, к которой необходимо выполнить запрос (получение ключа описано в п. 28.17.1).

#### Выборка

Новые методы:

|  |
| --- |
| /// <summary> Проверить, содержит ли выборка записи при загрузке из всех БД </summary>  /// <param name="withoutCollisions">Необходимость решения коллизий</param>  /// <returns>Возвращает True, если в выборке есть записи, иначе - False.</returns>  public bool ContainsRecordsInAllDb(bool withoutCollisions = true)    /// <summary> Проверить, содержит ли выборка записи при загрузке из конкретной БД </summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <returns>Возвращает True, если в выборке есть записи, иначе - False.</returns>  public bool ContainsRecordsInDb(string dbKey)    /// <summary> Возвращает количество записей в выборке при загрузке из всех БД. </summary>  /// <reremarks> Если выборка загружена, берет из число записей в коллекции, иначе - выполняет запрос. </reremarks>  public int GetRecordsCountFromAllDb(bool withoutCollisions = true)    /// <summary> Возвращает количество записей в выборке при загрузке из конкретной БД. </summary>  /// <reremarks> Если выборка загружена, берет из число записей в коллекции, иначе - выполняет запрос. </reremarks>  public int GetRecordsCountFromDb(string dbKey)    /// <summary> Загрузить из всех подключенных БД </summary>  /// <param name="withoutCollisions">Необходимость решения коллизий</param>  public override void LoadFromAllDb(bool withoutCollisions = true)    /// <summary> Загрузить из конкретной БД </summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  public override void LoadFromDb(string dbKey)    /// <summary> Загрузить полностью из всех подключенных БД </summary>  /// <param name="withoutCollisions">Необходимость решения коллизий</param>  public override void FullLoadFromAllDb(bool withoutCollisions = true)    /// <summary> Загрузить полностью из конкретной БД </summary>  /// <param name="dbKey"></param>  public override void FullLoadFromDb(string dbKey)    /// <summary>  /// Загрузить все записи из всех БД без постраничного режима,  /// даже если состояние выборки <see cref="Выборка.СостояниеВыборки.ЗагруженаПолностью"/>  /// </summary>  public void FullReloadFromAllDb()    /// <summary>  /// Загрузить все записи из конкретной БД без постраничного режима,  /// даже если состояние выборки <see cref="Выборка.СостояниеВыборки.ЗагруженаПолностью"/>  /// </summary>  public void FullReloadFromDb(string dbKey) |

#### SQLЗапрос

Новые методы:

|  |
| --- |
| /// <summary> Выполнить запрос во всех подключенных БД и получить курсор по записям </summary>  /// <returns>Табличный курсор</returns>  public ТабличныйКурсор ExecuteInAllDb()    /// <summary> Выполнить запрос в конкретной БД и получить курсор по записям </summary>  /// <returns>Табличный курсор</returns>  public ТабличныйКурсор ExecuteInDb(string dbKey)    /// <summary>Выполнить запрос с ограничением по времени во всех подключенных БД</summary>  /// <param name="timeout">Ограничение на время выполнения запроса в секундах</param>  /// <returns>Табличный курсор</returns>  public ТабличныйКурсор ExecuteInAllDbWithTimeout(int? timeout = null)    /// <summary>Выполнить запрос с ограничением по времени</summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <param name="timeout">Ограничение на время выполнения запроса в секундах</param>  /// <returns>Табличный курсор</returns>  public ТабличныйКурсор ExecuteInDbWithTimeout(string dbKey, int? timeout = null)    /// <summary>Выполнить запрос скалярно с ограничением по времени</summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <param name="timeout">Ограничение на время выполнения запроса в секундах</param>  /// <returns>Результат выполнения запроса</returns>  public object ExecuteScalarInDbWithTimeout(string dbKey, int? timeout = null)    /// <summary>Выполнить запрос как команду с ограничением по времени</summary>  /// <param name="dbKey">Ключ-идентификатор БД</param>  /// <param name="timeout">Ограничение на время выполнения запроса в секундах</param>  /// <returns>Результат выполнения запроса</returns>  public int ExecuteAsCommandInDbWithTimeout(string dbKey, int? timeout = null) |

## Новые дизайнерские увязки

### Новый раздел в дизайнере

Создание новых дизайнерских увязок представлено в новой вкладке в дизайнере отчетных форм.

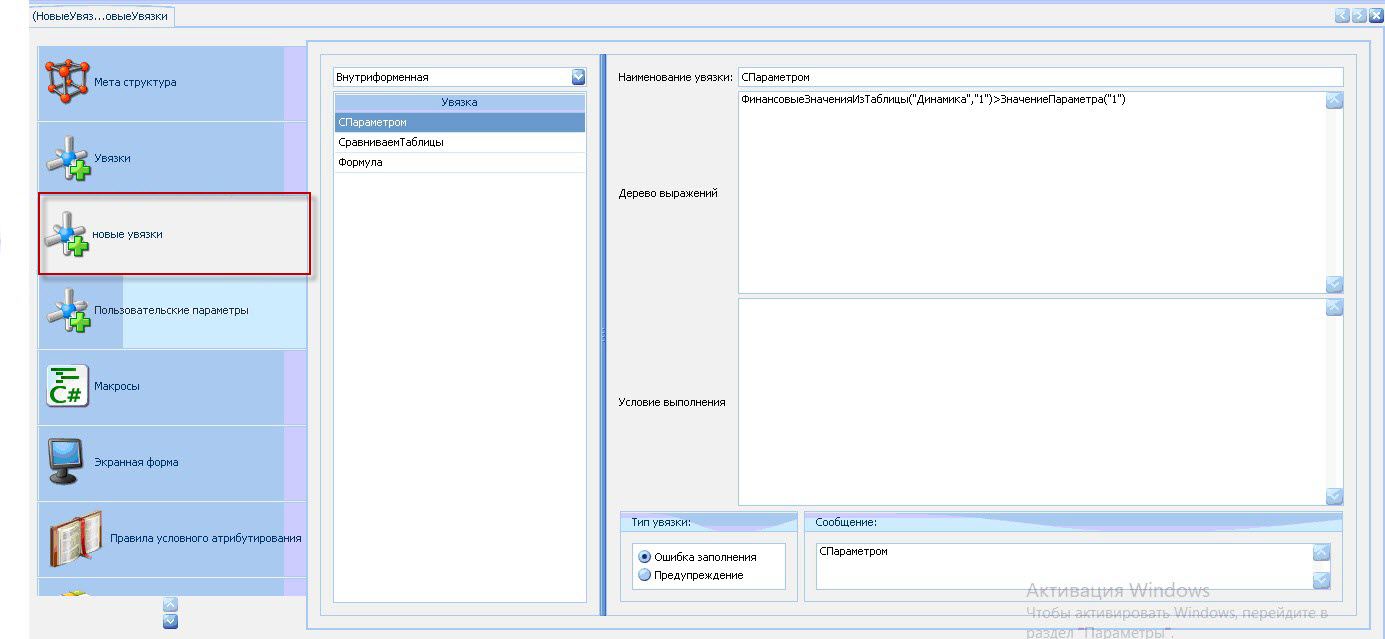


Рисунок 106 – Вкладка «Новые увязки»

|  |  |
| --- | --- |
| **Поле** | **Предназначение** |
| Наименование | Наименование увязки, отображаемое в протоколе проверки. |
| Дерево выражений | Набор функций и операций между функциями, в результате которых должно получиться логическое значение или список логических значений. |
| Условие выполнения | Набор функций и операций между функциями, в результате которых должно получиться логическое значение. |
| Тип ошибки | Тип ошибки проверки, указывает в протоколе тип ошибки проверки. Предупреждение или ошибка. |
| Сообщение | Сообщение увязки, отображаемое в протоколе проверки. |

### Написание увязок

Основными функциями увязок являются:

* «ФинансовыеЗначенияИзТаблицы»;
* «СтроковыеЗначенияИзТаблицы»;
* «СуммаПоСубтаблице»;
* «ПолучитьАттрибутИзДинамики»;
* «ПолучитьАттрибутИзСтатики»;
* «ЗначениеЯчейки».

Остальные функции используются для уточнения данных и их местоположения (другой отчетный период или компонент, или другая форма).

Основная функция, используемая в увязках – ФинансовыеЗначенияИзТаблицы.

Перед написанием увязки необходимо учесть следующие моменты:

* если сравнение происходит в разных формах, то необходимо определить, лежит ли другая форма в том же отчетном периоде и компоненте (в пределах текущего компонента). Если нет, то необходимо определить, как получать компонент;
* если заранее неизвестны код компонента и период, то необходимо использовать пользовательские параметры;
* необходимо ли отфильтровать строки по полям или нет;
* нужна ли группировка строк по ключевым полям для сравнения по ключам.

После выберите функцию с нужным количеством параметров и сконфигурируйте функцию, используя параметры и функции.

**Примечание** – Все параметры функции должны быть заключены в двойные кавычки, как на примерах выше (см. Рисунок 106). Названия вложенных функций нельзя заключать в кавычки, иначе Система посчитает этот параметр обычным текстом, а не функцией.

### Функции, операнды и операции между функциями

#### Общие сведения

Функция – минимальная операция получения объекта или данных. Функции могут принимать параметры для уточнения получаемых данных или получаемого объекта. В качестве параметров функций выступают текстовые значения или другие функции.

В таблице ниже представлен список функций Системы.

Таблица 75 – Функции Системы

| № | Функция | Параметры | Возвращаемый операнд | Описание функции | Пример |
| --- | --- | --- | --- | --- | --- |
| 1 | **СуммаПоСубтаблице** | а)Код таблицы, код графы  б)Код таблицы, код графы,функция форма  в)Код таблицы, код графы,графы фильтра, значение фильтра  г)Код таблицы, код графы,графы фильтра, значение фильтра, функция форма | Численный операнд | **Примечание** – Функция для динамических таблиц. Не рекомендуется использование со статической таблицей.  а)Возвращает сумму всех строк указанной таблицы по указанной графе в текущей форме  б)Возвращает сумму всех строк указанной таблицы по указанной графе в указанной форме  в)Возвращает сумму всех строк по фильтру указанной таблицы по указанной графе в текущей форме  г)Возвращает сумму всех строк по фильтру указанной таблицы по указанной графе в указанной форме | а)СуммаПоСубтаблице("Динамика","1")  б)СуммаПоСубтаблице("Динамика","1", Форма("КодФормы"))  в)СуммаПоСубтаблице("Динамика","1","ППП","125")  г)СуммаПоСубтаблице("Динамика","1","ППП","125",  Форма("КодФормы")) |
| 2 | ОтчетныйПериод | а)код  б)Дата начала, тип периода (Годовой,Полугодовой,Квартальный  Месячный,Недельный,Разовый) | ReportPeriodOperandType | а)Возвращает отчетный период по коду. Игнорируются периоды, у которых установлена настройка НеИспользуется.  б)Возвращает первый отчетный период с указанной датой и типом  Игнорируются периоды, у которых установлена настройка НеИспользуется. | а)ОтчетныйПериод("Годовая отчетность 2019")  б)ОтчетныйПериод("01.01.2019","Годовой") |
| 3 | ПредыдущийОП | Без параметров | ReportPeriodOperandType | Возвращает предыдущий Отчетный период, ориентируется на дату начала в зависимости от типа. Текущий – (1год/6мес/3мес/1мес)  Игнорируются периоды, у которых установлена настройка НеИспользуется. | ПредыдущийОП() |
| 4 | АналогичныйПрошлогоднийОП | Без параметров | ReportPeriodOperandType | Возвращает аналогичный период. Ищет по типу и дате начала. Тип как у текущего, дата начала текущий - 1 год  Игнорируются периоды, у которых установлена настройка НеИспользуется. | АналогичныйПрошлогоднийОП() |
| 5 | Компонент | а)Код компонента  б)Код компонента, функция получения отчетного периода  в)Функция получения отчетного периода | ComponentPeriodOperandType | а)Возвращает компонент по коду в текущем отчетном периоде.  Игнорируются компоненты, у которых установлена настройка НеИспользуется.  б)Возвращает компонент по коду в указанном отчетном периоде.  Игнорируются компоненты, у которых установлена настройка НеИспользуется.  в)Возвращает компонент по текущему коду в указанном отчетном периоде.  Игнорируются компоненты, у которых установлена настройка НеИспользуется. | а)Компонент("Автономные учреждения")  б)Компонент("Автономные учреждения", ПредыдущийОП())  в)Компонент(ПредыдущийОП()) |
| 6 | КомпонентСФормой | а)Код формы  б)Код формы, функция получения отчетного периода | ComponentPeriodOperandType | а)Возвращает первый компонент в текущем отчетном периоде в пакете форм которого присутствует указанная форма.  Игнорируются компоненты, у которых установлена настройка НеИспользуется.  б)Возвращает первый компонент в указанном отчетном периоде в пакете форм которого присутствует указанная форма.  Игнорируются компоненты, у которых установлена настройка НеИспользуется. | а)КомпонентСФормой("0503769")  б)КомпонентСФормой("0503769",ПредыдущийОП()) |
| 7 | КомпонентФормыпоГрЭквивалентности | Код формы | ComponentPeriodOperandType | Возвращает первый Компонент с аналогичной группой эквивалетности как и в текущем компоненте в текущем ОП в пакете форм которого есть форма с указанным кодом.  Игнорируются компоненты, у которых установлена настройка НеИспользуется. | КомпонентФормыпоГрЭквивалентности("0503769") |
| 8 | Форма | а)Код формы  б)Код формы, функция получения компонента отчетного периода | FormOperandType | а)Возвращает форму с указанным кодом в текущем компоненте. На текущем учреждении  б)Возвращает форму с указанным кодом в указанном компоненте. На текущем учреждении | а)Форма("0503769")  б)Форма("0503769",Компонент("Автономные учреждения"))  б)Форма("0503769",Компонент("Автономные учреждения",ПредыдущийОП())) |
| 9 | Набор | От 1 до n параметров | ТипОперандаМассив | Возвращает список значений, сформированный на основе параметров. | Набор("1","2","3") |
| 10 | Диапазон | Начало диапазона, конец диапозона | ТипОперандаМассив | Возвращает список значений от начала диапазона до конца. Шаг +1. Начало и конец включаются | Диапазон("1","10") |
| 11 | ЗначениеПараметра | Идентификатор параметра | Зависит от параметра | Возвращает значение пользовательского параметра. Конфигурирование параметров описано ниже. | ЗначениеПараметра("id1") |
| 12 | ПроверитьЛицензию | Код учреждения | ТипОперандаЛогический | Выполняет проверку кода учреждения с учреждением, которое записано в лицензии. | ПроверитьЛицензию("Министерство Информатизации Республики Татарстан") |
| 13 | ЗначениеЯчейки | а)Код таблицы, КодСтолбца, КодСтроки  б)Код таблицы, КодСтолбца, КодСтроки, Функция получения формы | ТипОперандаЧисленный | а)Возвращает значение ячейки в указанной строке в указанной таблице для текущей формы  б)Возвращает значение ячейки в указанной строке в указанной таблице для указанной формы | а)ЗначениеЯчейки("Кредиторская","5","050")  б)ЗначениеЯчейки("Кредиторская","5","050",  Форма("0503769"))  б)ЗначениеЯчейки("Кредиторская","5","050",  Форма("0503769",Компонент("Автономные учреждения")))  б)ЗначениеЯчейки("Кредиторская","5","050",  Форма("0503769",Компонент("Автономные учреждения", ПредыдущийОП()))) |
| 14 | ФинансовыеЗначенияИзТаблицы | а)Код таблицы,код столбца или функция Набор для указания нескольких столбцов  б)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,функция получения формы  в)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,код графы ключа или функция Набор для указания нескольких столбцов  г)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,код графы ключа или функция Набор для указания нескольких столбцов,функция получения формы  д)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра  е)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра, функция получения формы  ё)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра,  код графы ключа или функция Набор для указания нескольких столбцов  ж)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра,  код графы ключа или функция Набор для указания нескольких столбцов,функция получения формы | DictionaryOperandType | **Примечание** – Функция для динамических таблиц. Не рекомендуется использование со статической таблицей.  а)Возвращает финансовое значение указанного столбца/столбцов для каждой строки в текущей форме в указанной таблице.  б)Возвращает финансовое значение указанного столбца/столбцов для каждой строки в указанной форме в указанной таблице.  в)Возвращает финансовое значение указанного столбца/столбцов сгруппированных по указанным ключевым столбцам в текущей форме в указанной таблице.  г)Возвращает финансовое значение указанного столбца/столбцов сгруппированных по указанным ключевым столбцам в указанной форме в указанной таблице.  д)Возвращает финансовое значение указанного столбца/столбцов для каждой строки подходящей по фильтру в текущей форме в указанной таблице.  е)Возвращает финансовое значение указанного столбца/столбцов для каждой строки подходящей по фильтру в указанной форме в указанной таблице.  ё)Возвращает финансовое значение указанного столбца/столбцов сгруппированных по указанным ключевым столбцам подходящим по фильтру в текущей форме в указанной таблице  ж)Возвращает финансовое значение указанного столбца/столбцов сгруппированных по указанным ключевым столбцам подходящим по фильтру в указанной форме в указанной таблице. | а)ФинансовыеЗначенияИзТаблицы("Динамика","2") б)ФинансовыеЗначенияИзТаблицы("Динамика","2",Форма("НовыеУвязки",ЗначениеПараметра("2")) в)ФинансовыеЗначенияИзТаблицы("Динамика","2","КлючГрафа") г)ФинансовыеЗначенияИзТаблицы("Динамика","2",Набор("ЦС","ВР"), Форма("НовыеУвязки",ЗначениеПараметра("2"))) д)ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),"0000120") е)ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),"0000120",Форма("НовыеУвязки",ЗначениеПараметра("2")) ё)ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),"0000120","КБК") ж)ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),"0000120","КБК",Форма("НовыеУвязки",ЗначениеПараметра("2")) |
| 15 | СтроковыеЗначенияИзТаблицы | а)Код таблицы,код столбца или функция Набор для указания нескольких столбцов  б)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  функция получения формы  в)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра  г)Код таблицы,код столбца или функция Набор для указания нескольких столбцов,  графы фильтра, значение фильтра, функция получения формы | DictionaryOperandType | **Примечание** – Функция для динамических таблиц. Не рекомендуется использование со статической таблицей.  а)Возвращает значение строкой указанного столбца для каждой строки в текущей форме в указанной таблице.  б)Возвращает значение строкой указанного столбца для каждой строки в указанной форме в указанной таблице.  в)Возвращает значение строкой указанного столбца для каждой строки подходящей по фильтру в текущей форме в указанной таблице.  г)Возвращает значение строкой указанного столбца для каждой строки подходящей по фильтру в указанной форме в указанной таблице. | а)СтроковыеЗначенияИзТаблицы("КредиторскаяСуб","ППП") б)СтроковыеЗначенияИзТаблицы("КредиторскаяСуб","ППП", Форма("0503769",Компонент(ПредыдущийОП()))) в)СтроковыеЗначенияИзТаблицы("КредиторскаяСуб","ППП","ЦС","000") г)СтроковыеЗначенияИзТаблицы("КредиторскаяСуб","ППП","ЦС","000", Форма("0503769",Компонент(ПредыдущийОП()))) |
| 16 | ПолучитьАттрибутИзСтатики | а)Код таблицы,КодСтолбца,  КодСтроки,КодАттрибута  б)Код таблицы,КодСтолбца,  КодСтроки,КодАттрибута,  Функция получения формы | ТипОперандаСтроковый | а)Возвращает значение атрибута справочника в указанной строке в указанной таблице для текущей формы  б)Возвращает значение атрибута справочника в указанной строке в указанной таблице для указанной формы | а)ПолучитьАттрибутИзСтатики("Кредиторская","ППП",  "020","ГлаваПоБК")  б)ПолучитьАттрибутИзСтатики("Кредиторская","ППП",  "020","ГлаваПоБК",Форма("0503769"))  б)ПолучитьАттрибутИзСтатики("Кредиторская","ППП",  "020","ГлаваПоБК",Форма("0503769",Компонент("Автономные учреждения")))  б)ПолучитьАттрибутИзСтатики("Кредиторская","ППП",  "020","ГлаваПоБК",Форма("0503769",Компонент("Автономные учреждения", ПредыдущийОП()))) |
| 17 | ПолучитьАттрибутИзДинамики | а)Код таблицы,код столбца,  код аттрибута  б)Код таблицы,код столбца,  код аттрибута, функция получения формы  в)Код таблицы,код столбца,  код аттрибута, графы фильтра, значение фильтра  г)Код таблицы,код столбца,  код аттрибута, графы фильтра, значение фильтра, функция получения формы | DictionaryOperandType | а)Возвращает значение аттрибута справочника для указанного столбца для каждой строки динамической таблицы в текущей форме.  б)Возвращает значение аттрибута справочника для указанного столбца для каждой строки динамической таблицы в указанной форме.  в)Возвращает значение аттрибута справочника для указанного столбца для каждой строки подходящей по фильтру динамической таблицы в текущей форме.  г)Возвращает значение аттрибута справочника для указанного столбца для каждой строки подходящей по фильтру динамической таблицы в указанной форме. | а)ПолучитьАттрибутИзДинамики("КредиторскаяСуб","ППП","ГлаваПоБК") б)ПолучитьАттрибутИзДинамики("КредиторскаяСуб","ППП","ГлаваПоБК",  Форма("0503769",Компонент(ПредыдущийОП()))) в)ПолучитьАттрибутИзДинамики("КредиторскаяСуб","ППП","ГлаваПоБК","ЦС","000") г)ПолучитьАттрибутИзДинамики("КредиторскаяСуб","ППП","ГлаваПоБК","ЦС","000",  Форма("0503769",Компонент(ПредыдущийОП()))) |
| 18 | КоличествоСтрок | а)Код таблицы  б)Код таблицы, функция получения формы | ТипОперандаЧисленный | а)Возвращает количество строк в указанной таблице в текущей форме  б)Возвращает количество строк в указанной таблице в указанной форме | а)КоличествоСтрок("КредиторскаяСуб")  б)КоличествоСтрок("КредиторскаяСуб",  Форма("0503769"))  б)КоличествоСтрок("КредиторскаяСуб",   Форма("0503769",Компонент("Автономные учреждения")))  б)КоличествоСтрок("КредиторскаяСуб",  Форма("0503769",Компонент("Автономные учреждения", ПредыдущийОП())))г |

Операнд – представление данных или объекта получаемых с помощью функций. Все функции и операции между функциями оперируют операндами. Значения констант также являются операндами.

В таблице ниже представлен список операндов Системы.

Таблица 76 – Операнды Системы

| **Операнд** | **Назначение** |
| --- | --- |
| ComponentPeriodOperandType | Операнд для представления результирующего операнда с типом КомпонентОтчетногоПериода.  Хранит в себе Компонент отчетного периода |
| ReportPeriodOperandType | Операнд для представления результирующего операнда с типом ОтчетныйПериод  Хранит в себе Отчетный период |
| FormOperandType | Операнд для представления результирующего операнда с типом ДанныеФормы  Хранит в себе данные формы |
| DictionaryOperandType | Операнд для представления результирующего операнда с типом Dictionary  Хранит в себе список пар, ключ значение. Ключ - строка, значение Операнд.  Используется в функциях для возврата списка сгруппированных строк по ключевым полям |
| ТипОперандаЛогический | Операнд для представления результирующего операнда с типом Логика.  Значение true/false; Истина или ложь |
| ТипОперандаМассив | Операнд для представления результирующего операнда с типом Массив.  Хранит в себе список операндов |
| ТипОперандаСтроковый | Операнд для представления результирующего операнда с типом Строка.  Хранит в себе строку |
| ТипОперандаЦелочисленный | Операнд для представления результирующего операнда с типом Целое число.  Хранит в себе целое число. Без дробных показателей |
| ТипОперандаЧисленный | Операнд для представления результирующего операнда с типом Число.  Хранит в себе Число. Может хранить дробные показатели |

#### Для разработки

Для реализации собственных функций создайте интерфейс «IChecksFunctionHandler». Реализацию интерфейса зарегистрируйте в IOC-контейнере с именем функции, которая будет использоваться в увязках:

|  |
| --- |
| public interface IChecksFunctionHandler  {  ///Вычислить значение функции  IТипОперанда Handler(string function, List<ПараметрВыраженияФункции> parameters,  БазовыйОбработчикПеременных variableHandler, Dictionary<string, object> globalParams);    ///Получить значение для расшифровки  ЗначениеПеременнойЯчейкиФормы GetTranscript(int index,  string key,  ВыражениеФункция function,  IТипОперанда value,  ПровайдерДанныхФормДляРасчетаУвязок dataProvider,  БазовыйОбработчикПеременных variableHandler,  ВычислительВыражений checkHandler);    } |

|  |
| --- |
| Component.For<IChecksFunctionHandler>().ImplementedBy<DynamicTableFinancialValuesFunctionHandler>().Named("ФинансовыеЗначенияИзТаблицы").LifestyleSingleton(), |

##### Метод «Handler»

Данный метод вызывается для получения значения функции, после того как все вложенные функции в этой функции уже были вычислены.

Таблица 77 – Метод «Handler»

|  |  |
| --- | --- |
| Параметр | Назначение |
| function | Строковое представление функции |
| parameters | Параметры функции, которые введены в скобках в дереве выражений увязки. Параметры функции уже буду вычислены. Для получения значения параметра в виде IТипОперанда, использовать variableHandler. |
| variableHandler | Используется для получения значений из parameters в виде IТипОперанда. Например, variableHandler.ПолучитьЗначение(parameters[0]) |
| globalParams | Содержит глобальные параметры проверки увязок. В данный момент содержит только ПровайдерДанныхФормДляРасчетаУвязок. Провайдер для получения текущей формы, текущего компонента или для поиска компонентов или ОП по различным параметрам. |

Для получения провайдера используйте следующий код:

|  |
| --- |
| ПровайдерДанныхФормДляРасчетаУвязок dataProvider;  //Получили провайдер данных  if (globalParams.TryGetValue(ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра, out var данные))  {  dataProvider = данные as ПровайдерДанныхФормДляРасчетаУвязок;  if (dataProvider == null)  {  throw new InvalidCastException(string.Format(ResourcesStrings.НеУдалосьПривестиПараметрКТипу, ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра, typeof(ПровайдерДанныхФормДляРасчетаУвязок)));  }  }  else  {  throw new ArgumentException(string.Format(ResourcesStrings.НеНайденГлобальныйПараметр, ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра));  } |
| public IТипОперанда Handler(string function, List<ПараметрВыраженияФункции> parameters, БазовыйОбработчикПеременных variableHandler, Dictionary<string, object> globalParams)  {  ПровайдерДанныхФормДляРасчетаУвязок dataProvider;    //Получили провайдер данных  if (globalParams.TryGetValue(ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра, out var данные))  {  dataProvider = данные as ПровайдерДанныхФормДляРасчетаУвязок;  if (dataProvider == null)  {  throw new InvalidCastException(string.Format(ResourcesStrings.НеУдалосьПривестиПараметрКТипу, ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра, typeof(ПровайдерДанныхФормДляРасчетаУвязок)));  }  }  else  {  throw new ArgumentException(string.Format(ResourcesStrings.НеНайденГлобальныйПараметр, ПровайдерДанныхФормДляРасчетаУвязок.ИмяПараметра));  }    if (parameters.Count < MinimumParameterCount || parameters.Count > MaximumParameterCount)  {  throw new ArgumentException($"Некорректное количество параметров функции. Функция {function} принимает от 2 до 6 параметров");  }    FormOperandType formOperand = null;    var tableCode = variableHandler.ПолучитьЗначение(parameters[0]).Значение as string;  var columns = ParseColumns(variableHandler, parameters, 1);  var keyColumns = new List<string>();  var filterColumns = new List<string>();  var filterValue = string.Empty;  var form = dataProvider.GetCurrentForm;    switch (parameters.Count)  {  case 3:  //Таблица,Графы,Форма  formOperand = variableHandler.ПолучитьЗначение(parameters[2]) as FormOperandType;  if (formOperand!=null)  {  form = formOperand.Значение as ДанныеФормы;  }  //Таблица,Графы,КлючГрафы  else  {  keyColumns = ParseColumns(variableHandler, parameters, 2);  }  break;  case 4:  //Таблица,Графы,КлючГрафы,Форма  formOperand = variableHandler.ПолучитьЗначение(parameters[3]) as FormOperandType;  if (formOperand!=null)  {  form = variableHandler.ПолучитьЗначение(parameters[3]).Значение as ДанныеФормы;  keyColumns = ParseColumns(variableHandler, parameters, 2);  }  //Таблица,Графы,ФильтрГрафы,ФильтрЗначение  else  {  filterColumns = ParseColumns(variableHandler, parameters, 2);  filterValue = variableHandler.ПолучитьЗначение(parameters[3]).Значение.ToString();  }  break;  case 5:  filterColumns = ParseColumns(variableHandler, parameters, 2);  filterValue = variableHandler.ПолучитьЗначение(parameters[3]).Значение.ToString();    //Таблица,Графы,ФильтрГрафы,ФильтрЗначение,Форма  formOperand = variableHandler.ПолучитьЗначение(parameters[4]) as FormOperandType;  if (formOperand!=null)  {  form = formOperand.Значение as ДанныеФормы;  }  //Таблица,Графы,ФильтрГрафы,ФильтрЗначение,КлючГрафы  else  {  keyColumns = ParseColumns(variableHandler, parameters, 4);  }  break;  case 6:  //Таблица,Графы,ФильтрГрафы,ФильтрЗначение,КлючГрафы,Форма  filterColumns = ParseColumns(variableHandler, parameters, 2);  filterValue = variableHandler.ПолучитьЗначение(parameters[3]).Значение.ToString();  keyColumns = ParseColumns(variableHandler, parameters, 4);  formOperand = variableHandler.ПолучитьЗначение(parameters[5]) as FormOperandType;  if (formOperand!=null)  {  form = formOperand.Значение as ДанныеФормы;  }  break;  }    var rows = form[tableCode].МатрицаЗначений.Строки.Values  .Where(x => FilterRow(x, filterColumns, filterValue))  .GroupBy(x => GenerateKey(x, keyColumns))  .ToDictionary(x => x.Key, y => y.ToList());    var data = new Dictionary<string,IТипОперанда>();    foreach (var row in rows)  {  var value = columns.Select(column => new ТипОперандаЧисленный(row.Value.Sum(y => y[column].ЗначениеЧисло)));    var operandValue = new ТипОперандаМассив(value);    data.Add(row.Key,operandValue);  }    return new DictionaryOperandType(data);  } |

##### Метод «GetTranscript»

Данный метод используется для формирования расшифровки увязки в протоколе проверки (Рисунок 107, Таблица 78).

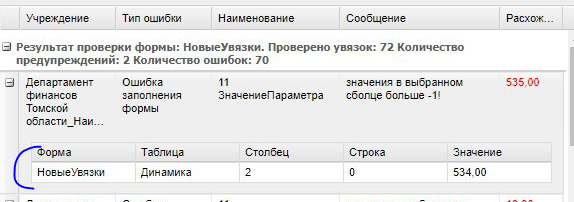


Рисунок 107 – Протокол проверки

Данный метод реализуйте только для функций, которые возвращают реальные данные, которые в дальнейшем используются при проверке. Для вспомогательных функций реализовывать не нужно. Если реализация не требуется, возвращать null.

Таблица 78 – Метод «GetTranscript»

| Параметр | Предназначение |
| --- | --- |
| index | Индекс обрабатываемого результата в массиве, если результат вычисления всех функций был массив. Игнорируйте, если ваша функция возвращает не массив. Иначе обрабатывайте только результат из массива по указанному индексу. |
| key | Ключ обрабатываемого результата в словаре, если результат вычисления всех функций был словарь. Игнорируйте, если ваша функция возвращает не словарь. Иначе обрабатывайте только результат из словаря по указанному ключу. |
| function | Модель функции со всеми вложенными функциями и параметрами. |
| value | Вычисленное значение функции |
| dataProvider | Провайдер для получения текущей формы, текущего компонента или для поиска компонентов или ОП по различным параметрам. |
| variableHandler | Обработчик переменных увязки. Используется в вычислителе выражений, для вычисления вложенных функций. |
| checkHandler | Вычислитель выражений. Используется для вычислений значений вложенных функций, т.к. в модели они не имею результата. |

|  |
| --- |
| public ЗначениеПеременнойЯчейкиФормы GetTranscript(int index,  string key,  ВыражениеФункция function,  IТипОперанда value,  ПровайдерДанныхФормДляРасчетаУвязок dataProvider,  БазовыйОбработчикПеременных variableHandler,  ВычислительВыражений checkHandler)  {  var parameters = function.Операнды;  var formCode = dataProvider.GetCurrentFormCode();    switch (parameters.Count)  {  case 3:  case 4:  case 5:  case 6:  if (parameters[parameters.Count-1] is ВыражениеФункция formFunction)  {  if(formFunction.СтроковоеПредставление.StartsWith("Форма"))  {  formCode = formFunction.Операнды[0].СтроковоеПредставление.Trim('\"');  }  }  break;  }    var tableCode = parameters[0].СтроковоеПредставление.Trim('\"');  var columns = new List<string>();  switch (checkHandler.Вычислить(parameters[1], variableHandler).Результат)  {  case ТипОперандаМассив keyColumnsOperand:  foreach (var keyColumn in keyColumnsOperand.ArrayValue)  {  var column = keyColumn.Значение.ToString();  columns.Add(column);  }  break;  case ТипОперандаСтроковый columnOperand:  columns.Add(columnOperand.Значение as string);  break;  }  if (value is DictionaryOperandType dictionaryOperandType)  {  if (dictionaryOperandType.DictionaryValue.ContainsKey(key))  {  if (dictionaryOperandType.DictionaryValue[key] is ТипОперандаМассив massive)  {  if (massive.ArrayValue.Count - 1 >= index)  {  return new ЗначениеПеременнойЯчейкиФормы(  function.СтроковоеПредставление,  string.Empty,  formCode,  tableCode,  $"{columns[index]}:{key}",  massive.ArrayValue[index].Значение.ToString(),  columns[index]);  }  }  }  else  {  return null;  }  }  return null;  } |

Для работоспособности подсказок для новых функций в Дизайнере в редакторе дерева создайте интерфейс «ICheckFunctionMetaDescription». Реализацию зарегистрируйте в IOC-контейнере:

|  |
| --- |
| /// <summary>Интерфейс для получения метаинформации функции увязки</summary>  public interface ICheckFunctionMetaDescription  {  /// <summary>Получить описание параметра функции</summary>  /// <param name="parameterNumber">Номер параметра функции</param>  /// <returns>Строковое описание параметра</returns>  string GetParameterFunctionDescription(int parameterNumber);    /// <summary>Свойство возвращающее отображаемое имя функции</summary>  string FunctionDisplayName { get; }    } |

|  |
| --- |
| Component.For<ICheckFunctionMetaDescription>().ImplementedBy<DynamicTableFinancialValuesFunctionMetaDescription>().LifestyleSingleton(), |

|  |
| --- |
| /// <summary>Метаинформация функции ФинансовыеЗначенияИзТаблицы</summary>  public class DynamicTableFinancialValuesFunctionMetaDescription : ICheckFunctionMetaDescription  {  /// <summary>Получить описание параметра функции</summary>  /// <param name="parameterNumber">Номер параметра функции</param>  /// <returns>Строковое описание параметра</returns>  public string GetParameterFunctionDescription(int parameterNumber)  {  switch (parameterNumber)  {  case 1:  return "Укажите код таблицы";  case 2:  return "Укажите столбец или несколько столбцов. Для указания нескольких столбцов используйте функцию Набор";  case 3:  return "Укажите графу для формирования ключа или для фильтрации строк или используйте функцию Форма для указания формы из которой необходимо получить значения";  case 4:  return "Укажите значение фильтра для фильтр графы или используйте функцию Форма для указания формы из которой необходимо получить значения";  case 5:  return "Укажите графу для формирования ключа или используйте функцию Форма для указания формы из которой необходимо получить значения";  case 6:  return "Используйте функцию Форма для указания формы из которой необходимо получить значения";  default:  return null;  }  }    /// <summary>Свойство возвращающее отображаемое имя функции</summary>  public string FunctionDisplayName => "ФинансовыеЗначенияИзТаблицы";  } |

Для реализации собственных операций между операндами создайте интерфейс «IOperandOperationsCalculator». Реализацию зарегистрируйте в IOC-контейнере с наименованием операции. Для наименования операции использовать только спецсимволы, иначе Система не распознает текст как операцию между операндами:

|  |
| --- |
| /// <summary>Интерфейс операций м/у двумя операндами</summary>  public interface IOperandOperationsCalculator  {  /// <summary>Произвести операцию над двумя операндами</summary>  /// <param name="firstOperand">Первый операнд</param>  /// <param name="secondOperand">Второй операнд</param>  /// <returns>Результат операции</returns>  IТипОперанда Calc(IТипОперанда firstOperand, IТипОперанда secondOperand);  } |

|  |
| --- |
| Component.For<IOperandOperationsCalculator>().ImplementedBy<EqualOperandOperationCalculator>().Named("=").LifestyleSingleton(), |

|  |
| --- |
| /// <summary>Калькулятор выполнения оперции равно двух операндов</summary>  public class EqualOperandOperationCalculator : IOperandOperationsCalculator  {  /// <summary>Фабрика калькуляторов</summary>  public ICalculatorFactory OperandTypeCalculator { get; set; }  /// <summary>Вычислить операцию</summary>  /// <param name="firstOperand">Первый операнд</param>  /// <param name="secondOperand">Второй операнд</param>  /// <returns>Результат выполнения операции</returns>  public IТипОперанда Calc(IТипОперанда firstOperand, IТипОперанда secondOperand)  {  var calculator = OperandTypeCalculator.CreateEqualsCalculator(firstOperand);  return calculator.Equal(firstOperand, secondOperand, false);  }  } |

Для реализации нового операнда создайте интерфейс «IТипОперанда», если новый операнд можно получать из значения параметра, то создайте интерфейс «IUserParameterOperandType<TParam>», вместо «IТипОперанда». Где «TParam» – тип параметра получаемый из пользовательского параметра:

|  |
| --- |
| /// <summary>Маркерный интерфейс для создания операнда из результата параметра</summary>  /// <typeparam name="TParam">Тип объекта который будет лежать в значении операнда</typeparam>  public interface IUserParameterOperandType<TParam> : IТипОперанда  {  }  /// <summary>Интерфейс операнда, используется для записи результата функций увязок</summary>  public interface IТипОперанда  {  /// <summary>Возвращает пустое значение или нет</summary>  bool ЗначениеПусто { get; }  /// <summary>Возвращает значение</summary>  object Значение { get; }  } |

**Примечание** – При использовании IUserParameterOperandType<TParam> реализацию зарегистрируйте в IOC-контейнере. Конструктор класса должен принимать значение операнда с названием value. Это необходимо для корректной инициализации операнда при получении значения параметра:

|  |
| --- |
| /// <summary>Класс для представления результирующего операнда с типом IОтчетныйПериод</summary>  [Serializable]  public sealed class ReportPeriodOperandType : IUserParameterOperandType<IОтчетныйПериод>  {  public ReportPeriodOperandType(IОтчетныйПериод value)  {  isNull = value == null;  this.value = value;  }  private bool isNull;    /// <summary>Возвращает пустое значение или нет</summary>  public bool ЗначениеПусто => isNull;  private object value;    /// <summary>Возвращает значение</summary>  public object Значение => value;  } |

|  |
| --- |
| Component.For<IUserParameterOperandType<IОтчетныйПериод>>().ImplementedBy<ReportPeriodOperandType>().LifestylePooled(), |

При реализации нового операнда создайте стандартные операции с операндами, реализовав следующие интерфейсы: «IOperandTypeArithmeticCalculator», «IOperandTypeEqualsCalculator», «IOperandTypeLogicalCalculator».

Данные реализации зарегистрируйте в IOC-контейнере.

### Пользовательские параметры

Пункт меню «Пользовательские параметры» представлен на рисунке ниже (Рисунок 108).

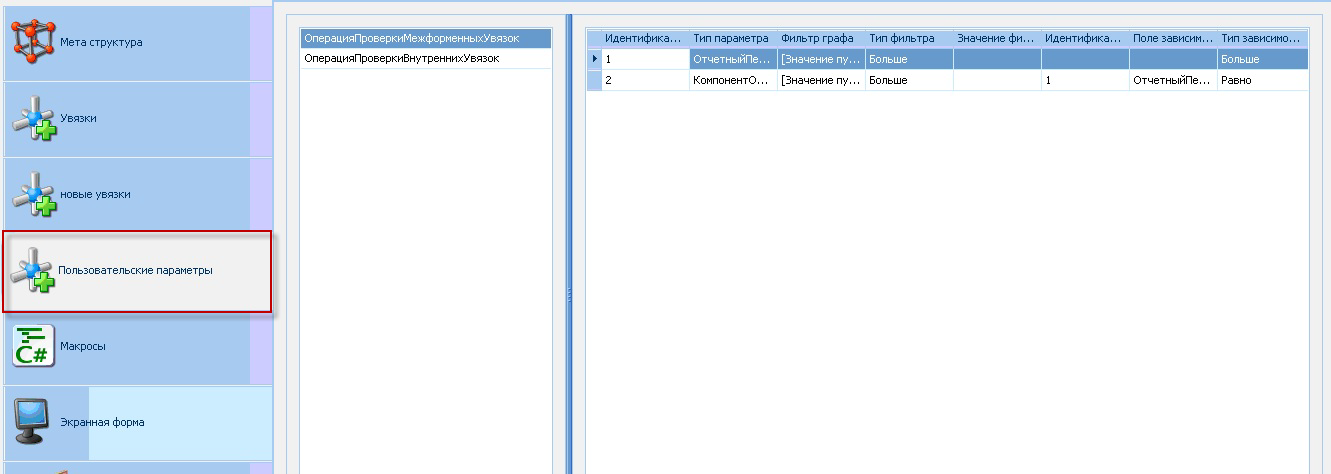


Рисунок 108 – Пункт меню «Пользовательские параметры»

Пользовательские параметры содержат следующие свойства:

* «Идентификатор параметра» – уникальный ключ параметра.
* «Тип параметра» – доступен выбор из выпадающего списка с возможными типами параметров (выбор отчетного периода, Выбор компонента отчетного период, выбор элемента цепочки, поле ввода числа, поле ввода текста);
* «Фильтр графа» – параметр для фильтрации списка возможных значений. Работает только с новым типом параметров: ОтчетныйПериод, КомпонентОтчетногоПериода, ЭлементЦепочки. Например, если необходимо настроить выбор отчетных периодов с датой начала > 01.01.2019, то из выпадающего списка выберите значение «ДатаНачала»;
* «Тип фильтра» – укажите тип фильтра. Доступны значения: «>», «<», «=», «Начинается на», «Заканчивается на», «содержит»;
* «Фильтр значение» – укажите значение фильтра. Например, если необходимо настроить выбор отчетных периодов с датой начала > 01.01.2019, то из выпадающего списка в поле «Тип фильтра» выберите значение «ДатаНачала», тип фильтра «Больше», а в поле «Фильтр значение» введите дату 01.01.2019;
* «Идентификатор зависимого параметра» – укажите идентификатор, от которого зависит указанный параметр;
* «Поле зависимости» – укажите поле в текущем параметре, по которому будет сформирован список на выбор;
* «Тип зависимости» – см. «Тип фильтра».

Пример:

В 1-ом параметре: Выбор отчетного периода.

Во 2-ом параметре: Выбор компонента отчетного периода. В поле Идентификатор зависимого параметра укажите 1 (1-ый параметр). Поле зависимости укажите ОтчетныйПериод (тип 1-ого параметра). Тип зависимости укажите Равно.

Результат: при формировании списка Компонентов на выбор, Система даст только те компоненты, которые есть в выбранном Отчетном периоде в параметре 1. Таким образом описывается зависимость между 1-ым и 2-ым параметрами.

### Примеры типовых увязок

Примеры типовых увязок представлены в таблице ниже (Таблица 79).

Таблица 79 – Примеры типовых увязок

| Увязка | Расшифровка |
| --- | --- |
| ФинансовыеЗначенияИзТаблицы("Динамика","1")>ФинансовыеЗначенияИзТаблицы("Динамика","2") | Сравнение 2 динамик по коду строки. Проверка 1 в 1.  Первая строка динамики левой части сравнивается с первой строкой динамики правой части.  Если в какой то части не хватает строк, то сравнение происходит с 0.  Например в левой динамике 10 строк, а правой 5 строк. Начиная с 6 строки, левая строка будет сравниваться с нулем. |
| ФинансовыеЗначенияИзТаблицы("Динамика","1","КлючГрафа")>ФинансовыеЗначенияИзТаблицы("Динамика","2","КлючГрафа") | Сравнение 2 динамических таблиц по ключам  Если в какой то части не хватает строк, то сравнение происходит с 0.  Например в левой динамике 10 строк по ключу, а правой 5 строк по ключам. Начиная с 6 строки, левая строка будет сравниваться с нулем. |
| ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),"0000120","КБК")>ФинансовыеЗначенияИзТаблицы("Динамика","2",,Набор("ЦС","ВР"),"0000120","КБК")  Левая часть:  Возьмем строки из таблицы Динамика в которых в склеенном значении столбцов ЦС и ВР находится значение 0000120, группируем по графе КБК, из полученных строк достанем значение графы 1.  Правая часть:  Возьмем строки из таблицы Динамика в которых в склеенном значении столбцов ЦС и ВР находится значение 0000120, группируем по графе КБК, из полученных строк достанем значение графы 2. | Сравнение 2 динамических таблиц по ключам предварительно отфильтрованных по значению фильтра.  Если в какой то части не хватает строк, то сравнение происходит с 0.  Например в левой динамике 10 строк по ключу, а правой 5 строк по ключам. Начиная с 6 строки, левая строка будет сравниваться с нулем. |
| ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"))>ФинансовыеЗначенияИзТаблицы("Динамика","2",Набор("ЦС","ВР"), Форма("НовыеУвязки",ЗначениеПараметра("2"))).  Конфигурация параметра:Описание: _scroll_external/attachments/params_conf-9bc00e49db9593e1dd7850c81976614cad23a83dea14336c4d7f4f91a180bfe6.jpg | Сравнение 2 динамических таблиц по ключам, таблица в правой части находится в другом компоненте. Компонент получаем из зависимого параметра.  Если в какой то части не хватает строк, то сравнение происходит с 0.  Например в левой динамике 10 строк по ключу, а правой 5 строк по ключам. Начиная с 6 строки, левая строка будет сравниваться с нулем. |
| ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР")) = ФинансовыеЗначенияИзТаблицы("Динамика","1",Набор("ЦС","ВР"),Форма("0503769"),Компонент(ПредыдущийОП())) + ФинансовыеЗначенияИзТаблицы("Динамика","2",Набор("ЦС","ВР"),Форма("0503769"),Компонент(ПредыдущийОП())) | Сравниваем динамическую таблицу по ключам текущей формы с суммой 2 таблицы с прошлого ОП.  Если в какой то части не хватает строк, то сравнение происходит с 0. |
| ФинансовыеЗначенияИзТаблицы("Динамика","1")>50 | Сравнение каждой строки динамической таблицы с числом константой |

### Операции с операндами

Операция с операндами – это операциями с двумя операндами результат которой является один операнд (Таблица 80).

Таблица 80 – Операции с операндами

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Операция | Левая часть | Правая часть | Результат операции | Описание операции | Примечание |
| = | 1)ТипОперандаСтроковый  2)ТипОперандаЦелочисленный  3)ТипОперандаЧисленный  4)ТипОперандаМассив  5)DictionaryOperandType | 1)  2)  3)  4)  5) | ТипОперандаЛогический,  DictionaryOperandType,  ТипОперандаМассив | Проверка на равенство значений двух операндов | Для DictionaryOperandType сравнение происходит по ключам.  Для ТипОперандаМассив сравнение происходит по номеру элемента.  Если ключ или номер отсутствует в правой или левой части сравнение происходит с 0 |
| + |  |  |  | Суммирование двух операндов |  |
| - |  |  |  | Вычитание значения правого операнда из левого операнда |  |
| \* |  |  |  | Умножение двух операндов. |  |
| / |  |  |  | Деление двух операндов. |  |
| > |  |  |  | Проверка, что левый операнд больше правого. |  |
| >= |  |  |  | Проверка, что левый операнд больше или равен правому. |  |
| < |  |  |  | Проверка, что левый операнд меньше правого. |  |
| <= |  |  |  | Проверка, что левый операнд меньше или равен правому. |  |
| <> |  |  |  | Проверка, что левый операнд не равен правому. |  |
| или |  |  |  | Логическое или для двух операндов. Возвращает истина если один из двух или оба операнда имеют значение «Истина» | Некорректное расхождение в протоколе |
| и |  |  |  | Логическое или для двух операндов. Возвращает истина если оба операнда имеют значение «Истина» | Некорректное расхождение в протоколе |

## Отправка сообщений, дублирование на почту, вложения

Сервис сообщений (messenger service) - это отдельное самостоятельное веб-приложение, отвечающее за обмен сообщениями внутри Системы.

У сервиса сообщений есть основные конфигурационные файлы:

* messengerService.config (см. п. 28.19.1.1);
* connection.config (см. п. 28.19.1.2);
* svody.config (см. п. 28.19.1.3).

### Конфигурационные файлы

#### messengerService.config

В данном конфигурационном файле прописывается способ работы с сообщениями и отправке их по почте. Способы представлены в таблицах ниже (Таблица 81, Таблица 82).

Таблица 81 – EWS

| Тэг | Внутри | Описание |
| --- | --- | --- |
| MailFrom | Почтовый адрес | Почтовый адрес, от которого будет отправлено сообщение по e-mail |
| User | Логин | Логин для почтовика |
| Password | Пароль | Пароль почтовика |
| Url | Ссылка | Адрес веб-службы EWS |
| Timeout | Количество миллисекунд | Время, через которое будет продублировано отправленное сообщение на почту |
| ExchangeVersion | Версия | Версия EWS |

Таблица 82 – SMTP

| Тэг | Внутри | Описание |
| --- | --- | --- |
| MailFrom | Почтовый адрес | Почтовый адрес, от которого будет отправлено сообщение по e-mail |
| Host | IP-адрес | IP-адрес до почтового сервиса |
| Port | Порт | Номер порта |
| Timeout | Количество миллисекунд | Время, через которое будет продублировано отправленное сообщение на почту |
| SslEnabled | True или False | SSL-протокол |
| LoginFrom | Логин | Логин для почтовика |
| PasswordFrom | Пароль | Пароль почтовика |
| DelSendedMessage | true или false | Флаг удаления сообщения из центра сообщений после отправки на почту |
| WithAttachments | true или false | Флаг возможности отправки сообщения с вложениями |

#### connection.config

В данном конфигурационном файле нем прописывается подключение к БД (Таблица 83).

Таблица 83 – Подключение к БД

|  |  |  |
| --- | --- | --- |
| Тэг | Внутри | Описание |
| DbDialect | Тип СУБД | Тип СУБД, например, Oracle, Npgsql |
| UserName | Логин | Имя пользователя БД |
| Password | Пароль | Пароль пользователя БД |
| Host | IP-адрес | IP-адрес сервера БД |
| Port | Порт | Номер порта |
| DbName | Имя БД | Наименование БД |

#### svody.config

После того, как сервис сообщений сконфигурирован, отредактируйте файл svody.config веб-приложения сводов.

Пример:

|  |
| --- |
| <Bars.MessengerService>  <Elements>  <!-- Адрес сервера, где развернут сервис отправки сообщений (без знака "/" в конце адреса, например "http://192.168.12.12") -->  <URL>https://svody3.bars.group/tula\_culture\_messengerservice</URL>  <!-- Период вызова сервиса для отправки сообщений (в секундах) -->  <Period>10</Period>  </Elements>  </Bars.MessengerService> |

### Центр сообщений

При отправке сообщений выполняется работа с БД со следующими таблицами (Таблица 84 – Таблица 86).

Таблица 84 – adresatsoobscheniya (Адресат сообщения)

| Поле | Тип | Описание |
| --- | --- | --- |
| ADRESAT\_ID | Guid | Адресат сообщения, пользователь |
| SOOBSHAENIE\_ID | Guid | Идентификатор сообщения |
| PROCHITANO | bool | Прочитано ли сообщение |
| UDALENO | bool | Удалено ли сообщение |
| OTPRAVITNAEMAIL | bool | Отправлять по электронной почте |
| SOOBSHAENIEOTPRAVLENO | bool | Отправлено ли сообщение |

Таблица 85 – moisoobshaeniya (Мои сообщения)

| Поле | Тип | Описание |
| --- | --- | --- |
| TELO | Файл | Устанавливает и возвращает основное содержание сообщения в виде файла |
| OTKOGO\_ID | Guid | Устанавливает и возвращает отправителя сообщения |
| DATEOTPR | DateTime | Устанавливает и возвращает дату отправления сообщения |
| TEMA | string | Устанавливает и возвращает строку-заголовок сообщения |
| OTPRAVLENO | bool | Устанавливает и возвращает флаг, определяющий отправлено ли сообщение |
| UDALENO | bool | Устанавливает и возвращает флаг, определяющий удалено ли сообщение |
| MESSAGEHASH | string | Устанавливает и возвращает уникальный код сообщения (хеш-код) |
| SENDREADCONFIRMATION | bool | Устанавливает и возвращает необходимость запроса уведомления о прочтении |
| TARGETS | string | Адресаты сообщения строкой через точку с запятой |

Таблица 86 – vlozheniesoobschenia (Вложение сообщения)

| Поле | Тип | Описание |
| --- | --- | --- |
| FAIL | Файл | Устанавливает и возвращает вложенный в сообщение файл |
| SOOBSHAENIE | Guid | Устанавливает и возвращает сообщение, которому принадлежит вложение |
| REMOTEFILEPATH | string | Имя файла на файл-сервере |
| ATTACHMENTSTORAGETYPE | AttachmentsStorageType | Место хранения файлов (1 - в базе данных, 2 - на жестком/сетевом диске) |
| FILESIZE | long | Размер файла в байтах |
| NAME | string | Имя файла |

### Примеры

В центре сообщений есть возможность выбора оператора, которому можно отправить сообщение. Также есть функция дублирования на почту. Ей можно воспользоваться в случае, если у оператора в атрибутах прописан e-mail.

Пример самостоятельной отправки сообщения:

|  |
| --- |
| public async Task ОтправитьСообщениеAsync()  {  // формирование сообщения  var сообщение = new МоиСообщения();  сообщение.Тема = "Тема сообщения";  сообщение.ТелоСообщения = "Тело сообщения";  сообщение.ДатаОтпр = DateTime.Now;  сообщение.ОтКого = МенеджерПользователей.ТекущийПользователь;  сообщение.Отправлено = true;  сообщение.ХешСообщения = КонтрольнаяСумма.ПолучитьПоСтроке(сообщение.ТелоСообщения);    // формирование списка операторов  // в данном случае выбираются те, у которых заполнено поле Email  var списокОператоров = DataStore.GetQuery<ОператорУчреждения>()  .Where(x => !string.IsNullOrEmpty(x.Email))  .FillFields()  .ToList();    var пакетАдресатов = new ПакетХранимыхОбъектов();  var адресаты = new StringBuilder();    foreach (var оператор in списокОператоров)  {  var адресат = new АдресатСообщения(оператор.СистемныйПользователь, сообщение)  {  ОтправитьНаEmail = true // или false, если не нужно дублировать на Email  };  адресаты.Append(оператор.СистемныйПользователь.Наименование);  адресаты.Append("; ");  пакетАдресатов.Добавить(адресат);  }    сообщение.Targets = адресаты.ToString();    await DbConnectionPool.InTransactionAsync(  async tr =>  {  // здесь происходит запись в таблицу moisoobshaeniya  await сообщение.СохранитьAsync();    await using (var stream = new MemoryStream(Encoding.UTF32.GetBytes(сообщение.ТелоСообщения)))  {  await Типы.Файл.SaveStreamToFieldAsync(stream, сообщение, "Тело");  }    var objectToDelete = DataStore.GetQuery<АдресатСообщения>().FirstOrDefault(a => a.СообщениеИдентификатор == сообщение.Идентификатор);    if (objectToDelete != null)  {  var objects = new ПакетХранимыхОбъектов();  objects.Добавить(objectToDelete);  var deleteCmd = new КомандаНаУдаление(objects);    var deleteResult = await deleteCmd.ВыполнитьAsync();  if (deleteResult.Ошибка != null)  {  throw deleteResult.Ошибка;  }  }    // здесь происходит запись в таблицу adresatsoobscheniya  await пакетАдресатов.СохранитьОбъектыAsync();  }  );  } |

Пример добавления вложения:

|  |
| --- |
| public async Task ДобавитьВложенияСообщенияAsync(МоиСообщения сообщение, List<string> путиДоФайлов)  {  if (сообщение != null && сообщение.Состояние != СостояниеХранимогоОбъекта.Новый)  {  var attachmentsService = AttachmentsServicesFactory<ВложениеСообщения>.GetService();  var attachmentStorageType = AttachmentsStorageSetting.AttachmentsStorage.Value.AttachmentsStorageType;  await DbConnectionPool.InTransactionAsync(  async tr =>  {  foreach (var путь in путиДоФайлов)  {  await using var fileStream = File.Open(путь, FileMode.Open);  var имяФайла = Path.GetFileName(путь);  var вложение = new ВложениеСообщения  {  Наименование = имяФайла,  Сообщение = сообщение,  AttachmentStorageType = attachmentStorageType,  FileSize = fileStream.Length  };  // здесь происходит запись в таблицу vlozheniesoobschenia  await attachmentsService.SaveAttachmentAsync(вложение, fileStream);  }  });  }  else  {  throw new Exception("Сообщение об ошибке");  }  } |

Пример отправки сообщения с помощью Системы сообщений:

|  |
| --- |
| public async Task ОтправитьСообщениеСПомощьюСистемыСообщений()  {  var сообщение = new МоиСообщения();    // сформировать сообщение    // sendToEmail и saveAsDraft по умолчанию false  await СистемаСообщений.SendAsync(сообщение, sendToEmail: true, saveAsDraft: false);    var вложения = new List<ВложениеСообщения>();    // сформировать список вложений    await СистемаСообщений.SetAttachementsWebAsync(сообщение, вложения);  } |

## Расширение Фронтенд Системы версии 5.0 на проектном уровне

### Темы

Темы необходимы для изменения внешнего вида приложения или базового поведения компонентов.

#### Создание темы

Создание темы описано в инструкциях:

* **https://docs.sencha.com/extjs/7.1.0/guides/core\_concepts/theming.html;**
* **https://docs.sencha.com/extjs/7.1.0/guides/core\_concepts/classic\_theming.html.**

Также для редактирования темы можно скопировать папку существующей темы и изменять её.

#### Настройка сборки приложения с темой

В папку **BARS.Svody.Web.ExtJsClient\packages\local** добавьте папку с необходимой темой.

В папке BARS.Svody.Web.ExtJsClient, находящейся в папке src, найдите файл app.json и отредактируйте его.

В секцию builds добавьте описание темы.

Описание темы:

|  |
| --- |
| "Наименование манифеста": {  "theme": "имя вашей темы"  } |

**Примечание** – Имя темы в описании должно совпадать с наименованием папки.

Пример:

|  |
| --- |
| "builds": {  "svody" : {  "theme": "theme-svody"  },  "svody-dark": {  "theme": "theme-svody-dark"  },  "svody2": {  "theme": "theme-svody2"  }  } |

В папке BARS.Svody.Web.ExtJsClient вызовите консоль администрирования (CMD). Для этого в строке, где указан полный путь, введите «cmd» и нажмите на клавишу <Enter> (Рисунок 30).

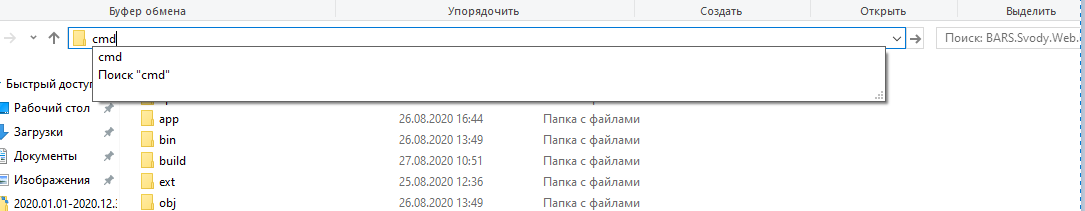


Рисунок 109 – Вызов консоли администрирования

Для отладки и локального использования:

sencha app build development;

sencha app watch svody-project (тут можно указать любой из манифестов указанных в секции builds).

Для сборки для использования в production:

sencha app build production.

При development-сборке никакие файлы копировать не нужно

При production сборке все необходимые файлы загрузятся в папку **BARS.Svody.Web.Host\wwwroot** (Рисунок 110).

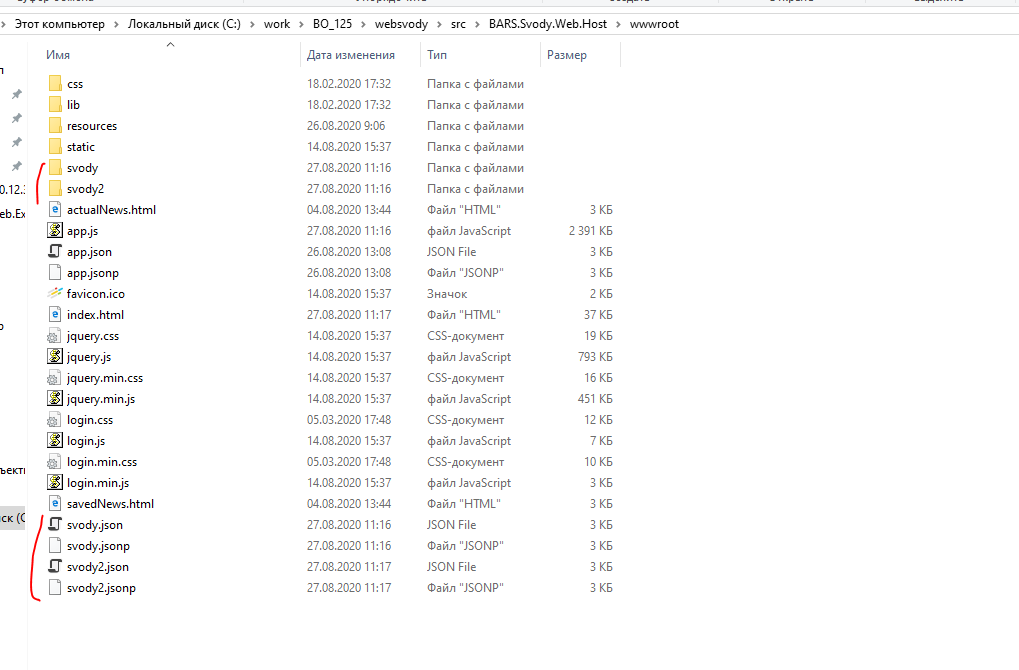


Рисунок 110 – Папка BARS.Svody.Web.Host\wwwroot

В production-приложение добавьте папки, соответствующие названиям манифестов (в примере svody, svody-dark и svody2), json и jsonp файлы с названиями как у манифеста.

Если в теме есть JavaScript-код (overrides или src), в папку приложения добавьте файл app.js, в нем находится весь js-код приложения.

После добавления всех необходимых файлов, отредактируйте файл svody.config.

В нем появится новая секция Bars.Themes, добавьте строку с описанием темы.

Строка с описанием темы:

|  |
| --- |
| <Theme Name="Наименование манифеста" DisplayName = "Отображаемое имя в списке" PreviewImageCls = "Название класса с картинкой для идентификации темы" /> |

Если в кастомной теме нет картинки, которая должна отображаться при выборе темы, то параметр PreviewImageCls не указывается (Рисунок 111).

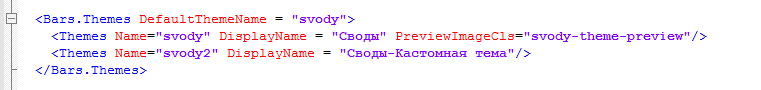


Рисунок 111 – Параметр PreviewImageCls

В атрибуте DefaultThemeName указывается используемая по умолчанию тема. Она используется в следующих случаях:

* если пользователь не выбирал тему из списка;
* если выбранная тема перестала быть валидной;
* если выбранная тема не валидная;
* если отключена настройка «Отслеживать и сохранять настройки пользователей».

После этого перезапустите приложение, тема появится в списке для выбора.

**Примечание** – Описанные выше действия необходимо выполнять при каждом обновлении. Так как sencha компонует все стили в несколько css-файлов, туда включаются стили из темы и стили из приложения, которые могут затронуть необходимые стили.

### Custom.scss

Custom.scss – Это scss файл, в котором можно размещать свои стили, которые будут загружены вне зависимости от выбранной пользователем темы.

Например, такой стиль может понадобиться для картинки предпросмотра темы (PreviewImageCls).

#### Рекомендации по использованию

Создайте файл с аналогичным названием в теме, например в **sass\src**.

Перед сборкой приложения замените этим файлом файл **BARS.Svody.Web.ExtJsClient\app\Custom.scss.** Файл будет автоматически подключен к приложению.

### apiJs

apiJs – Это точка расширения для загрузки кастомного js кода. Подробное описание представлено в п. 28.11.

## Регистрация SOAP-сервисов

Реализована возможность регистрации SOAP-сервисов из прикладных API в качестве endpoint-ов платформы.

Пример секции Bars.SoapServices в файле svody.config:

|  |
| --- |
| <Bars.SoapServices>  <Entries Name="SampleService" Type="Api.LegacyWcf.SampleService, Api.LegacyWcf" Endpoint="/SampleService.asmx"/>  <Entries Name="SampleService2" Type="Api.LegacyWcf.SampleService2, Api.LegacyWcf" Endpoint="/SampleService2.asmx"/>  </Bars.SoapServices> |

Атрибуты секции Bars.SoapServices представлены в таблице ниже (Таблица 87).

Таблица 87 – Атрибуты секции Bars.SoapServices

| Атрибут | Назначение |
| --- | --- |
| Name | Наименование сервиса |
| Type | Полное название типа по которому генерируется WDSL-описание сервиса и которому будут переданы запросы |
| Endpoint | Конечная точка маршрута сервиса |

**Пример** – *Пример кода. Реализована сборка с названием Api.LegacyWcf.*

Код сервиса «SampleService»:

|  |
| --- |
| using System.Runtime.Serialization;  using System.ServiceModel;    namespace Api.LegacyWcf  {  [ServiceContract]  public class SampleService  {  [OperationContract]  public void VoidMethod()  {  }    [OperationContract]  public ResponseDto GetResult(RequestDto requestDto)  {  return new ResponseDto  {  Name = "Test" + requestDto.Id,  Id = requestDto.Id  };  }  }    [DataContract]  public class RequestDto  {  [DataMember]  public int Id { get; set; }  }    [DataContract]  public class ResponseDto  {  [DataMember]  public string Name { get; set; }    [DataMember]  public int Id { get; set; }  }  } |

Код сервиса «SampleService2»:

|  |
| --- |
| using System.Runtime.Serialization;  using System.ServiceModel;    namespace Api.LegacyWcf  {  [ServiceContract]  public interface ISampleService  {  [OperationContract]  void TestMethod();  }    public class SampleService2 : ISampleService  {  public void TestMethod()  {  }  }  } |

Для корректной работы сервиса на самом сервисе и типах используемых в качестве входных и выходных значений должны быть использованы соответствующие атрибуты. Необходимые атрибуты представлены в таблице ниже (Таблица 88).

Таблица 88 – Атрибуты, необходимые для корректной работы сервиса

|  |  |
| --- | --- |
| Атрибут | Размещение |
| System.ServiceModel.ServiceContractAttribute | На классе реализующем сервис, либо на интерфейсе описывающем его |
| System.ServiceModel.OperationContractAttribute | На всех методах класса сервиса или интерфейса которые должны быть доступны для вызова |
| System.Runtime.Serialization.DataContractAttribute | На всех классах моделей возвращаемых и принимаемых значений |
| System.Runtime.Serialization.DataMemberAttribute | На всех свойствах моделей возвращаемых и принимаемых значений |

Если сервис корректно добавлен в секцию Bars.SoapServices и все необходимые атрибуты использованы сервис будет доступен по ссылкам: **https://url\_приложения/SampleService.asmx** и **https://url\_приложения/SampleService2.asmx** соответственно.

# Аналитические выборки

## Использование классов в аналитических выборках

Для использования вложенных классов в аналитических выборках, в XML-структуре укажите наименование вложенного класса. В теге <Классы> указываются все вложенные классы.

**Примечание** – Необходимо учесть, что сторонние модификаторы класса кроме public не поддерживаются, для методов класса таких ограничений нет.

|  |
| --- |
| <Классы>  <Класс>Test</Класс>  </Классы> |

Пример:

|  |
| --- |
| //<ПроизвольнаяВыборка ТипПредставления="OLAP" Группа="Расходы" ДопГруппа="" СборкаПоОтчетномуПериоду="False" ЗагружатьДанныеРодительскихУчреждений="False" РасчетДляВыборки="">  <Поля>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="A1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Статус</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="B3" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">ГРБС</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="B1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Наименование показателя</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="B2" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Код строки</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="F1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Код группы вида расхода</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="G1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Статус формы Приложение 5</Поле>  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтолбец" РежимФормированияИтогов="БезИтогов" ИмяПеременной="G2" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Столбец</Поле>  <Поле ТипЗначения="Финансовый" ТипПоляOLAP="Ресурс" РежимФормированияИтогов="БезИтогов" ИмяПеременной="V1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения=""/>  </Поля>  <ДанныеСборки>  <Строки />  </ДанныеСборки>  <Макросы>  <Макрос Автозапуск="False">ПолучитьСписокЗапросовФорм</Макрос>  <Макрос Автозапуск="True">Макрос0</Макрос>  </Макросы>  <Классы>  <Класс>Test</Класс>  </Классы>  </ПроизвольнаяВыборка> |

Пример выборки представлен ниже (Рисунок 112).

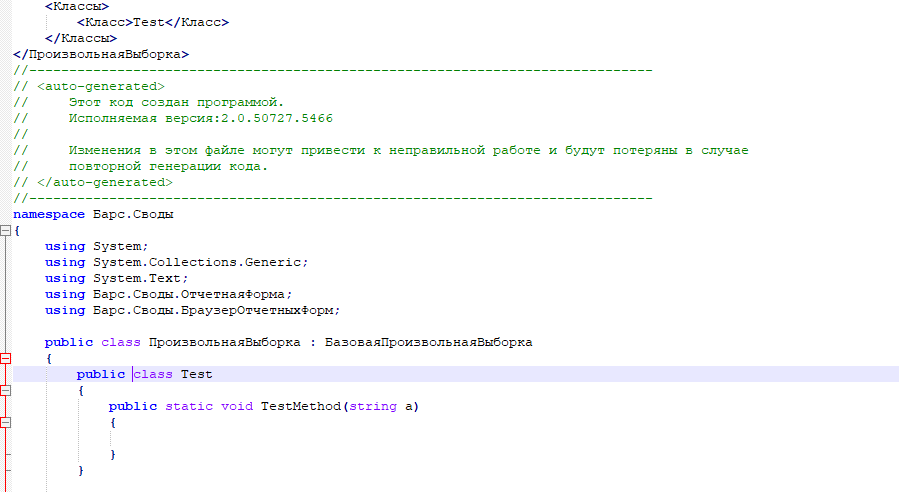


Рисунок 112 – Пример выборки

## Создание аналитической выборки

Создание аналитической выборки начинается с задания структуры.

Задание структуры аналитической выборки:

|  |
| --- |
| <ПроизвольнаяВыборка ТипПредставления="Линейный/OLAP" Код="Код выборки" Наименование="Наименование выборки" Группа="Группа выборки" ДопГруппа="Доп. группа выборки" СборкаПоОтчетномуПериоду="True/False" ЗагружатьДанныеРодительскихУчреждений="True/False" РасчетДляВыборки="">    <Поля>  Здесь можно задать поля, которые зададут структуру будущей аналитической выборки.  Пример:  <Поле ТипЗначения="Строка" ТипПоляOLAP="ИзмерениеСтрока" РежимФормированияИтогов="БезИтогов" ИмяПеременной="I1" Точность="2" ВыражениеДляВычисления="" ГруппаРасчета="" КодРасчетаДляВыборки="" ПеременнаяРасчетаВыборки="" ЗаголовокОбъединения="">Рекомендованные значения</Поле>  </Поля>    <ДанныеСборки>  <Строки />  </ДанныеСборки>    <Макросы>  <Макрос Автозапуск="True"> Макрос0 </Макрос> - макрос, используемый для сбора данных для аналитической выборки.  Далее необходимо описать все макросы, разработанные в ходе написания выборки.  </Макросы>      <НастройкиЭкспорта ПечататьЗаголовкиНаКаждойСтранице="True/False" ПечататьЗаголовкиПолейФильтра="True/False" ПечататьЗаголовкиПолейИзмеренийСтолбцов="True/False" ПечататьЗаголовкиПолейИзмеренийСтрок="True/False" ПечататьЗаголовкиПолейРесурсов="True/False" ПечататьОписаниеВыборки="True/False" />  </ПроизвольнаяВыборка> |

Данные для задания структуры выборки представлены ниже (Таблица 89).

Таблица 89 – Данные для задания структуры выборки

| Тип Значения | Тип поля OLAP | Режим Формирования Итогов |
| --- | --- | --- |
| Финансовый | ИзмерениеСтолбец | Число |
| Строка | ИзмерениеСтрока | Сумма |
| ДатаВремя | Ресурс | Минимум |
| Целое | ПолеФильтрации | Максимум |
| Число | СкрытоеПоле | Среднее |
| Guid |  | СтандартноеОтклонение |
|  |  | СтандатноеОтклонениеПопуляции |
|  |  | Дисперсия |
|  |  | ДисперсияПопуляции |
|  |  | Другое |
|  |  | БезИтогов |
|  |  | ВыражениеПоСумме |

Описание структуры полей в будущей выборке может выполняться в коде.

Описание структуры полей в будущей выборке:

|  |
| --- |
| public void ДобавитьПоля()  {  List<ПолеПроизвольнойВыборки> поля = new List<ПолеПроизвольнойВыборки>();  ПолеПроизвольнойВыборки полеПроизвольнойВыборки;    if (ПеременнаяОбъявлена("поля"))  {  поля = (List<ПолеПроизвольнойВыборки>)ПолучитьЗначениеПеременной("поля");  this.Описание.Поля = поля;  }  else  {  полеПроизвольнойВыборки = new ПолеПроизвольнойВыборки();  полеПроизвольнойВыборки.Заголовок = "Наименование";  полеПроизвольнойВыборки.ИмяПеременной = "A1";  полеПроизвольнойВыборки.ТипЗначения = ТипЗначенияПоляOLAPКуба.Строка;  полеПроизвольнойВыборки.ТипПоляOLAP = ТипПоляOLAPКуба.ИзмерениеСтрока;  полеПроизвольнойВыборки.РежимФормированияИтогов = РежимФормированияИтоговСтолбцаАналитическойТаблицы.БезИтогов;  полеПроизвольнойВыборки.Точность = 0;  поля.Add(полеПроизвольнойВыборки);    //описываем все следующие поля  //...    УстановитьЗначениеПеременной("поля", поля);  this.Описание.Поля = поля;  } |

Если описание полей выполняется в коде, вызовите метод «ДобавитьПоля()» в теле метода «Макрос0()».

Пример структуры выборки представлен на рисунке ниже (Рисунок 113).

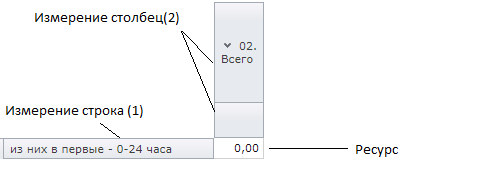


Рисунок 113 – Пример структуры выборки

Следующим шагом разработайте макрос, описывающий логику сбора данных для формирования выборки (Макрос0 в структуре выборки). Макрос должен возвращать List<object[]>. Запись результатов должна соответствовать структуре, которая была задана в создаваемой выборке.

Общий алгоритм добавления значения в отдельный ресурс выборки:

* первым значением передается наименование строки, в которую будет передаваться значение. Если измерение строки больше одного, то необходимо передавать наименование строк в той последовательности, в которой необходимо отображать их в конечной выборке;
* аналогично передаются столбцы. Если столбцы имеют общее первое измерение, необходимо передавать его каждый раз, меняя значение наименования второго и последующих измерениях;
* последним передается значение, которое необходимо записать в ресурс, обозначенный строкой и столбцом, обозначенными выше.

Например, структура имеет вид представленный на рисунке (Рисунок 114).

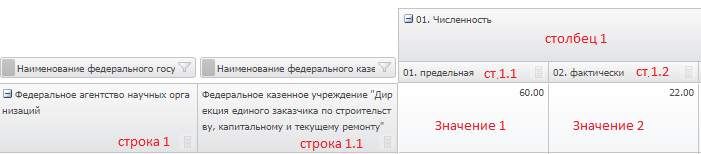


Рисунок 114 – Структура аналитической выборки

Запись значений в таком случае будет выполняться следующим образом:

Запись значений:

|  |
| --- |
| //Инициализируем список  List<object[]> результат = new List<object[]>();    //Добавление Значения 1  результат.Add(new Object[]  {  строка1,  строка1.1,  столбец1,  столбец1.1,  значение1  });      //Добавление Значения 2  результат.Add(new Object[]  {  строка1,  строка1.1,  столбец1,  столбец1.2,  значение2  }); |

После получения данных и добавления полей добавьте следующие методы:

Добавление методов:

|  |
| --- |
| //Получаем список необходимых форм, используемых в методе Макрос0()  public override List<ЗапросНаПолучениеФормы> ПолучитьСписокЗапросовФорм(string ИмяОперации)  {  List<ЗапросНаПолучениеФормы> результат = new List<ЗапросНаПолучениеФормы>();  ЗапросНаПолучениеФормы запрос = new ЗапросНаПолучениеФормы("#кодФормы#");  запрос.ЗагружатьСправочники = false;  результат.Add(запрос);  return результат;  }    //Получаем данные, возвращаемые методом Макрос0()  public override List<object[]> ПолучитьДанные()  {  List<object[]> результат = new List<object[]>();  результат.AddRange(Макрос0());  return результат;  }    //Метод получения идентификаторов форм для последующего заполнения  public override List<string> ПолучитьСписокИдентификаторовФорм()  {  List<string> результат = new List<string>();  return результат;  }    //Метод используется для заполнения форм  public override void ЗаполнитьФормы(List<ДанныеФормы> Формы)  {  }      protected override void ОсвободитьДополнительныеРесурсы()  {  base.ОсвободитьДополнительныеРесурсы();  } |

Данные методы необходимы для формирования итоговой выборки в классе «ПровайдерПроизвольнойВыборки».

Лист регистрации изменений

| Изм. | Версия Системы | Версия документа | Дата внесения изменений | Автор изменений | Краткое описание изменений |
| --- | --- | --- | --- | --- | --- |
| 1 | 5.1 | 1 | 23.05.2022 | Пырихина Е.В. | Документ создан |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Override – замещение, используется для внедрения дополнительных файлов с измененным программным кодом. Таким образом, внутренние файлы с кодом не изменяются, а необходимые изменения считываются с новых файлов. [↑](#footnote-ref-1)
2. \* В кастомном контроллере необходимо реализовать все те же методы с той же сигнатурой, что и в контроллере карточки учреждения платформы DepartmentController, и по необходимости изменить реализацию необходимых методов [↑](#footnote-ref-2)